There is also a random-number generator that uses the computer's disk drive [439]. It measures the time required to read a disk block and uses the variation in that time as a random number source. It filters the timing data to remove structure that comes from quantization, then applies a fast Fourier transform to vectors of the numbers. This removes bias and correlation. Finally, it uses the spectral angles for frequencies in $(0, \pi)$, normalized to the unit interval, as the random bits. A large part of the variation in disk rotation speed is caused by air turbulence, so there is randomness in the system. There are caveats, though. If you keep too many bits of the output, you are using the fast Fourier transform as a random-number generator and risk predictability. And it's best to read the same disk block over and over, so that your filtering doesn't have to remove structure that comes from the disk-scheduler. An implementation of this system was able to collect about 100 bits per minute [439].

Using the Computer's Clock

If you want a single random bit (or even a few), take the least significant bit from any clock register. This might not be terribly random in a UNIX system because of various potential synchronizations, but it works on some personal computers.

Beware of getting too many bits this way. Executing the same subroutine several times in succession could easily skew bits generated in this manner. For example, if each bit generation subroutine takes an even number of clock ticks to execute, you will get an endless stream of the same bit out of the generator. If each subroutine takes an odd number of clock ticks to execute, you will get an endless stream of alternating bits out of the generator. Even if the resonance isn't this obvious, the resultant bit stream will be far from random.

One random-number generator works this way [918]:

> Our truly random number generator . . . works by setting an alarm and then incrementing a counter register rapidly in the CPU until an interrupt occurs. The contents of the register are then XORed with the contents of an output buffer byte (truncating the register's data to 8 bits). After each byte of the output buffer is filled, the buffer is further processed by doing a right, circular shift of each character by 2 bits. This has the effect of moving the most active (and random) least significant bits into the most significant positions. The entire process is then repeated 3 times. Finally each character of the buffer has been touched by the two most random bits of the counter register after interrupts. That is $4n$ interrupts have occurred where n is the number of desired random bytes.

This method is very sensitive to the randomness of system interrupts and the granularity of the clock. The output looked pretty good when tested on real UNIX machines.

Measuring Keyboard Latency

People's typing patterns are both random and nonrandom. They are nonrandom enough that they can be used as a means of identification, but they are random enough that they can be used to generate random bits. Measure the time between

successive keystrokes, then take the least significant bits of those measurements. These bits are going to be pretty random. This technique may not work on a UNIX terminal, since the keystrokes pass through filters and other mechanisms before they get to your program, but it will work on most personal computers.

Ideally, you only want to collect one random bit per keystroke. Collecting more may skew the results, depending on how good a typist is sitting at the keyboard. This technique is limited, though. While it's easy to have someone type 100 words or so when it is time to generate a key, it isn't reasonable to ask the typist to type a 100,000-word essay to generate a keystream for a one-time pad.

Biases and Correlations

A major problem with all these systems is that there could be nonrandomness in the generated sequence. The underlying physical processes might be random, but many kinds of measuring instruments are between the digital part of the computer and the physical process. Those instruments could easily introduce problems.

A way to eliminate **bias**, or skew, is to XOR several bits together. If a random bit is biased toward 0 by a factor e, then the probability of 0 can be written as:

$$P(0) = .5 + e$$

XORing two of these bits together yields:

$$P(0) = (.5 + e)^2 + (.5 - e)^2 = .5 + 2e^2$$

By the same calculation, XORing 4 bits together yields:

$$P(0) = .5 + 8e^4$$

XORing m bits will exponentially converge to an equal probability of 0 and 1. If you know the maximum bias you are willing to accept for your application, you can calculate how many bits you need to XOR together to get random bits below that bias.

An even better method is to look at the bits in pairs. If the 2 bits are the same, discard them and look at the next pair. If the 2 bits are different, take the first bit as the output of the generator. This eliminates bias completely. Other techniques for reducing bias use transition mappings, compression, and fast Fourier transforms [511].

The potential problem with both methods is that if there is a **correlation** between adjacent bits, then these methods will increase the bias. One way to correct this is to use multiple random sources. Take four different random sources and XOR the bits together; or take two random sources, and look at those bits in pairs.

For example, take a radioactive source and hook a Geiger counter to your computer. Take a pair of noisy diodes and record as an event every time the noise exceeds a certain peak. Measure atmospheric noise. Get a random bit from each and XOR them together to produce the random bit. The possibilities are endless.

The mere fact that a random-number generator has a bias does not necessarily mean that it is unusable. It just means that it is less secure. For example, consider the problem of Alice generating a triple-DES 168-bit key. All she has is a random-bit generator with a bias toward 0: It produces 55 percent 0s and 45 percent 1s. This means that there are only 0.99277 bits of entropy per key bit, as opposed to 1 bit of

entropy if the generator were perfect. Mallory, trying to break the key, can optimize his brute-force search to try the most probable key first (000 . . . 0), and work toward the least probable key (111 . . . 1). Because of the bias, Mallory can expect to find the key in 2^{109} attempts. If there were no bias, Mallory would expect to make 2^{111} attempts. The resultant key is less secure, but not appreciably so.

Distilling Randomness

In general, the best way to generate random numbers is to find a whole lot of seemingly random events and distill randomness from them. This randomness can then be stored in a pool or reservoir that applications can draw on as needed. One-way hash functions are ready-made for the job; they're fast, so you can shovel quite a bit through them without worrying too much about performance or the actual randomness of each observation. Hash almost anything you can find that has at least some randomness. Try:

- A copy of every keystroke
- Mouse commands
- The sector number, time of day, and seek latency for every disk operation
- Actual mouse position
- Number of current scanline of monitor
- Contents of the actually displayed image
- Contents of FATs, kernel tables, and so on
- Access/modify times of /dev/tty
- CPU load
- Arrival times of network packets
- Input from a microphone
- /dev/audio without a microphone attached

If your system uses separate crystal oscillators for its CPU and time-of-day clocks, try reading the time of day in a tight loop. On some (but not all) systems this will reflect the random phase jitter between the two oscillators.

Since much of the randomness in these events is in their timing, use the most finely grained time-of-day clock you can find. A standard PC uses an Intel 8254 clock chip (or equivalent) driven at 1.1931818 megahertz, so reading the counter register directly gives you 838-nanosecond resolution. To avoid skewing the results, avoid taking your event samples on a timer interrupt.

Here is the process in C with MD5 (see Section 18.5) as the hash function:

```
char Randpool[16];

/* Call early and call often on a wide variety of random or semi-
 * random system events to churn the randomness pool.
```

```
 * The exact format and length of randevent doesn't matter as long as
 * its contents are at least somewhat unpredictable.
 */
void churnrand(char *randevent,unsigned int randlen)
{
     MD5_CTX md5;
     MD5Init(&md5);
     MD5Update(&md5,Randpool,sizeof(Randpool));
     MD5Update(&md5,randevent,randlen);
     MD5Final(Randpool,&md5);
}
```

After calling churnrand() enough to build up sufficient randomness in Randpool, you can now generate random bits from it. MD5 again comes in handy, this time as a counter-mode pseudo-random byte-stream generator.

```
long Randcnt;
void genrand(char *buf,unsigned int buflen)
{
     MD5_CTX md5;
     char tmp[16];
     unsigned int n;

     while(buflen != 0) {
          /* Hash the pool with a counter */
          MD5Init(&md5);
          MD5Update(&md5,Randpool,sizeof(Randpool));
          MD5Update(&md5,(unsigned char *)&Randcnt,sizeof(Randcnt));
          MD5Final(tmp,&md5);
          Randcnt++; /* Increment counter */

          /* Copy 16 bytes or requested amount, whichever is less,
          * to the user's buffer */
          n = (buflen < 16) ? buflen : 16;
          memcpy(buf,tmp,n);
          buf += n;
          buflen -= n;
     }
}
```

The hash function is crucial here for several reasons. First, it provides an easy way to generate an arbitrary amount of pseudo-random data without having to call churnrand() each time. In effect, the system degrades gracefully from perfect to practical randomness when the demand exceeds the supply. In this case it becomes *theoretically* possible to use the result from one genrand() call to determine a previous or subsequent result. But this requires inverting MD5, which is computationally infeasible.

This is important since the routine doesn't know what each caller will do with the random data it returns. One call might generate a random number for a protocol that is sent in the clear, perhaps in response to a direct request by an attacker. The very next call might generate a secret key for an unrelated session that the attacker

wishes to penetrate. Obviously, it is very important that an attacker not be able to deduce the secret key from the nonce.

One problem remains. There must be sufficient randomness in the Randpool[] array before the first call to genrand(). If the system has been running for a while with a local user typing on the keyboard, no problem. But what about a standalone system that reboots automatically without seeing any keyboard or mouse input?

This is a tough one. A partial solution would require the operator to type for a while after the very first reboot, and to create a seed file on disk before shutting down to carry the randomness in Randseed[] across reboots. But do not save the Randseed[] array directly. An attacker who steals this file could determine all of the results from genrand() after the last call to churnrand() prior to the file being created.

The fix to this problem is to hash the Randseed[] array before storing it, perhaps by just calling genrand(). When the system reboots, you read in the seed file, pass it to churnrand(), then promptly destroy it. Unfortunately, this does not deal with the threat of someone stealing the seed file between reboots and using it to guess future values of the genrand() function. I see no solution to this problem other than to wait until enough external random events have taken place after a reboot before allowing genrand() to produce results.

CHAPTER 18

One-Way Hash Functions

18.1 BACKGROUND

A one-way hash function, $H(M)$, operates on an arbitrary-length pre-image message, M. It returns a fixed-length value, h.

$$h = H(M), \text{ where } h \text{ is of length } m$$

Many functions can take an arbitrary-length input and return an output of fixed length, but one-way hash functions have additional characteristics that make them one-way [1065]:

Given M, it is easy to compute h.

Given h, it is hard to compute M such that $H(M) = h$.

Given M, it is hard to find another message, M', such that $H(M) = H(M')$.

If Mallory could do the hard things, he would undermine the security of every protocol that uses the one-way hash function. The whole point of the one-way hash function is to provide a "fingerprint" of M that is unique. If Alice signed M by using a digital signature algorithm on $H(M)$, and Bob could produce M', another message different from M where $H(M) = H(M')$, then Bob could claim that Alice signed M'.

In some applications, one-wayness is insufficient; we need an additional requirement called **collision-resistance**.

It is hard to find two random messages, M and M', such that $H(M) = H(M')$.

Remember the birthday attack from Section 7.4? It is not based on finding another message M', such that $H(M) = H(M')$, but based on finding two random messages, M and M', such that $H(M) = H(M')$.

The following protocol, first described by Gideon Yuval [1635], shows how—if the previous requirement were not true—Alice could use the birthday attack to swindle Bob.

(1) Alice prepares two versions of a contract: one is favorable to Bob; the other bankrupts him.

(2) Alice makes several subtle changes to each document and calculates the hash value for each. (These changes could be things like: replacing SPACE with SPACE-BACKSPACE-SPACE, putting a space or two before a carriage return, and so on. By either making or not making a single change on each of 32 lines, Alice can easily generate 2^{32} different documents.)

(3) Alice compares the hash values for each change in each of the two documents, looking for a pair that matches. (If the hash function only outputs a 64-bit value, she would usually find a matching pair with 2^{32} versions of each.) She reconstructs the two documents that hash to the same value.

(4) Alice has Bob sign the version of the contract that is favorable to him, using a protocol in which he only signs the hash value.

(5) At some time in the future, Alice substitutes the contract Bob signed with the one that he didn't. Now she can convince an adjudicator that Bob signed the other contract.

This is a big problem. (One moral is to always make a cosmetic change to any document you sign.)

Other similar attacks could be mounted assuming a successful birthday attack. For example, an adversary could send an automated control system (on a satellite, perhaps) random message strings with random signature strings. Eventually, one of those random messages will have a valid signature. The adversary would have no idea what the command would do, but if his only objective was to tamper with the satellite, this would do it.

Length of One-Way Hash Functions

Hash functions of 64 bits are just too small to survive a birthday attack. Most practical one-way hash functions produce 128-bit hashes. This forces anyone attempting the birthday attack to hash 2^{64} random documents to find two that hash to the same value, not enough for lasting security. NIST, in its Secure Hash Standard (SHS), uses a 160-bit hash value. This makes the birthday attack even harder, requiring 2^{80} random hashes.

The following method has been proposed to generate a longer hash value than a given hash function produces.

(1) Generate the hash value of a message, using a one-way hash function listed in this book.

(2) Prepend the hash value to the message.
(3) Generate the hash value of the concatenation of the message and the hash value.
(4) Create a larger hash value consisting of the hash value generated in step (1) concatenated with the hash value generated in step (3).
(5) Repeat steps (1) through (3) as many times as you wish, concatenating as you go.

Although this method has never been proved to be either secure or insecure, various people have some serious reservations about it [1262,859].

Overview of One-Way Hash Functions

It's not easy to design a function that accepts an arbitrary-length input, let alone make it one-way. In the real world, one-way hash functions are built on the idea of a **compression function**. This one-way function outputs a hash value of length n given an input of some larger length m [1069,414]. The inputs to the compression function are a message block and the output of the previous blocks of text (see Figure 18.1). The output is the hash of all blocks up to that point. That is, the hash of block M_i is

$$h_i = f(M_i, h_{i-1})$$

This hash value, along with the next message block, becomes the next input to the compression function. The hash of the entire message is the hash of the last block.

The pre-image should contain some kind of binary representation of the length of the entire message. This technique overcomes a potential security problem resulting from messages with different lengths possibly hashing to the same value [1069,414]. This technique is sometimes called **MD-strengthening** [930].

Various researchers have theorized that if the compression function is secure, then this method of hashing an arbitrary-length pre-image is also secure—but nothing has been proved [1138,1070,414].

A lot has been written on the design of one-way hash functions. For more mathematical information, consult [1028,793,791,1138,1069,414,91,858,1264]. Bart Preneel's thesis [1262] is probably the most comprehensive treatment of one-way hash functions.

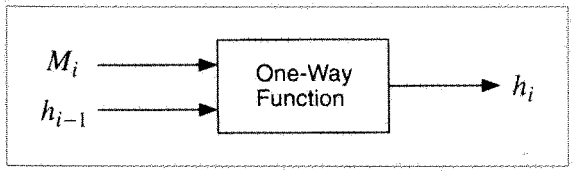


Figure 18.1 One-way function.

18.2 SNEFRU

Snefru is a one-way hash function designed by Ralph Merkle [1070]. (Snefru, like Khufu and Khafre, was an Egyptian pharaoh.) Snefru hashes arbitrary-length messages into either 128-bit or 256-bit values.

First the message is broken into chunks, each 512-m in length. (The variable m is the length of the hash value.) If the output is a 128-bit hash value, then the chunks are each 384 bits long; if the output is a 256-bit hash value, then the chunks are each 256 bits long.

The heart of the algorithm is function H, which hashes a 512-bit value into an m-bit value. The first m bits of H's output are the hash of the block; the rest are discarded. The next block is appended to the hash of the previous block and hashed again. (The initial block is appended to a string of zeros.) After the last block (if the message isn't an even number of blocks long, zeros are used to pad the last block), the first m bits are appended to a binary representation of the length of the message and hashed one final time.

Function H is based on E, which is a reversible block-cipher function that operates on 512-bit blocks. H is the last m bits of the output of E XORed with the first m bits of the input of E.

The security of Snefru resides in function E, which randomizes data in several passes. Each pass is composed of 64 randomizing rounds. In each round a different byte of the data is used as an input to an S-box; the output word of the S-box is XORed with two neighboring words of the message. The S-boxes are constructed in a manner similar to those in Khafre (see Section 13.7). Some rotations are thrown in, too. Originally Snefru was designed with two passes.

Cryptanalysis of Snefru

Using differential cryptanalysis, Biham and Shamir demonstrated the insecurity of two-pass Snefru (128-bit hash value) [172]. Their attack finds pairs of messages that hash to the same value within minutes.

On 128-bit Snefru, their attacks work better than brute force for four passes or less. A birthday attack against Snefru takes 2^{64} operations; differential cryptanalysis can find a pair of messages that hash to the same value in $2^{28.5}$ operations for three-pass Snefru and $2^{44.5}$ operations for four-pass Snefru. Finding a message that hashes to a given value by brute force requires 2^{128} operations; differential cryptanalysis takes 2^{56} operations for three-pass Snefru and 2^{88} operations for four-pass Snefru.

Although Biham and Shamir didn't analyze 256-bit hash values, they extended their analysis to 224-bit hash values. Compared to a birthday attack that requires 2^{112} operations, they can find messages that hash to the same value in $2^{12.5}$ operations for two-pass Snefru, 2^{33} operations for three-pass Snefru, and 2^{81} operations for four-pass Snefru.

Currently, Merkle recommends using Snefru with at least eight passes [1073]. However, with this many passes the algorithm is significantly slower than either MD5 or SHA.

18.3 *N*-HASH

N-Hash is an algorithm invented by researchers at Nippon Telephone and Telegraph, the same people who invented FEAL, in 1990 [1105,1106]. *N*-Hash uses 128-bit message blocks, a complicated randomizing function similar to FEAL's, and produces a 128-bit hash value.

The hash of each 128-bit block is a function of the block and the hash of the previous block.

$$H_0 = I, \text{ where } I \text{ is a random initial value}$$
$$H_i = g(M_i, H_{i-1}) \oplus M_i \oplus H_{i-1}$$

The hash of the entire message is the hash of the last message block. The random initial value, I, can be any value determined by the user (even all zeros).

The function g is a complicated one. Figure 18.2 is an overview of the algorithm. Initially, the 128-bit hash of the previous message block, H_{i-1}, has its 64-bit left half

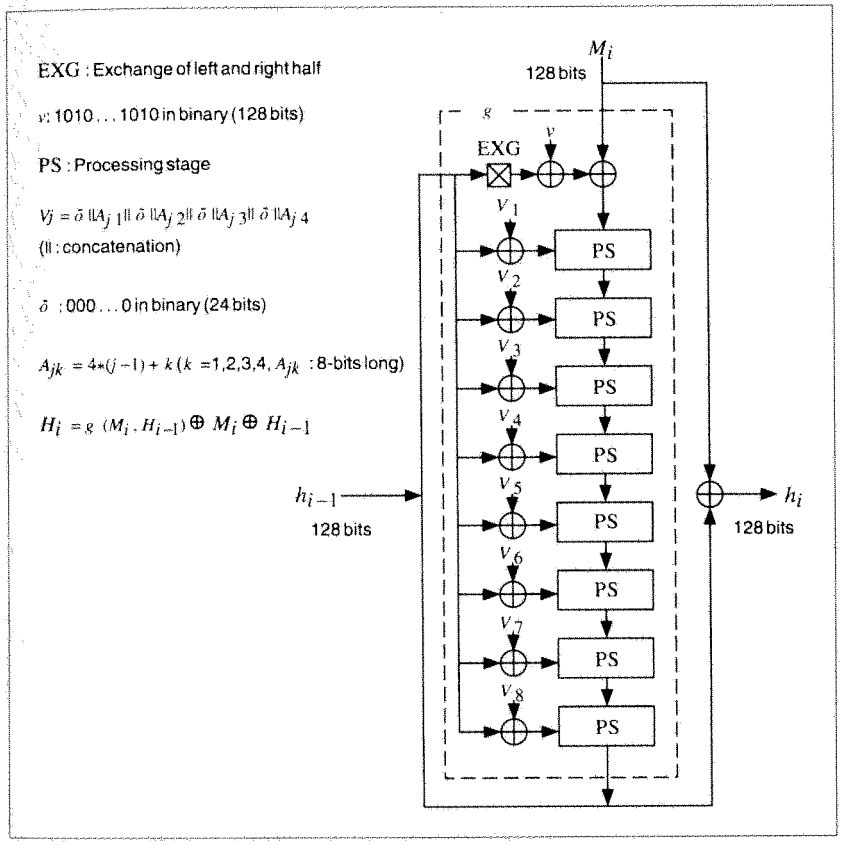


Figure 18.2 Outline of N-*Hash.*

and 64-bit right half swapped; it is then XORed with a repeating one/zero pattern (128 bits worth), and then XORed with the current message block, M_i. This value then cascades into N ($N = 8$ in the figures) processing stages. The other input to the processing stage is the previous hash value XORed with one of eight binary constant values.

One processing stage is given in Figure 18.3. The message block is broken into four 32-bit values. The previous hash value is also broken into four 32-bit values. The function f is given in Figure 18.4. Functions S_0 and S_1 are the same as they were in FEAL.

$$S_0(a,b) = \text{rotate left two bits } ((a + b) \bmod 256)$$
$$S_1(a,b) = \text{rotate left two bits } ((a + b + 1) \bmod 256)$$

The output of one processing stage becomes the input to the next processing stage. After the last processing stage, the output is XORed with the M_i and H_{i-1}, and then the next block is ready to be hashed.

Cryptanalysis of N-Hash

Bert den Boer discovered a way to produce collisions in the round function of *N*-Hash [1262]. Biham and Shamir used differential cryptanalysis to break 6-round

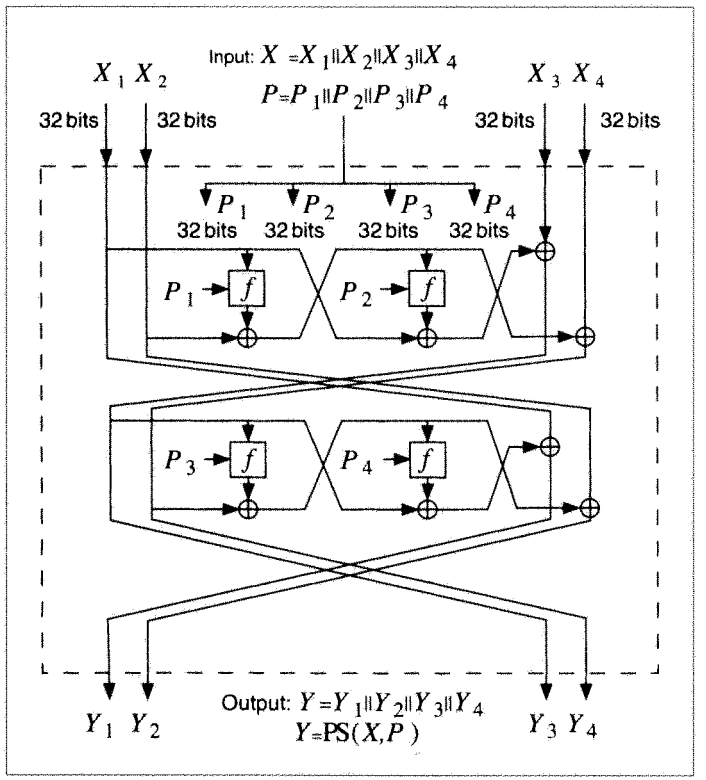


Figure 18.3 One processing stage of N-Hash.

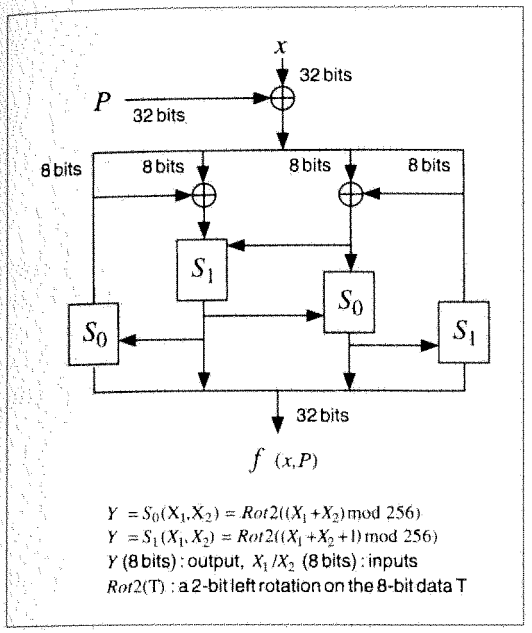


Figure 18.4 Function f.

N-Hash [169,172]. Their particular attack (there certainly could be others) works for any *N* that is divisible by 3, and is more efficient than the birthday attack for any *N* less than 15.

The same attack can find pairs of messages that hash to the same value for 12-round *N*-Hash in 2^{56} operations, compared to 2^{64} operations for a brute-force attack. *N*-hash with 15 rounds is safe from differential cryptanalysis: The attack requires 2^{72} operations.

The algorithm's designers recommend using *N*-Hash with at least 8 rounds [1106]. Given the proven insecurity of *N*-Hash and FEAL (and its speed with 8 rounds), I recommend using another algorithm entirely.

18.4 MD4

MD4 is a one-way hash function designed by Ron Rivest [1318,1319,1321]. MD stands for **Message Digest**; the algorithm produces a 128-bit hash, or message digest, of the input message.

In [1319], Rivest outlined his design goals for the algorithm:

> *Security.* It is computationally infeasible to find two messages that hashed to the same value. No attack is more efficient than brute force.
> *Direct Security.* MD4's security is not based on any assumption, like the difficulty of factoring.

Speed. MD4 is suitable for high-speed software implementations. It is based on a simple set of bit manipulations on 32-bit operands.
Simplicity and Compactness. MD4 is as simple as possible, without large data structures or a complicated program.
Favor Little-Endian Architectures. MD4 is optimized for microprocessor architectures (specifically Intel microprocessors); larger and faster computers make any necessary translations.

After the algorithm was first introduced, Bert den Boer and Antoon Bosselaers successfully cryptanalyzed the last two of the algorithm's three rounds [202]. In an unrelated cryptanalytic result, Ralph Merkle successfully attacked the first two rounds [202]. Eli Biham discussed a differential cryptanalysis attack against the first two rounds of MD4 [159]. Even though these attacks could not be extended to the full algorithm, Rivest strengthened the algorithm. The result is MD5.

18.5 MD5

MD5 is an improved version of MD4 [1386,1322]. Although more complex than MD4, it is similar in design and also produces a 128-bit hash.

Description of MD5

After some initial processing, MD5 processes the input text in 512-bit blocks, divided into 16 32-bit sub-blocks. The output of the algorithm is a set of four 32-bit blocks, which concatenate to form a single 128-bit hash value.

First, the message is padded so that its length is just 64 bits short of being a multiple of 512. This padding is a single 1-bit added to the end of the message, followed by as many zeros as are required. Then, a 64-bit representation of the message's length (before padding bits were added) is appended to the result. These two steps serve to make the message length an exact multiple of 512 bits in length (required for the rest of the algorithm), while ensuring that different messages will not look the same after padding.

Four 32-bit variables are initialized:

$$A = 0x01234567$$
$$B = 0x89abcdef$$
$$C = 0xfedcba98$$
$$D = 0x76543210$$

These are called **chaining variables**.

Now, the main loop of the algorithm begins. This loop continues for as many 512-bit blocks as are in the message.

The four variables are copied into different variables: a gets A, b gets B, c gets C, and d gets D.

The main loop has four rounds (MD4 had only three rounds), all very similar. Each round uses a different operation 16 times. Each operation performs a nonlinear func-

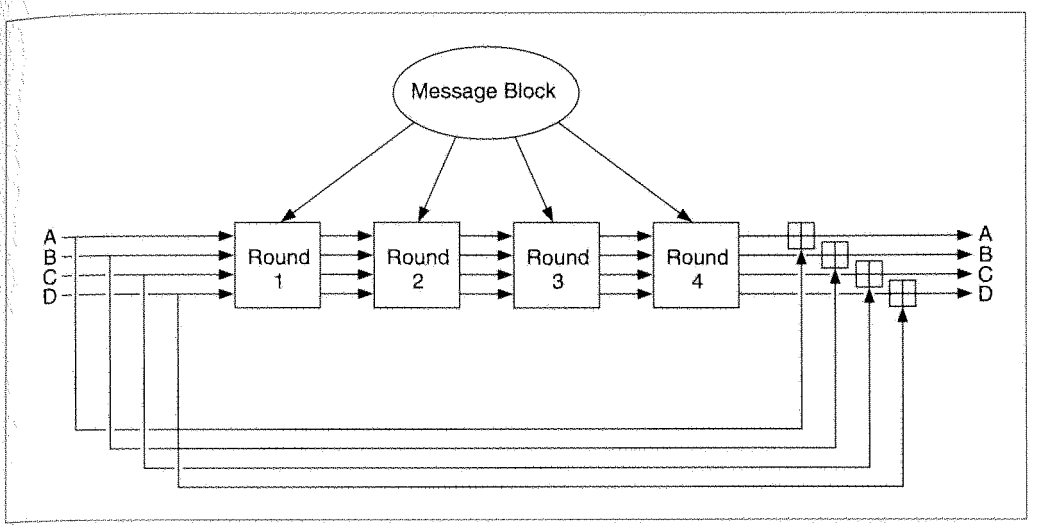


Figure 18.5 MD5 main loop.

tion on three of a, b, c, and d. Then it adds that result to the fourth variable, a sub-block of the text and a constant. Then it rotates that result to the right a variable number of bits and adds the result to one of a, b, c, or d. Finally the result replaces one of a, b, c, or d. See Figures 18.5 and 18.6.

There are four nonlinear functions, one used in each operation (a different one for each round).

$$F(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee (\neg Z))$$

($\oplus$ is XOR, $\wedge$ is AND, $\vee$ is OR, and $\neg$ is NOT.)

These functions are designed so that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of the result will also be independent and unbiased. The function F is the bit-wise conditional: If X then Y else Z. The function H is the bit-wise parity operator.

If M_j represents the jth sub-block of the message (from 0 to 15), and $<<<s$ represents a left circular shift of s bits, the four operations are:

$$FF(a,b,c,d,M_j,s,t_i) \text{ denotes } a = b + ((a + F(b,c,d) + M_j + t_i) <<< s)$$
$$GG(a,b,c,d,M_j,s,t_i) \text{ denotes } a = b + ((a + G(b,c,d) + M_j + t_i) <<< s)$$
$$HH(a,b,c,d,M_j,s,t_i) \text{ denotes } a = b + ((a + H(b,c,d) + M_j + t_i) <<< s)$$
$$II(a,b,c,d,M_j,s,t_i) \text{ denotes } a = b + ((a + I(b,c,d) + M_j + t_i) <<< s)$$

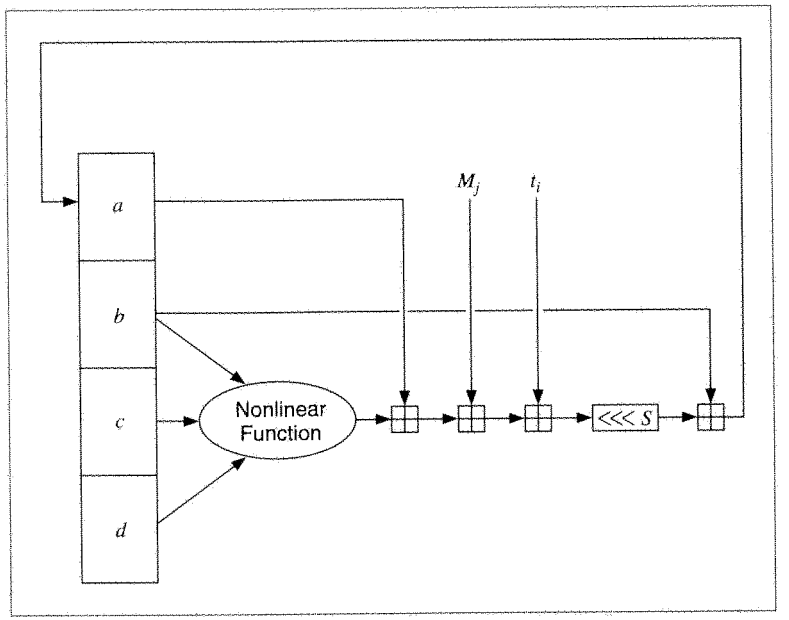


Figure 18.6 One MD5 operation.

The four rounds (64 steps) look like:

Round 1:

FF (*a*, *b*, *c*, *d*, M_0, 7, 0xd76aa478)
FF (*d*, *a*, *b*, *c*, M_1, 12, 0xe8c7b756)
FF (*c*, *d*, *a*, *b*, M_2, 17, 0x242070db)
FF (*b*, *c*, *d*, *a*, M_3, 22, 0xc1bdceee)
FF (*a*, *b*, *c*, *d*, M_4, 7, 0xf57c0faf)
FF (*d*, *a*, *b*, *c*, M_5, 12, 0x4787c62a)
FF (*c*, *d*, *a*, *b*, M_6, 17, 0xa8304613)
FF (*b*, *c*, *d*, *a*, M_7, 22, 0xfd469501)
FF (*a*, *b*, *c*, *d*, M_8, 7, 0x698098d8)
FF (*d*, *a*, *b*, *c*, M_9, 12, 0x8b44f7af)
FF (*c*, *d*, *a*, *b*, M_{10}, 17, 0xffff5bb1)
FF (*b*, *c*, *d*, *a*, M_{11}, 22, 0x895cd7be)
FF (*a*, *b*, *c*, *d*, M_{12}, 7, 0x6b901122)
FF (*d*, *a*, *b*, *c*, M_{13}, 12, 0xfd987193)
FF (*c*, *d*, *a*, *b*, M_{14}, 17, 0xa679438e)
FF (*b*, *c*, *d*, *a*, M_{15}, 22, 0x49b40821)

Round 2:

GG (*a*, *b*, *c*, *d*, M_1, 5, 0xf61e2562)
GG (*d*, *a*, *b*, *c*, M_6, 9, 0xc040b340)
GG (*c*, *d*, *a*, *b*, M_{11}, 14, 0x265e5a51)
GG (*b*, *c*, *d*, *a*, M_0, 20, 0xe9b6c7aa)
GG (*a*, *b*, *c*, *d*, M_5, 5, 0xd62f105d)
GG (*d*, *a*, *b*, *c*, M_{10}, 9, 0x02441453)
GG (*c*, *d*, *a*, *b*, M_{15}, 14, 0xd8a1e681)
GG (*b*, *c*, *d*, *a*, M_4, 20, 0xe7d3fbc8)
GG (*a*, *b*, *c*, *d*, M_9, 5, 0x21e1cde6)
GG (*d*, *a*, *b*, *c*, M_{14}, 9, 0xc33707d6)
GG (*c*, *d*, *a*, *b*, M_3, 14, 0xf4d50d87)
GG (*b*, *c*, *d*, *a*, M_8, 20, 0x455a14ed)
GG (*a*, *b*, *c*, *d*, M_{13}, 5, 0xa9e3e905)
GG (*d*, *a*, *b*, *c*, M_2, 9, 0xfcefa3f8)
GG (*c*, *d*, *a*, *b*, M_7, 14, 0x676f02d9)
GG (*b*, *c*, *d*, *a*, M_{12}, 20, 0x8d2a4c8a)

Round 3:

HH (*a*, *b*, *c*, *d*, M_5, 4, 0xfffa3942)
HH (*d*, *a*, *b*, *c*, M_8, 11, 0x8771f681)
HH (*c*, *d*, *a*, *b*, M_{11}, 16, 0x6d9d6122)
HH (*b*, *c*, *d*, *a*, M_{14}, 23, 0xfde5380c)
HH (*a*, *b*, *c*, *d*, M_1, 4, 0xa4beea44)
HH (*d*, *a*, *b*, *c*, M_4, 11, 0x4bdecfa9)
HH (*c*, *d*, *a*, *b*, M_7, 16, 0xf6bb4b60)
HH (*b*, *c*, *d*, *a*, M_{10}, 23, 0xbebfbc70)
HH (*a*, *b*, *c*, *d*, M_{13}, 4, 0x289b7ec6)
HH (*d*, *a*, *b*, *c*, M_0, 11, 0xeaa127fa)
HH (*c*, *d*, *a*, *b*, M_3, 16, 0xd4ef3085)
HH (*b*, *c*, *d*, *a*, M_6, 23, 0x04881d05)
HH (*a*, *b*, *c*, *d*, M_9, 4, 0xd9d4d039)
HH (*d*, *a*, *b*, *c*, M_{12}, 11, 0xe6db99e5)
HH (*c*, *d*, *a*, *b*, M_{15}, 16, 0x1fa27cf8)
HH (*b*, *c*, *d*, *a*, M_2, 23, 0xc4ac5665)

Round 4:
II (*a*, *b*, *c*, *d*, M_0, 6, 0xf4292244)
II (*d*, *a*, *b*, *c*, M_7, 10, 0x432aff97)
II (*c*, *d*, *a*, *b*, M_{14}, 15, 0xab9423a7)
II (*b*, *c*, *d*, *a*, M_5, 21, 0xfc93a039)
II (*a*, *b*, *c*, *d*, M_{12}, 6, 0x655b59c3)
II (*d*, *a*, *b*, *c*, M_3, 10, 0x8f0ccc92)
II (*c*, *d*, *a*, *b*, M_{10}, 15, 0xffeff47d)
II (*b*, *c*, *d*, *a*, M_1, 21, 0x85845dd1)
II (*a*, *b*, *c*, *d*, M_8, 6, 0x6fa87e4f)
II (*d*, *a*, *b*, *c*, M_{15}, 10, 0xfe2ce6e0)
II (*c*, *d*, *a*, *b*, M_6, 15, 0xa3014314)
II (*b*, *c*, *d*, *a*, M_{13}, 21, 0x4e0811a1)
II (*a*, *b*, *c*, *d*, M_4, 6, 0xf7537e82)
II (*d*, *a*, *b*, *c*, M_{11}, 10, 0xbd3af235)
II (*c*, *d*, *a*, *b*, M_2, 15, 0x2ad7d2bb)
II (*b*, *c*, *d*, *a*, M_9, 21, 0xeb86d391)

Those constants, t_i, were chosen as follows:

In step *i*, t_i is the integer part of 2^{32}*abs(sin(*i*)), where *i* is in radians.

After all of this, *a*, *b*, *c*, and *d* are added to *A*, *B*, *C*, *D*, respectively, and the algorithm continues with the next block of data. The final output is the concatenation of *A*, *B*, *C*, and *D*.

Security of MD5

Ron Rivest outlined the improvements of MD5 over MD4 [1322]:

1. A fourth round has been added.
2. Each step now has a unique additive constant.
3. The function G in round 2 was changed from $((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$ to $((X \wedge Z) \vee (Y \neg (Z)))$ to make *G* less symmetric.
4. Each step now adds in the result of the previous step. This promotes a faster avalanche effect.
5. The order in which message sub-blocks are accessed in rounds 2 and 3 is changed, to make these patterns less alike.
6. The left circular shift amounts in each round have been approximately optimized, to yield a faster avalanche effect. The four shifts used in each round are different from the ones used in other rounds.

Tom Berson attempted to use differential cryptanalysis against a single round of MD5 [144], but his attack is ineffective against all four rounds. A more successful attack by den Boer and Bosselaers produces collisions using the compression function in MD5 [203,1331,1336]. This does not lend itself to attacks against MD5 in practical applications, and it does not affect the use of MD5 in Luby-Rackoff-like encryption algorithms (see Section 14.11). It does mean that one of the basic design principles of MD5—to design a collision-resistant compression function—has been violated. Although it is true that "there seems to be a weakness in the compression function, but it has no practical impact on the security of the hash function" [1336], I am wary of using MD5.

18.6 MD2

MD2 is another 128-bit one-way hash function designed by Ron Rivest [801,1335]. It, along with MD5, is used in the PEM protocols (see Section 24.10). The security of MD2 is dependent on a random permutation of bytes. This permutation is fixed, and depends on the digits of π. $S_0, S_1, S_2, \ldots, S_{255}$ is the permutation. To hash a message M:

(1) Pad the message with i bytes of value i so that the resulting message is a multiple of 16 bytes long.

(2) Append a 16-byte checksum to the message.

(3) Initialize a 48-byte block: $X_0, X_1, X_2, \ldots, X_{47}$. Set the first 16 bytes of X to be 0, the second 16 bytes of X to be the first 16 bytes of the message, and the third 16 bytes of X to be the XOR of the first 16 bytes of X and the second 16 bytes of X.

(4) This is the compression function:

$t = 0$
For $j = 0$ to 17
For $k = 0$ to 47
$t = X_k$ XOR S_t
$X_k = t$
$t = (t + j) \bmod 256$

(5) Set the second 16 bytes of X to be the second 16 bytes of the message, and the third 16 bytes of X to be the XOR of the first 16 bytes of X and the second 16 bytes of X. Do step (4). Repeat steps (5) and (4) with every 16 bytes of the message, in turn.

(6) The output is the first 16 bytes of X.

Although no weaknesses in MD2 have been found (see [1262]), it is slower than most other suggested hash functions.

18.7 SECURE HASH ALGORITHM (SHA)

NIST, along with the NSA, designed the Secure Hash Algorithm (SHA) for use with the Digital Signature Standard (see Section 20.2) [1154]. (The standard is the Secure Hash Standard (SHS); SHA is the algorithm used in the standard.)
According to the *Federal Register* [539]:

> A Federal Information Processing Standard (FIPS) for Secure Hash Standard (SHS) is being proposed. This proposed standard specified a Secure Hash Algorithm (SHA) for use with the proposed Digital Signature Standard . . .. Additionally, for applications not requiring a digital signature, the SHA is to be used whenever a secure hash algorithm is required for Federal applications.

And

> This Standard specifies a Secure Hash Algorithm (SHA), which is necessary to ensure the security of the Digital Signature Algorithm (DSA). When a message of any length $< 2^{64}$ bits is input, the SHA produces a 160-bit output called a message digest. The message digest is then input to the DSA, which computes the signature for the message. Signing the message digest rather than the message often improves the efficiency of the process, because the message digest is usually much smaller than the message. The same message digest should be obtained by the verifier of the signature when the received version of the message is used as input to SHA. The SHA is called secure because it is designed to be computationally infeasible to recover a message corresponding to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with a very high probability, result in a different message digest, and the signature will fail to verify. The SHA is based on principles similar to those used by Professor Ronald L. Rivest of MIT when designing the MD4 message digest algorithm [1319], and is closely modelled after that algorithm.

SHA produces a 160-bit hash, longer than MD5.

Description of SHA

First, the message is padded to make it a multiple of 512 bits long. Padding is exactly the same as in MD5: First append a one, then as many zeros as necessary to make it 64 bits short of a multiple of 512, and finally a 64-bit representation of the length of the message before padding.

Five 32-bit variables (MD5 has four variables, but this algorithm needs to produce a 160-bit hash) are initialized as follows:

A = 0x67452301
B = 0xefcdab89
C = 0x98badcfe
D = 0x10325476
E = 0xc3d2e1f0

The main loop of the algorithm then begins. It processes the message 512 bits at a time and continues for as many 512-bit blocks as are in the message.

First the five variables are copied into different variables: a gets A, b gets B, c gets C, d gets D, and e gets E.

The main loop has four rounds of 20 operations each (MD5 has four rounds of 16 operations each). Each operation performs a nonlinear function on three of a, b, c, d, and e, and then does shifting and adding similar to MD5.

SHA's set of nonlinear functions is:

$$f_t(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z), \text{ for } t = 0 \text{ to } 19.$$
$$f_t(X,Y,Z) = X \oplus Y \oplus Z, \text{ for } t = 20 \text{ to } 39.$$
$$f_t(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), \text{ for } t = 40 \text{ to } 59.$$
$$f_t(X,Y,Z) = X \oplus Y \oplus Z, \text{ for } t = 60 \text{ to } 79.$$

Four constants are used in the algorithm:

$$K_t = \text{0x5a827999}, \text{ for } t = 0 \text{ to } 19.$$
$$K_t = \text{0x6ed9eba1}, \text{ for } t = 20 \text{ to } 39.$$
$$K_t = \text{0x8f1bbcdc}, \text{ for } t = 40 \text{ to } 59.$$
$$K_t = \text{0xca62c1d6}, \text{ for } t = 60 \text{ to } 79.$$

(If you wonder where those numbers came from: $\text{0x5a827999} = 2^{1/2}/4$, $\text{0x6ed9eba1} = 3^{1/2}/4$, $\text{0x8f1bbcdc} = 5^{1/2}/4$, and $\text{0xca62c1d6} = 10^{1/2}/4$.)

The message block is transformed from 16 32-bit words (M_0 to M_{15}) to 80 32-bit words (W_0 to W_{79}) using the following algorithm:

$$W_t = M_t, \text{ for } t = 0 \text{ to } 15$$
$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1, \text{ for } t = 16 \text{ to } 79.$$

(As an interesting aside, the original SHA specification did not have the left circular shift. The change "corrects a technical flaw that made the standard less secure than had been thought" [543]. The NSA has refused to elaborate on the exact nature of the flaw.)

If t is the operation number (from 1 to 80), W_t represents the tth sub-block of the expanded message, and $\lll s$ represents a left circular shift of s bits, then the main loop looks like:

FOR $t = 0$ to 79
  $TEMP = (a \lll 5) + f_t(b,c,d) + e + W_t + K_t$
  $e = d$
  $d = c$
  $c = b \lll 30$
  $b = a$
  $a = TEMP$

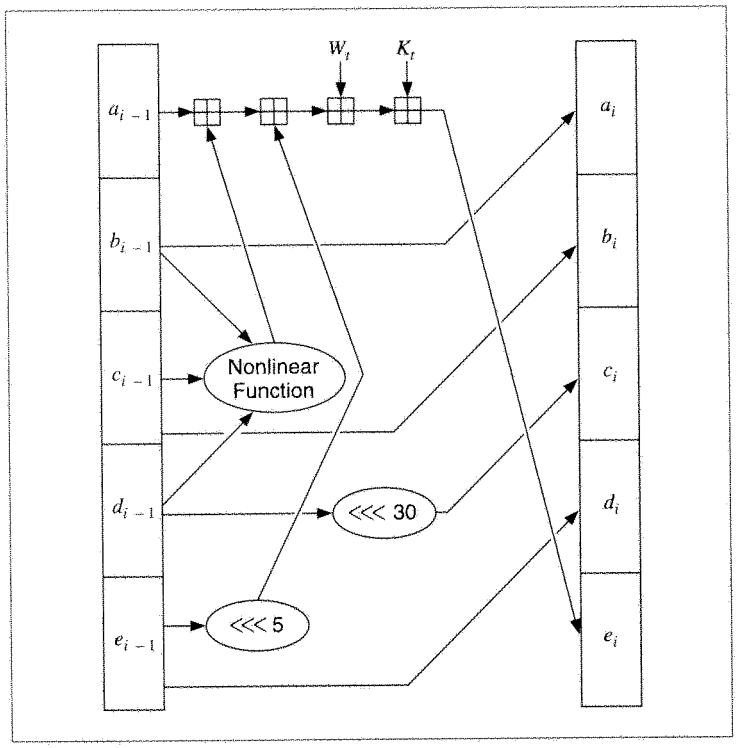


Figure 18.7 One SHA operation.

Figure 18.7 shows one operation. Shifting the variables accomplishes the same thing as MD5 does by using different variables in different locations.

After all of this, *a*, *b*, *c*, *d*, and *e* are added to *A*, *B*, *C*, *D*, and *E* respectively, and the algorithm continues with the next block of data. The final output is the concatenation of *A*, *B*, *C*, *D*, and *E*.

Security of SHA

SHA is very similar to MD4, but has a 160-bit hash value. The main changes are the addition of an expand transformation and the addition of the previous step's output into the next step for a faster avalanche effect. Ron Rivest made public the design decisions behind MD5, but SHA's designers did not. Here are Rivest's MD5 improvements to MD4 and how they compare with SHA's:

1. "A fourth round has been added." SHA does this, too. However, in SHA the fourth round uses the same f function as the second round.
2. "Each step now has a unique additive constant." SHA keeps the MD4 scheme where it reuses the constants for each group of 20 rounds.
3. "The function G in round 2 was changed from $((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$ to $((X \wedge Z) \vee (Y \wedge \neg (Z)))$ to make G less symmetric." SHA uses the MD4 version: $((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$.

4. "Each step now adds in the result of the previous step. This promotes a faster avalanche effect." This change has been made in SHA as well. The difference in SHA is that a fifth variable is added, and not *b*, *c*, or *d*, which is already used in f_t. This subtle change makes the den Boer-Bosselaers attack against MD5 impossible against SHA.
5. "The order in which message sub-blocks are accessed in rounds 2 and 3 is changed, to make these patterns less alike." SHA is completely different, since it uses a cyclic error-correcting code.
6. "The left circular shift amounts in each round have been approximately optimized, to yield a faster avalanche effect. The four shifts used in each round are different from the ones used in other rounds." SHA uses a constant shift amount in each round. This shift amount is relatively prime to the word size, as in MD4.

This leads to the following comparison: SHA is MD4 with the addition of an expand transformation, an extra round, and better avalanche effect; MD5 is MD4 with improved bit hashing, an extra round, and better avalanche effect.

There are no known cryptographic attacks against SHA. Because it produces a 160-bit hash, it is more resistant to brute-force attacks (including birthday attacks) than 128-bit hash functions covered in this chapter.

18.8 RIPE-MD

RIPE-MD was developed for the European Community's RIPE project [1305] (see Section 25.7). The algorithm is a variation of MD4, designed to resist known cryptanalytic attacks, and produce a 128-bit hash value. The rotations and the order of the message words are modified. Additionally, two instances of the algorithm, differing only in the constants, run in parallel. After each block, the output of both instances are added to the chaining variables. This seems to make the algorithm highly resistant to cryptanalysis.

18.9 HAVAL

HAVAL is a variable-length one-way hash function [1646]. It is a modification of MD5. HAVAL processes messages in blocks of 1024 bits, twice those of MD5. It has eight 32-bit chaining variables, twice those of MD5. It has a variable number of rounds, from three to five (each of which has 16 steps), and it can produce a hash length of 128, 160, 192, 224, or 256 bits.

HAVAL replaces MD5's simple nonlinear functions with highly nonlinear 7-variable functions, each of which satisfies the strict avalanche criterion. Each round uses a single function, but in every step a different permutation is applied to the inputs. It has a new message order and every step (except those in the first round) uses a different additive constant. The algorithm also has two rotations.

The core of the algorithm is

$$TEMP = (f(j,A,B,C,D,E,F,G) <<< 7) + (H <<< 11) + M[i][r(j)] + K(j)$$
$$H = G;\ G = F;\ F = E;\ E = D;\ D = C;\ C = B;\ B = A;\ A = TEMP$$

The variable number of rounds and variable-length output mean there are 15 versions of this algorithm. Den Boer's and Bosselaers's attack against MD5 [203] does not apply to HAVAL because of the rotation of H.

18.10 OTHER ONE-WAY HASH FUNCTIONS

MD3 is yet another hash function designed by Ron Rivest. It had several flaws and never really made it out of the laboratory, although a description was recently published in [1335].

A group of researchers at the University of Waterloo have proposed a one-way hash function based on iterated exponentiation in $GF(2^{593})$ [22]. In this scheme, a message is divided into 593-bit blocks; beginning with the first block, the blocks are successively exponentiated. Each exponent is the result of the computation with the previous block; the first exponent is given by an IV.

Ivan Damgård designed a one-way hash function based on the knapsack problem (see Section 19.2) [414]; it can be broken in about 2^{32} operations [290,1232,787].

Steve Wolfram's cellular automata [1608] have been proposed as a basis for one-way hash functions. An early implementation [414] is insecure [1052,404]. Another one-way hash function, Cellhash [384,404], and an improved version, Subhash [384,402,405], are based on cellular automata; both are designed for hardware. Boognish mixes the design principles of Cellhash with those of MD4 [402,407]. StepRightUp can be implemented as a hash function as well [402].

Claus Schnorr proposed a one-way hash function based on the discrete Fourier transform, called FFT-Hash, in the summer of 1991 [1399]; it was broken a few months later by two independent groups [403,84]. Schnorr proposed a revised version, called FFT-Hash II (the previous version was renamed FFT-Hash I) [1400], which was broken a few weeks later [1567]. Schnorr has proposed further modifications [1402,1403] but, as it stands, the algorithm is much slower than the others in this chapter. Another hash function, called SL_2 [1526], is insecure [315].

Additional theoretical work on constructing one-way hash functions from one-way functions and one-way permutations can be found in [412,1138,1342].

18.11 ONE-WAY HASH FUNCTIONS USING SYMMETRIC BLOCK ALGORITHMS

It is possible to use a symmetric block cipher algorithm as a one-way hash function. The idea is that if the block algorithm is secure, then the one-way hash function will also be secure.

The most obvious method is to encrypt the message with the algorithm in CBC or CFB mode, a fixed key, and IV; the last ciphertext block is the hash value. These methods are described in various standards using DES: both modes in [1143], CBC in [1145], CBC in [55,56,54]. This just isn't good enough for one-way hash functions, although it will work for a MAC (see Section 18.14) [29].

A cleverer approach uses the message block as the key, the previous hash value as the input, and the current hash value as the output.

The actual hash functions proposed are even more complex. The block size is usually the key length, and the size of the hash value is the block size. Since most block algorithms are 64 bits, several schemes are designed around a hash that is twice the block size.

Assuming the hash function is correct, the security of the scheme is based on the security of the underlying block function. There are exceptions, though. Differential cryptanalysis is easier against block functions in hash functions than against block functions used for encryption: The key is known, so several tricks can be applied; only one right pair is needed for success; and you can generate as much chosen plaintext as you want. Some work on these lines is [1263,858,1313].

What follows is a summary of the various hash functions that have appeared in the literature [925,1465,1262]. Statements about attacks against these schemes assume that the underlying block cipher is secure; that is, the best attack against them is brute force.

One useful measure for hash functions based on block ciphers is the **hash rate**, or the number of n-bit messages blocks, where n is the block size of the algorithm, processed per encryption. The higher the hash rate, the faster the algorithm. (This measure was given the opposite definition in [1262], but the definition given here is more intuitive and is more widely used. This can be confusing.)

Schemes Where the Hash Length Equals the Block Size

The general scheme is as follows (see Figure 18.8):

$$H_0 = I_H, \text{ where } I_H \text{ is a random initial value}$$
$$H_i = E_A(B) \oplus C$$

where A, B, and C can be either M_i, H_{i-1}, $(M_i \oplus H_{i-1})$, or a constant (assumed to be 0). H_0 is some random initial value: I_H. The message is divided up into block-size chunks, M_i, and processed individually. And there is some kind of MD-strengthening, perhaps the same padding procedure used in MD5 and SHA.

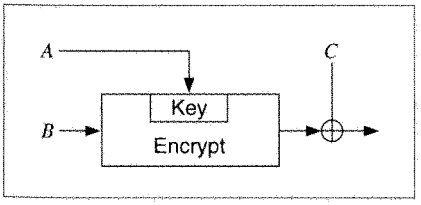


Figure 18.8 General hash function where the hash length equals the block size.

Table 18.1
Secure Hash Functions Where the Block Length Equals the Hash Size

$H_i = E_{H_{i-1}}(M_i) \oplus M_i$
$H_i = E_{H_{i-1}}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
$H_i = E_{H_{i-1}}(M_i) \oplus H_{i-1} \oplus M_i$
$H_i = E_{H_{i-1}}(M_i \oplus H_{i-1}) \oplus M_i$
$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1}$
$H_i = E_{M_i}(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$
$H_i = E_{M_i}(H_{i-1}) \oplus M_i \oplus H_{i-1}$
$H_i = E_{M_i}(M_i \oplus H_{i-1}) \oplus H_{i-1}$
$H_i = E_{M_i \oplus H_{i-1}}(M_i) \oplus M_i$
$H_i = E_{M_i \oplus H_{i-1}}(H_{i-1}) \oplus H_{i-1}$
$H_i = E_{M_i \oplus H_{i-1}}(M_i) \oplus H_{i-1}$
$H_i = E_{M_i \oplus H_{i-1}}(H_{i-1}) \oplus M_i$

The three different variables can take on one of four possible values, so there are 64 total schemes of this type. Bart Preneel studied them all [1262].

Fifteen are trivially weak because the result does not depend on one of the inputs. Thirty-seven are insecure for more subtle reasons. Table 18.1 lists the 12 secure schemes remaining: The first 4 are secure against all attacks (see Figure 18.9) and the last 8 are secure against all but a fixed-point attack, which is not really worth worrying about.

The first scheme was described in [1028]. The third scheme was described in [1555,1105,1106] and was proposed as an ISO standard [766]. The fifth scheme was proposed by Carl Meyer, but is commonly called Davies-Meyer in the literature [1606,1607,434,1028]. The tenth scheme was proposed as a hash-function mode for LOKI [273].

The first, second, third, fourth, ninth, and eleventh schemes have a hash rate of 1; the key length equals the block length. The others have a rate of k/n, where k is the key length. This means that if the key length is shorter than the block length, then the message block can only be the length of the key. It is not recommended that the message block be longer than the key length, even if the encryption algorithm's key length is longer than the block length.

If the block algorithm has a DES-like complementation property and DES-like weak keys, there is an additional attack that is possible against all 12 schemes. The attack isn't very dangerous and not really worth worrying about. However, you can solve it by fixing bits 2 and 3 of the key to "01" or "10" [1081,1107]. Of course, this reduces the length of k from 56 bits to 54 bits (in DES, for example) and decreases the hash rate.

The following schemes, proposed in the literature, have been shown to be insecure.

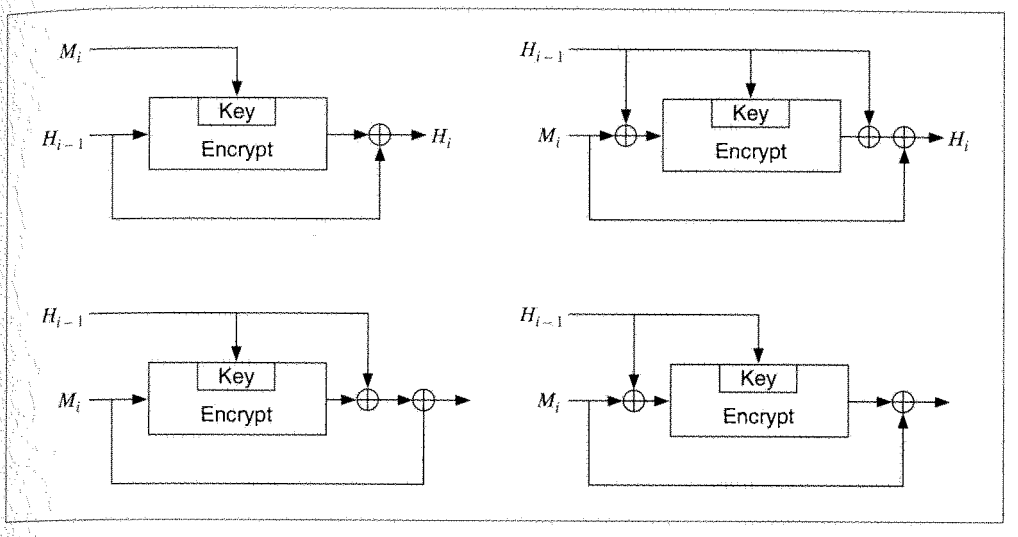


Figure 18.9 The four secure hash functions where the block length equals the hash size.

This scheme [1282] was broken in [369]:

$$H_i = E_{M_i}(H_{i-1})$$

Davies and Price proposed a variant which cycles the entire message through the algorithm twice [432,433]. Coppersmith's attack works on this variant with not much larger computational requirements [369].

Another scheme [432,458] was shown insecure in [1606]:

$$H_i = E_{M_i \oplus H_i - 1}(H_{i-1})$$

This scheme was shown insecure in [1028] (c is a constant):

$$H_i = E_c(M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$$

Modified Davies-Meyer

Lai and Massey modified the Davies-Meyer technique to work with the IDEA cipher [930,925]. IDEA has a 64-bit block size and 128-bit key size. Their scheme is

$$H_0 = I_H, \text{ where } I_H \text{ is a random initial value}$$
$$H_i = E_{H_{i-1}, M_i}(H_{i-1})$$

This function hashes the message in blocks of 64 bits and produces a 64-bit hash value (See Figure 18.10).

No known attack on this scheme is easier than brute force.

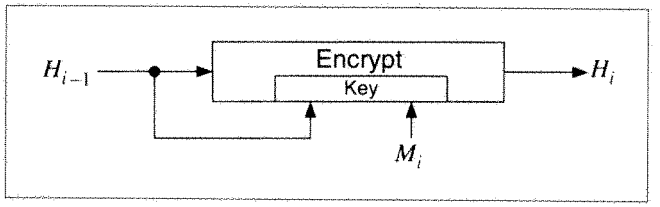


Figure 18.10 Modified Davies-Meyer.

Preneel-Bosselaers-Govaerts-Vandewalle

This hash function, first proposed in [1266], produces a hash value twice the block length of the encryption algorithm: A 64-bit algorithm produces a 128-bit hash.

With a 64-bit block algorithm, the scheme produces two 64-bit hash values, G_i and H_i, which are concatenated to produce the 128-bit hash. With most block algorithms, the block size is 64 bits. Two adjacent message blocks, L_i and R_i, each the size of the block length, are hashed together.

$$G_0 = I_G, \text{ where } I_G \text{ is a random initial value}$$
$$H_0 = I_H, \text{ where } I_H \text{ is another random initial value}$$
$$G_i = E_{L_i \oplus H_{i-1}}(R_i \oplus G_{i-1}) \oplus R_i \oplus G_{i-1} \oplus H_{i-1}$$
$$H_i = E_{L_i \oplus R_i}(H_{i-1} \oplus G_{i-1}) \oplus L_i \oplus G_{i-1} \oplus H_{i-1}$$

Lai demonstrates attacks against this scheme that, in some instances, make the birthday attack trivially solvable [925,926]. Preneel [1262] and Coppersmith [372] also have successful attacks against this scheme. Do not use it.

Quisquater-Girault

This scheme, first proposed in [1279], generates a hash that is twice the block length and has a hash rate of 1. It has two hash values, G_i and H_i, and two blocks, L_i and R_i, are hashed together.

$$G_0 = I_G, \text{ where } I_G \text{ is a random initial value}$$
$$H_0 = I_H, \text{ where } I_H \text{ is another random initial value}$$
$$W_i = E_{L_i}(G_{i-1} \oplus R_i) \oplus R_i \oplus H_{i-1}$$
$$G_i = E_{R_i}(W_i \oplus L_i) \oplus G_{i-1} \oplus H_{i-1} \oplus L_i$$
$$H_i = W_i \oplus G_{i-1}$$

This scheme appeared in a 1989 draft ISO standard [764], but was dropped in a later version [765]. Security problems with this scheme were identified in [1107,925, 1262,372]. (Actually, the version in the proceedings was strengthened after the version presented at the conference was attacked.) In some instances the birthday attack is solvable with a complexity of 2^{39}, not 2^{64}, through brute force. Do not use this scheme.

LOKI Double-Block

This algorithm is a modification of Quisquater-Girault, specifically designed to work with LOKI [273]. All parameters are as in Quisquater-Girault.

$G_0 = I_G$, where I_G is a random initial value

$H_0 = I_H$, where I_H is another random initial value

$W_i = E_{L_i \oplus G_{i-1}}(G_{i-1} \oplus R_i) \oplus R_i \oplus H_{i-1}$

$G_i = E_{R_i \oplus H_{i-1}}(W_i \oplus L_i) \oplus G_{i-1} \oplus H_{i-1} \oplus L_i$

$H_i = W_i \oplus G_{i-1}$

Again, in some instances the birthday attack is trivially solvable [925,926,1262, 372,736]. Do not use this scheme.

Parallel Davies-Meyer

This is yet another attempt at an algorithm with a hash rate of 1 that produces a hash twice the block length [736].

$G_0 = I_G$, where I_G is a random initial value

$H_0 = I_H$, where I_H is another random initial value

$G_i = E_{L_i \oplus R_i}(G_{i-1} \oplus L_i) \oplus L_i \oplus H_{i-1}$

$H_i = E_{L_i}(H_{i-1} \oplus R_i) \oplus R_i \oplus H_{i-1}$

Unfortunately, this scheme isn't secure either [928,861]. As it turns out, a double-length hash function with a hash rate of 1 cannot be more secure than Davies-Meyer [861].

Tandem and Abreast Davies-Meyer

Another way around the inherent limitations of a block cipher with a 64-bit key uses an algorithm, like IDEA (see Section 13.9), with a 64-bit block and a 128-bit key. These two schemes produce a 128-bit hash value and have a hash rate of ½ [930,925].

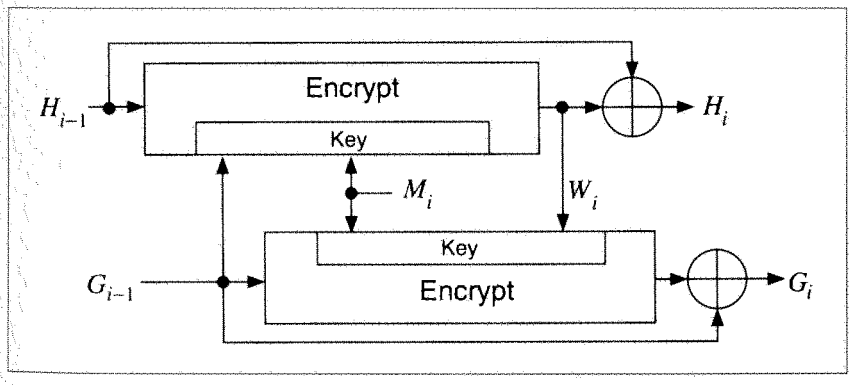


Figure 18.11 Tandem Davies-Meyer.

In this first scheme, two modified Davies-Meyer functions work in tandem (see Figure 18.11).

$$G_0 = I_G, \text{ where } I_G \text{ is some random initial value}$$
$$H_0 = I_H, \text{ where } I_H \text{ is some other random initial value}$$
$$W_i = E_{G_{i-1}, M_i}(H_{i-1})$$
$$G_i = G_{i-1} \oplus E_{M_i, W_i}(G_{i-1})$$
$$H_i = W_i \oplus H_{i-1}$$

The following scheme uses two modified Davies-Meyer functions side-by-side (see Figure 18.12).

$$G_0 = I_G, \text{ where } I_G \text{ is some random initial value}$$
$$H_0 = I_H, \text{ where } I_H \text{ is some other random initial value}$$
$$G_i = G_{i-1} \oplus E_{M_i, H_{i-1}}(\neg G_{i-1})$$
$$H_i = H_{i-1} \oplus E_{G_{i-1}, M_i}(H_{i-1})$$

In both schemes, the two 64-bit hash values G_i and H_i are concatenated to produce a single 128-bit hash.

As far as anyone knows, these algorithms have ideal security for a 128-bit hash function: Finding a message that hashes to a given hash value requires 2^{128} attempts, and finding two random messages that hash to the same value requires 2^{64} attempts—assuming that there is no better way to attack the block algorithm than by using brute force.

MDC-2 and MDC-4

MDC-2 and MDC-4 were first developed at IBM [1081,1079]. MDC-2, sometimes called Meyer-Schilling, is under consideration as an ANSI and ISO standard [61,765]; a variant was proposed in [762]. MDC-4 is specified for the RIPE project [1305] (see Section 25.7). The specifications use DES as the block function, although in theory any encryption algorithm could be used.

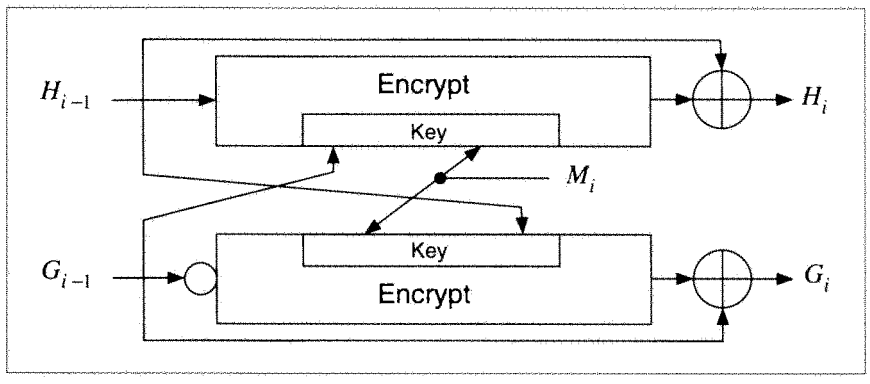


Figure 18.12 Abreast Davies-Meyer.

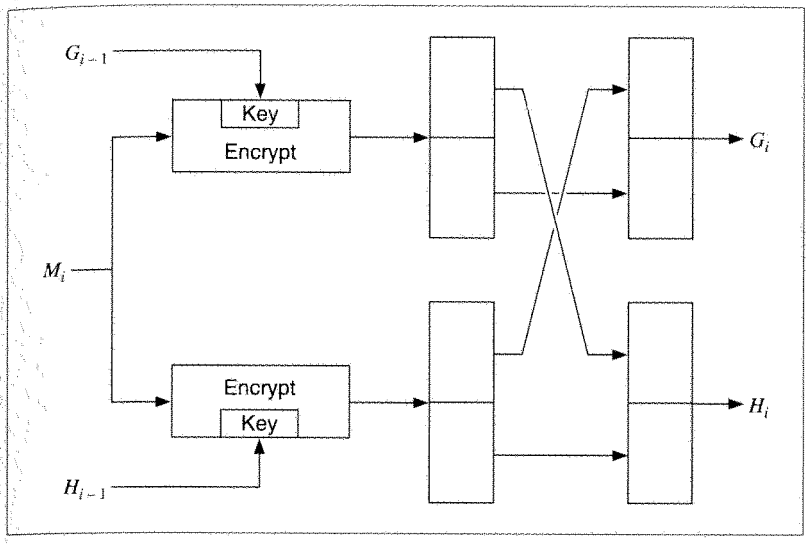


Figure 18.13 MDC-2.

MDC-2 has a hash rate of ½, and produces a hash value twice the length of the block size. It is shown in Figure 18.13. MDC-4 also produces a hash value twice the length of the block size, and has a hash rate of ¼ (see Figure 18.14).

These schemes have been analyzed in [925,1262]. They are secure against current computing power, but they are not nearly as secure as the designers have estimated. If the block algorithm is DES, they have been looked at with respect to differential cryptanalysis [1262].

Both MDC-2 and MDC-4 are patented [223].

AR Hash Function

The AR hash function was developed by Algorithmic Research, Ltd. and has been distributed by the ISO for information purposes only [767]. Its basic structure is a variant of the underlying block cipher (DES in the reference) in CBC mode. The last two ciphertext blocks and a constant are XORed to the current message block and encrypted by the algorithm. The hash is the last two ciphertext blocks computed. The message is processed twice, with two different keys, so the hash function has a hash rate of ½. The first key is 0x0000000000000000, the second key is 0x2a4152 2f4446502a, and c is 0x0123456789abcdef. The result is compressed to a single 128-bit hash value. See [750] for the details.

$$H_i = E_K(M_i \oplus H_{i-1} \oplus H_{i-2} \oplus c) \oplus M_i$$

This sounds interesting, but it is insecure. After considerable preprocessing, it is possible to find collisions for this hash function easily [416].

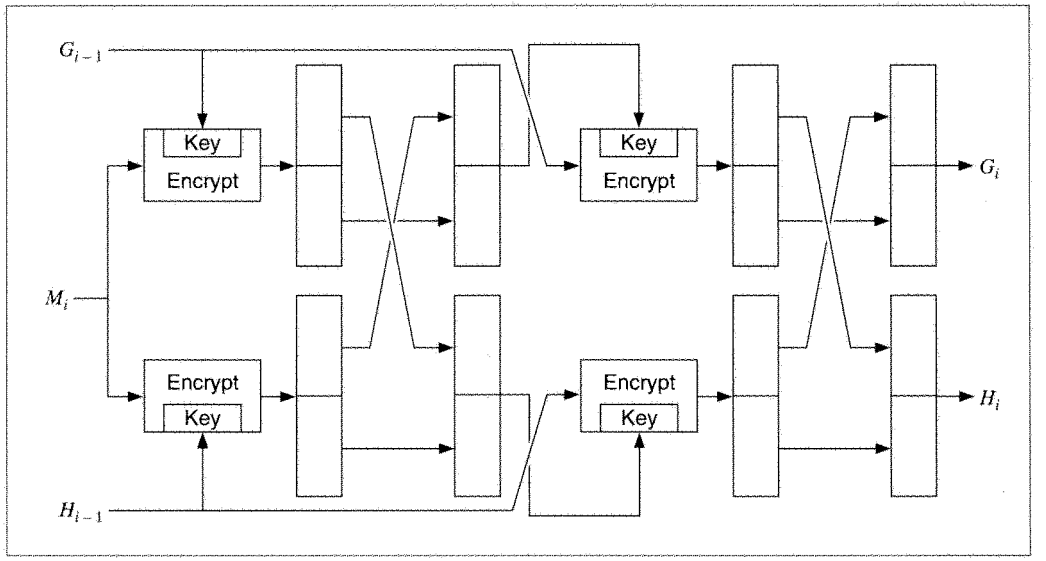


Figure 18.14 MDC-4.

GOST Hash Function

This hash function comes from Russia, and is specified in the standard GOST R 34.11-94 [657]. It uses the GOST block algorithm (see Section 14.1), although in theory it could use any block algorithm with a 64-bit block size and a 256-bit key. The function produces a 256-bit hash value.

The compression function, $H_i = f(M_i, H_{i-1})$ (both operands are 256-bit quantities) is defined as follows:

(1) Generate four GOST encryption keys by some linear mixing of M_i, $H_i - 1$, and some constants.

(2) Use each key to encrypt a different 64 bits of H_{i-1} in ECB mode. Store the resulting 256 bits into a temporary variable, S.

(3) H_i is a complex, although linear, function of S, M_i, and H_{i-1}.

The final hash of M is not the hash of the last block. There are actually three chaining variables: H_n is the hash of the last message block, Z is the XOR of all the message blocks, and L is the length of the message. Given those variables and the padded last block, M', the final hash value is:

$$H = f(Z \oplus M', f(L, f(M', H_n)))$$

The documentation is a bit confusing (and in Russian), but I think all that is correct. In any case, this hash function is specified for use with the Russian Digital Signature Standard (see Section 20.3).

Other Schemes

Ralph Merkle proposed a scheme using DES, but it's slow; it only processes seven message bits per iteration and each iteration involves two DES encryptions [1065, 1069]. Another scheme [1642,1645] is insecure [1267]; it was once proposed as an ISO standard.

18.12 Using Public-Key Algorithms

It is possible to use a public-key encryption algorithm in a block chaining mode as a one-way hash function. If you then throw away the private key, breaking the hash would be as difficult as reading the message without the private key.

Here's an example using RSA. If M is the message to be hashed, n is the product of two primes p and q, and e is another large number relatively prime to $(p-1)(q-1)$, then the hash function, $H(M)$, would be

$$H(M) = M^e \bmod n$$

An even easier solution would be to use a single strong prime as the modulus p. Then:

$$H(M) = M^e \bmod p$$

Breaking this problem is probably as difficult as finding the discrete logarithm of e. The problem with this algorithm is that it's far slower than any others discussed here. I don't recommend it for that reason.

18.13 Choosing a One-Way Hash Function

The contenders seem to be SHA, MD5, and constructions based on block ciphers; the others really haven't been studied enough to be in the running. I vote for SHA. It has a longer hash value than MD5, is faster than the various block-cipher constructions, and was developed by the NSA. I trust the NSA's abilities at cryptanalysis, even if they don't make their results public.

Table 18.2 gives timing measurements for some hash functions. They are meant for comparison purposes only.

18.14 Message Authentication Codes

A message authentication code, or MAC, is a key-dependent one-way hash function. MACs have the same properties as the one-way hash functions discussed previously, but they also include a key. (This does not imply that you can publish the key to a MAC and use it as a one-way hash function.) Only someone with the identical key can verify the hash. They are very useful to provide authenticity without secrecy.

MACs can be used to authenticate files between users. They can also be used by a single user to determine if his files have been altered, perhaps by a virus. A user

Table 18.2
Encryption Speeds of Some Hash Functions on a 33 MHz 486SX

Algorithm	Hash Length	Encryption Speed (kilobytes/second)
Abreast Davies-Meyer (with IDEA)	128	22
Davies-Meyer (with DES)	64	9
GOST Hash	256	11
HAVAL (3 passes)	variable	168
HAVAL (4 passes)	variable	118
HAVAL (5 passes)	variable	95
MD2	128	23
MD4	128	236
MD5	128	174
N-HASH (12 rounds)	128	29
N-HASH (15 rounds)	128	24
RIPE-MD	128	182
SHA	160	75
Snerfu (4 passes)	128	48
Snerfu (8 passes)	128	23

could compute the MAC of his files and store that value in a table. If the user used instead a one-way hash function, then the virus could compute the new hash value after infection and replace the table entry. A virus could not do that with a MAC, because the virus does not know the key.

An easy way to turn a one-way hash function into a MAC is to encrypt the hash value with a symmetric algorithm. Any MAC can be turned into a one-way hash function by making the key public.

CBC-MAC

The simplest way to make a key-dependent one-way hash function is to encrypt a message with a block algorithm in CBC or CFB modes. The hash is the last encrypted block, encrypted once more in CBC or CFB modes. The CBC method is specified in ANSI X9.9 [54], ANSI X9.19 [56], ISO 8731-1 [759], ISO 9797 [763], and an Australian standard [1496]. Differential cryptanalysis can break this scheme with reduced-round DES or FEAL as the underlying block algorithms [1197].

The potential security problem with this method is that the receiver must have the key, and that key allows him to generate messages with the same hash value as a given message by decrypting in the reverse direction.

Message Authenticator Algorithm (MAA)

This algorithm is an ISO standard [760]. It produces a 32-bit hash, and was designed for mainframe computers with a fast multiply instruction [428].

$$v = v \lll 1$$
$$e = v \oplus w$$
$$x = (((e + y) \bmod 2^{32}) \vee A \wedge C) \star (x \oplus M_i)) \bmod 2^{32} - 1$$
$$y = (((e + x) \bmod 2^{32}) \vee B \wedge D) \star (y \oplus M_i)) \bmod 2^{32} - 2$$

Iterate these for each message block, M_i, and the resultant hash is the XOR of x and y. The variables v and e are determined from the key. A, B, C, and D are constants.

This algorithm is probably in wide use, but I can't believe it is all that secure. It was designed a long time ago, and isn't very complicated.

Bidirectional MAC

This MAC produces a hash value twice the length of the block algorithm [978]. First, compute the CBC-MAC of the message. Then, compute the CBC-MAC of the message with the blocks in reverse order. The bidirectional MAC value is simply the concatenation of the two. Unfortunately, this construction is insecure [1097].

Jueneman's Methods

This MAC is also called a quadratic congruential manipulation detection code (QCMDC) [792,789]. First, divide the message into m-bit blocks. Then:

$H_0 = I_H$, where I_H is the secret key
$H_i = (H_{i-1} + M_i)^2 \bmod p$, where p is a prime less than $2^m - 1$
and + denotes integer addition

Jueneman suggests $n = 16$ and $p = 2^{31} - 1$. In [792] he also suggests that an additional key be used as H_1, with the actual message starting at H_2.

Because of a variety of birthday-type attacks discovered in conjunction with Don Coppersmith, Jueneman suggested computing the QCMDC four times, using the result of one iteration as the IV for the next iteration, and then concatenating the results to obtain a 128-bit hash value [793]. This was further strengthened by doing the four iterations in parallel and cross-linking them [790,791]. This scheme was broken by Coppersmith [376].

Another variant [432,434] replaced the addition operation with an XOR and used message blocks significantly smaller than p. H_0 was also set, making it a keyless one-way hash function. After this scheme was attacked [612], it was strengthened as part of the European Open Shop Information-TeleTrust project [1221], quoted in CCITT X.509 [304], and adopted in ISO 10118 [764,765]. Unfortunately, Coppersmith has broken this scheme as well [376]. There has been some research using exponents other than 2 [603], but none of it has been promising.

RIPE-MAC

RIPE-MAC was invented by Bart Preneel [1262] and adopted by the RIPE project [1305] (see Section 18.8). It is based on ISO 9797 [763], and uses DES as a block encryption function. RIPE-MAC has two flavors: one using normal DES, called

RIPE-MAC1, and another using triple-DES for even greater security, called RIPE-MAC3. RIPE-MAC1 uses one DES encryption per 64-bit message block; RIPE-MAC3 uses three.

The algorithm consists of three parts. First, the message is expanded to a length that is a multiple of 64 bits. Next, the expanded message is divided up into 64-bit blocks. A keyed compression function is used to hash these blocks, under the control of a secret key, into a single block of 64 bits. This is the step that uses either DES or triple-DES. Finally, the output of this compression is subjected to another DES-based encryption with a different key, derived from the key used in the compression. See [1305] for details.

IBC-Hash

IBC-Hash is another MAC adopted by the RIPE project [1305] (see Section 18.8). It is interesting because it is provably secure; the chance of successful attack can be quantified. Unfortunately, every message must be hashed with a different key. The chosen level of security puts constraints on the maximum message size that can be hashed—something no other function in this chapter does. Given these considerations, the RIPE report recommends that IBC-Hash be used only for long, infrequently sent messages.

The heart of the function is

$$h_i = (M_i \bmod p) + v) \bmod 2^n$$

The secret key is the pair p and v, where p is an n-bit prime and v is a random number less than 2^n. The M_i values are derived by a carefully specified padding procedure. The probabilities of breaking both the one-wayness and the collision-resistance can be quantified, and users can choose their security level by changing the parameters.

One-Way Hash Function MAC

A one-way hash function can also be used as a MAC [1537]. Assume Alice and Bob share a key K, and Alice wants to send Bob a MAC for message M. Alice concatenates K and M, and computes the one-way hash of the concatenation: $H(K,M)$. This hash is the MAC. Since Bob knows K, he can reproduce Alice's result. Mallory, who does not know K, can't.

This method works with MD-strengthening techniques, but has serious problems. Mallory can always add new blocks to the end of the message and compute a valid MAC. This attack can be thwarted if you put the message length at the beginning, but Preneel is suspicious of this scheme [1265]. It is better to put the key at the end of the message, $H(M,K)$, but this has some problems as well [1265]. If H is one-way but not collision-free, Mallory can forge messages. Still better is $H(K,M,K)$, or $H(K_1,M,K_2)$, where K_1 and K_2 are different [1537]. Preneel is still suspicious [1265].

The following constructions seem secure:

$H(K_1, H(K_2,M))$

$H(K, H(K,M))$

$H(K,p,M,K)$, where p pads K to a full message block.

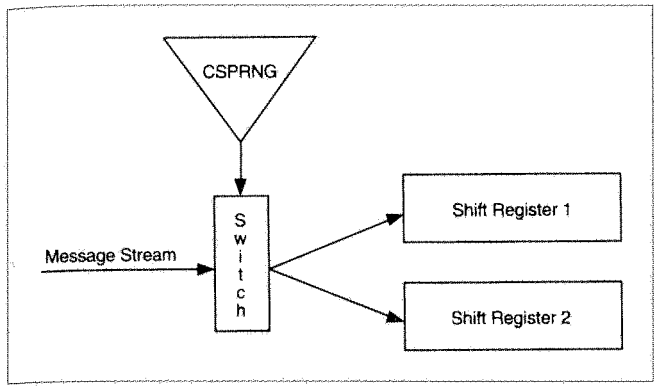


Figure 18.15 Stream cipher MAC.

The best approach is to concatenate at least 64 bits of the key with each message block. This makes the one-way hash function less efficient, because the message blocks are smaller, but it is much more secure [1265].

Alternatively, use a one-way hash function and a symmetric algorithm. Hash the file, then encrypt the hash. This is more secure than first encrypting the file and then hashing the encrypted file, but it is vulnerable to the same attack as the $H(M,K)$ approach [1265].

Stream Cipher MAC

This MAC scheme uses stream ciphers (see Figure 18.15) [932]. A cryptographically secure pseudo-random-bit generator demultiplexes the message stream into two substreams. If the output bit of the bit generator k_i, is 1, then the current message bit m_i, is routed to the first substream; if the k_i is 0, the m_i is routed to the second substream. The substreams are each fed into a different LFSR (see Section 16.2). The output of the MAC is simply the final states of the shift registers.

Unfortunately, this method is not secure against small changes in the message [1523]. For example, if you alter the last bit of the message, then only 2 bits in the corresponding MAC value need to be altered to create a fake MAC; this can be done with reasonable probability. The author presents a more secure, and more complicated, alternative.

CHAPTER 19

Public-Key Algorithms

19.1 BACKGROUND

The concept of public-key cryptography was invented by Whitfield Diffie and Martin Hellman, and independently by Ralph Merkle. Their contribution to cryptography was the notion that keys could come in pairs—an encryption key and a decryption key—and that it could be infeasible to generate one key from the other (see Section 2.5). Diffie and Hellman first presented this concept at the 1976 National Computer Conference [495]; a few months later, their seminal paper "New Directions in Cryptography" was published [496]. (Due to a glacial publishing process, Merkle's first contribution to the field didn't appear until 1978 [1064].)

Since 1976, numerous public-key cryptography algorithms have been proposed. Many of these are insecure. Of those still considered secure, many are impractical. Either they have too large a key or the ciphertext is much larger than the plaintext.

Only a few algorithms are both secure and practical. These algorithms are generally based on one of the hard problems discussed in Section 11.2. Of these secure and practical public-key algorithms, some are only suitable for key distribution. Others are suitable for encryption (and by extension for key distribution). Still others are only useful for digital signatures. Only three algorithms work well for both encryption and digital signatures: RSA, ElGamal, and Rabin. All of these algorithms are slow. They encrypt and decrypt data much more slowly than symmetric algorithms; usually that's too slow to support bulk data encryption.

Hybrid cryptosystems (see Section 2.5) speed things up: A symmetric algorithm with a random session key is used to encrypt the message, and a public-key algorithm is used to encrypt the random session key.

Security of Public-Key Algorithms

Since a cryptanalyst has access to the public key, he can always choose any message to encrypt. This means that a cryptanalyst, given $C = E_K[P]$, can guess the value

of P and easily check his guess. This is a serious problem if the number of possible plaintext messages is small enough to allow exhaustive search, but can be solved by padding messages with a string of random bits. This makes identical plaintext messages encrypt to different ciphertext messages. [For more about this concept, see Section 23.15.]

This is especially important if a public-key algorithm is used to encrypt a session key. Eve can generate a database of all possible session keys encrypted with Bob's public key. Sure, this requires a large amount of time and memory, but for a 40-bit exportable key or a 56-bit DES key, it's a whole lot less time and memory than breaking Bob's public key. Once Eve has generated the database, she will have his key and can read his mail at will.

Public-key algorithms are designed to resist chosen-plaintext attacks; their security is based both on the difficulty of deducing the secret key from the public key and the difficulty of deducing the plaintext from the ciphertext. However, most public-key algorithms are particularly susceptible to a chosen-ciphertext attack [see Section 1.1].

In systems where the digital signature operation is the inverse of the encryption operation, this attack is impossible to prevent unless different keys are used for encryption and signatures.

Consequently, it is important to look at the whole system and not just at the individual parts. Good public-key protocols are designed so that the various parties can't decrypt arbitrary messages generated by other parties—the proof-of-identity protocols are a good example [see Section 5.2].

19.2 KNAPSACK ALGORITHMS

The first algorithm for generalized public-key encryption was the knapsack algorithm developed by Ralph Merkle and Martin Hellman [713,1074]. It could only be used for encryption, although Adi Shamir later adapted the system for digital signatures [1413]. Knapsack algorithms get their security from the knapsack problem, an **NP-complete** problem. Although this algorithm was later found to be insecure, it is worth examining because it demonstrates how an **NP-complete** problem can be used for public-key cryptography.

The knapsack problem is a simple one. Given a pile of items, each with different weights, is it possible to put some of those items into a knapsack so that the knapsack weighs a given amount? More formally: Given a set of values $M_1, M_2, \ldots, M_n$, and a sum S, compute the values of b_i such that

$$S = b_1M_1 + b_2M_2 + \ldots + b_nM_n$$

The values of b_i can be either zero or one. A one indicates that the item is in the knapsack; a zero indicates that it isn't.

For example, the items might have weights of 1, 5, 6, 11, 14, and 20. You could pack a knapsack that weighs 22; use weights 5, 6, and 11. You could not pack a knapsack that weighs 24. In general, the time required to solve this problem seems to grow exponentially with the number of items in the pile.

The idea behind the Merkle-Hellman knapsack algorithm is to encode a message as a solution to a series of knapsack problems. A block of plaintext equal in length to the number of items in the pile would select the items in the knapsack (plaintext bits corresponding to the *b* values), and the ciphertext would be the resulting sum. Figure 19.1 shows a plaintext encrypted with a sample knapsack problem.

The trick is that there are actually two different knapsack problems, one solvable in linear time and the other believed not to be. The easy knapsack can be modified to create the hard knapsack. The public key is the hard knapsack, which can easily be used to encrypt but cannot be used to decrypt messages. The private key is the easy knapsack, which gives an easy way to decrypt messages. People who don't know the private key are forced to try to solve the hard knapsack problem.

Superincreasing Knapsacks

What is the easy knapsack problem? If the list of weights is a **superincreasing sequence**, then the resulting knapsack problem is easy to solve. A superincreasing sequence is a sequence in which every term is greater than the sum of all the previous terms. For example, {1,3,6,13,27,52} is a superincreasing sequence, but {1,3,4,9, 15,25} is not.

The solution to a **superincreasing knapsack** is easy to find. Take the total weight and compare it with the largest number in the sequence. If the total weight is less than the number, then it is not in the knapsack. If the total weight is greater than or equal to the number, then it is in the knapsack. Reduce the weight of the knapsack by the value and move to the next largest number in the sequence. Repeat until finished. If the total weight has been brought to zero, then there is a solution. If the total weight has not, there isn't.

For example, consider a total knapsack weight of 70 and a sequence of weights of {2,3,6,13,27,52}. The largest weight, 52, is less than 70, so 52 is in the knapsack. Subtracting 52 from 70 leaves 18. The next weight, 27, is greater than 18, so 27 is not in the knapsack. The next weight, 13, is less than 18, so 13 is in the knapsack. Subtracting 13 from 18 leaves 5. The next weight, 6, is greater than 5, so 6 is not in the knapsack. Continuing this process will show that both 2 and 3 are in the knapsack and the total weight is brought to 0, which indicates that a solution has been found. Were this a Merkle-Hellman knapsack encryption block, the plaintext that resulted from a ciphertext value of 70 would be 110101.

Non-superincreasing, or normal, knapsacks are hard problems; they have no known quick algorithm. The only known way to determine which items are in the

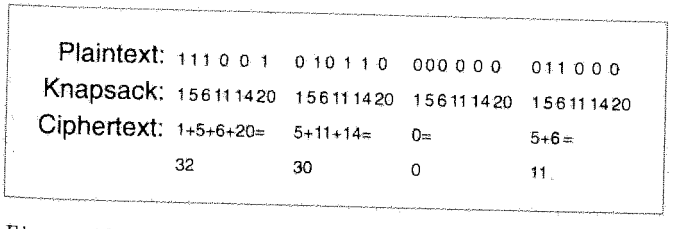


Figure 19.1 Encryption with knapsacks.

knapsack is to methodically test possible solutions until you stumble on the correct one. The fastest algorithms, taking into account the various heuristics, grow exponentially with the number of possible weights in the knapsack. Add one item to the sequence of weights, and it takes twice as long to find the solution. This is much more difficult than a superincreasing knapsack where, if you add one more weight to the sequence, it simply takes another operation to find the solution.

The Merkle-Hellman algorithm is based on this property. The private key is a sequence of weights for a superincreasing knapsack problem. The public key is a sequence of weights for a normal knapsack problem with the same solution. Merkle and Hellman developed a technique for converting a superincreasing knapsack problem into a normal knapsack problem. They did this using modular arithmetic.

Creating the Public Key from the Private Key

Without going into the number theory, this is how the algorithm works: To get a normal knapsack sequence, take a superincreasing knapsack sequence, for example {2,3,6,13,27,52}, and multiply all of the values by a number n, mod m. The modulus should be a number greater than the sum of all the numbers in the sequence: for example, 105. The multiplier should have no factors in common with the modulus: for example, 31. The normal knapsack sequence would then be

$$2 * 31 \bmod 105 = 62$$
$$3 * 31 \bmod 105 = 93$$
$$6 * 31 \bmod 105 = 81$$
$$13 * 31 \bmod 105 = 88$$
$$27 * 31 \bmod 105 = 102$$
$$52 * 31 \bmod 105 = 37$$

The knapsack would then be {62,93,81,88,102,37}.

The superincreasing knapsack sequence is the private key. The normal knapsack sequence is the public key.

Encryption

To encrypt a binary message, first break it up into blocks equal to the number of items in the knapsack sequence. Then, allowing a one to indicate the item is present and a zero to indicate that the item is absent, compute the total weights of the knapsacks—one for every message block.

For example, if the message were 011000110101101110 in binary, encryption using the previous knapsack would proceed like this:

message = 011000 110101 101110

011000 corresponds to 93 + 81 = 174

110101 corresponds to 62 + 93 + 88 + 37 = 280

101110 corresponds to 62 + 81 + 88 + 102 = 333

The ciphertext would be

174,280,333

Decryption

A legitimate recipient of this message knows the private key: the original superincreasing knapsack, as well as the values of n and m used to transform it into a normal knapsack. To decrypt the message, the recipient must first determine n^{-1} such that $n(n^{-1}) \equiv 1 \pmod{m}$. Multiply each of the ciphertext values by $n^{-1} \bmod m$, and then partition with the private knapsack to get the plaintext values.

In our example, the superincreasing knapsack is {2,3,6,13,27,52}, m is equal to 105, and n is equal to 31. The ciphertext message is 174,280,333. In this case n^{-1} is equal to 61, so the ciphertext values must be multiplied by 61 mod 105.

$174 * 61 \bmod 105 = 9 = 3 + 6$, which corresponds to 011000

$280 * 61 \bmod 105 = 70 = 2 + 3 + 13 + 52$, which corresponds to 110101

$333 * 61 \bmod 105 = 48 = 2 + 6 + 13 + 27$, which corresponds to 101110

The recovered plaintext is 011000 110101 101110.

Practical Implementations

With a knapsack sequence of only six items, it's not hard to solve the problem even if it isn't superincreasing. Real knapsacks should contain at least 250 items. The value for each term in the superincreasing knapsack should be somewhere between 200 and 400 bits long, and the modulus should be somewhere between 100 to 200 bits long. Real implementations of the algorithm use random-sequence generators to produce these values.

With knapsacks like that, it's futile to try to solve them by brute force. If a computer could try a million possibilities per second, trying all possible knapsack values would take over 10^{46} years. Even a million machines working in parallel wouldn't solve this problem before the sun went nova.

Security of Knapsacks

It wasn't a million machines that broke the knapsack cryptosystem, but a pair of cryptographers. First a single bit of plaintext was recovered [725]. Then, Shamir showed that knapsacks can be broken in certain circumstances [1415,1416]. There were other results—[1428,38,754,516,488]—but no one could break the general Merkle-Hellman system. Finally, Shamir and Zippel [1418,1419,1421] found flaws in the transformation that allowed them to reconstruct the superincreasing knapsack from the normal knapsack. The exact arguments are beyond the scope of this book, but a nice summary of them can be found in [1233,1244]. At the conference where the results were presented, the attack was demonstrated on stage using an Apple II computer [492,494].

Knapsack Variants

Since the original Merkle-Hellman scheme was broken, many other knapsack systems have been proposed: multiple iterated knapsacks, Graham-Shamir knapsacks, and others. These have all been analyzed and broken, generally using the same cryptographic techniques, and litter the cryptographic highway [260,253,269,921,15,919, 920,922,366,254,263,255]. Good overviews of these systems and their cryptanalyses can be found in [267,479,257,268].

Other algorithms have been proposed that use ideas similar to those used in knapsack cryptosystems, but these too have been broken. The Lu-Lee cryptosystem [990,13] was broken in [20,614,873]; a modification [507] is also insecure [1620]. Attacks on the Goodman-McAuley cryptosystem are in [646,647,267,268]. The Pieprzyk cryptosystem [1246] can be broken by similar attacks. The Niemi cryptosystem [1169], based on modular knapsacks, was broken in [345,788]. A newer multistage knapsack [747] has not yet been broken, but I am not optimistic. Another variant is [294].

While a variation of the knapsack algorithm is currently secure—the Chor-Rivest knapsack [356], despite a "specialized attack" [743]—the amount of computation required makes it far less useful than the other algorithms discussed here. A variant, called the Powerline System, is not secure [958]. Most important, considering the ease with which all the other variations fell, it doesn't seem prudent to trust them.

Patents

The original Merkle-Hellman algorithm is patented in the United States [720] and worldwide (see Table 19.1). Public Key Partners (PKP) licenses the patent, along with other public-key cryptography patents (see Section 25.5). The U.S. patent will expire on August 19, 1997.

19.3 RSA

Soon after Merkle's knapsack algorithm came the first full-fledged public-key algorithm, one that works for encryption and digital signatures: RSA [1328,1329]. Of all the public-key algorithms proposed over the years, RSA is by far the easiest to understand and implement. (Martin Gardner published an early description of the algorithm in his "Mathematical Games" column in *Scientific American* [599].) It is

Table 19.1
Foreign Merkle-Hellman Knapsack Patents

Country	Number	Date of Issue
Belgium	871039	5 Apr 1979
Netherlands	7810063	10 Apr 1979
Great Britain	2006580	2 May 1979
Germany	2843583	10 May 1979
Sweden	7810478	14 May 1979
France	2405532	8 Jun 1979
Germany	2843583	3 Jun 1982
Germany	2857905	15 Jul 1982
Canada	1128159	20 Jul 1982
Great Britain	2006580	18 Aug 1982
Switzerland	63416114	14 Jan 1983
Italy	1099780	28 Sep 1985

also the most popular. Named after the three inventors—Ron Rivest, Adi Shamir, and Leonard Adleman—it has since withstood years of extensive cryptanalysis. Although the cryptanalysis neither proved nor disproved RSA's security, it does suggest a confidence level in the algorithm.

RSA gets its security from the difficulty of factoring large numbers. The public and private keys are functions of a pair of large (100 to 200 digits or even larger) prime numbers. Recovering the plaintext from the public key and the ciphertext is conjectured to be equivalent to factoring the product of the two primes.

To generate the two keys, choose two random large prime numbers, p and q. For maximum security, choose p and q of equal length. Compute the product:

$$n = pq$$

Then randomly choose the encryption key, e, such that e and $(p-1)(q-1)$ are relatively prime. Finally, use the extended Euclidean algorithm to compute the decryption key, d, such that

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

In other words,

$$d = e^{-1} \bmod ((p-1)(q-1))$$

Note that d and n are also relatively prime. The numbers e and n are the public key; the number d is the private key. The two primes, p and q, are no longer needed. They should be discarded, but never revealed.

To encrypt a message m, first divide it into numerical blocks smaller than n (with binary data, choose the largest power of 2 less than n). That is, if both p and q are 100-digit primes, then n will have just under 200 digits and each message block, m_i, should be just under 200 digits long. (If you need to encrypt a fixed number of blocks, you can pad them with a few zeros on the left to ensure that they will always be less than n. The encrypted message, c, will be made up of similarly sized message blocks, c_i, of about the same length. The encryption formula is simply

$$c_i = m_i^e \bmod n$$

To decrypt a message, take each encrypted block c_i and compute

$$m_i = c_i^d \bmod n$$

Since

$$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i m_i^{k(p-1)(q-1)} = m_i * 1 = m_i; \text{ all } (\bmod n)$$

the formula recovers the message. This is summarized in Table 19.2.

The message could just as easily have been encrypted with d and decrypted with e; the choice is arbitrary. I will spare you the number theory that proves why this works; most current texts on cryptography cover it in detail.

A short example will probably go a long way to making this clearer. If $p = 47$ and $q = 71$, then

Table 19.2
RSA Encryption

Public Key:
n product of two primes, p and q (p and q must remain secret)
e relatively prime to $(p-1)(q-1)$

Private Key:
d $e^{-1} \bmod ((p-1)(q-1))$

Encrypting:
$c = m^e \bmod n$

Decrypting:
$m = c^d \bmod n$

$$n = pq = 3337$$

The encryption key, e, must have no factors in common with

$$(p-1)(q-1) = 46 * 70 = 3220$$

Choose e (at random) to be 79. In that case

$$d = 79^{-1} \bmod 3220 = 1019$$

This number was calculated using the extended Euclidean algorithm (see Section 11.3). Publish e and n, and keep d secret. Discard p and q.

To encrypt the message

$$m = 6882326879666683$$

first break it into small blocks. Three-digit blocks work nicely in this case. The message is split into six blocks, m_i, in which

$$m_1 = 688$$
$$m_2 = 232$$
$$m_3 = 687$$
$$m_4 = 966$$
$$m_5 = 668$$
$$m_6 = 003$$

The first block is encrypted as

$$688^{79} \bmod 3337 = 1570 = c_1$$

Performing the same operation on the subsequent blocks generates an encrypted message:

$$c = 1570\ 2756\ 2091\ 2276\ 2423\ 158$$

Decrypting the message requires performing the same exponentiation using the decryption key of 1019, so

$$1570^{1019} \bmod 3337 = 688 = m_1$$

The rest of the message can be recovered in this manner.

RSA in Hardware

Much has been written on the subject of hardware implementations of RSA [1314, 1474,1456,1316,1485,874,1222,87,1410,1409,1343,998,367,1429,523,772]. Good survey articles are [258,872]. Many different chips perform RSA encryption [1310,252, 1101,1317,874,69,737,594,1275,1563,509,1223]. A partial list of currently available RSA chips, from [150,258], is listed in Table 19.3. Not all are available on the open market.

Speed of RSA

In hardware, RSA is about 1000 times slower than DES. The fastest VLSI hardware implementation for RSA with a 512-bit modulus has a throughput of 64 kilobits per second [258]. There are also chips that perform 1024-bit RSA encryption. Currently chips are being planned that will approach 1 megabit per second using a 512-bit modulus; they will probably be available in 1995. Manufacturers have also implemented RSA in smart cards; these implementations are slower.

In software, DES is about 100 times faster than RSA. These numbers may change slightly as technology changes, but RSA will never approach the speed of symmetric algorithms. Table 19.4 gives sample software speeds of RSA [918].

Software Speedups

RSA encryption goes much faster if you're smart about choosing a value of e. The three most common choices are 3, 17, and 65537 ($2^{16} + 1$). (The binary representation of 65537 has only two ones, so it takes only 17 multiplications to exponentiate.) X.509 recommends 65537 [304], PEM recommends 3 [76], and PKCS #1 (see Section 24.14) recommends 3 or 65537 [1345]. There are no security problems with using

Table 19.3
Existing RSA Chips

Company	Clock Speed	Baud Rate Per 512 Bits	Clock Cycles Per 512 Bit Encryption	Technology	Bits per Chip	Number of Transistors
Alpha Techn.	25 MHz	13 K	.98 M	2 micron	1024	180,000
AT&T	15 MHz	19 K	.4 M	1.5 micron	298	100,000
British Telecom	10 MHz	5.1 K	1 M	2.5 micron	256	—
Business Sim. Ltd.	5 MHz	3.8 K	.67 M	Gate Array	32	—
Calmos Syst. Inc.	20 MHz	28 K	.36 M	2 micron	593	95,000
CNET	25 MHz	5.3 K	2.3 M	1 micron	1024	100,000
Cryptech	14 MHz	17 K	.4 M	Gate Array	120	33,000
Cylink	30 MHz	6.8 K	1.2 M	1.5 micron	1024	150,000
GEC Marconi	25 MHz	10.2 K	.67 M	1.4 micron	512	160,000
Pijnenburg	25 MHz	50 K	.256 M	1 micron	1024	400,000
Sandia	8 MHz	10 K	.4 M	2 micron	272	86,000
Siemens	5 MHz	8.5 K	.03 M	1 micron	512	60,000

Table 19.4
RSA Speeds for Different Modulus Lengths with an 8-bit Public Key (on a SPARC II)

	512 bits	768 bits	1,024 bits
Encrypt	0.03 sec	0.05 sec	0.08 sec
Decrypt	0.16 sec	0.48 sec	0.93 sec
Sign	0.16 sec	0.52 sec	0.97 sec
Verify	0.02 sec	0.07 sec	0.08 sec

any of these three values for e (assuming you pad messages with random values—see later section), even if a whole group of users uses the same value for e.

Private key operations can be speeded up with the Chinese remainder theorem if you save the values of p and q, and additional values such as $d \bmod (p-1)$, $d \bmod (q-1)$, and $q^{-1} \bmod p$ [1283,1276]. These additional numbers can easily be calculated from the private and public keys.

Security of RSA

The security of RSA depends wholly on the problem of factoring large numbers. Technically, that's a lie. It is *conjectured* that the security of RSA depends on the problem of factoring large numbers. It has never been mathematically proven that you need to factor n to calculate m from c and e. It is conceivable that an entirely different way to cryptanalyze RSA might be discovered. However, if this new way allows the cryptanalyst to deduce d, it could also be used as a new way to factor large numbers. I wouldn't worry about it too much.

It is also possible to attack RSA by guessing the value of $(p-1)(q-1)$. This attack is no easier than factoring n [1616].

For the ultraskeptical, some RSA variants have been proved to be as difficult as factoring (see Section 19.5). Also look at [36], which shows that recovering even certain bits of information from an RSA-encrypted ciphertext is as hard as decrypting the entire message.

Factoring n is the most obvious means of attack. Any adversary will have the public key, e, and the modulus, n. To find the decryption key, d, he has to factor n. Section 11.4 discusses the current state of factoring technology. Currently, a 129-decimal-digit modulus is at the edge of factoring technology. So, n must be larger than that. Read Section 7.2 on public key length.

It is certainly possible for a cryptanalyst to try every possible d until he stumbles on the correct one. This brute-force attack is even less efficient than trying to factor n.

From time to time, people claim to have found easy ways to break RSA, but to date no such claim has held up. For example, in 1993 a draft paper by William Payne proposed a method based on Fermat's little theorem [1234]. Unfortunately, this method is also slower than factoring the modulus.

There's another worry. Most common algorithms for computing primes p and q are probabilistic; what happens if p or q is composite? Well, first you can make the odds of that happening as small as you want. And if it does happen, the odds are that

encryption and decryption won't work properly—you'll notice right away. There are a few numbers, called Carmichael numbers, which certain probabilistic primality algorithms will fail to detect. These are exceedingly rare, but they are insecure [746]. Honestly, I wouldn't worry about it.

Chosen Ciphertext Attack against RSA

Some attacks work against the implementation of RSA. These are not attacks against the basic algorithm, but against the protocol. It's important to realize that it's not enough to use RSA. Details matter.

Scenario 1: Eve, listening in on Alice's communications, manages to collect a ciphertext message, c, encrypted with RSA in her public key. Eve wants to be able to read the message. Mathematically, she wants m, in which

$$m = c^d$$

To recover m, she first chooses a random number, r, such that r is less than n. She gets Alice's public key, e. Then she computes

$$x = r^e \bmod n$$
$$y = xc \bmod n$$
$$t = r^{-1} \bmod n$$

If $x = r^e \bmod n$, then $r = x^d \bmod n$.

Now, Eve gets Alice to sign y with her private key, thereby decrypting y. (Alice has to sign the message, not the hash of the message.) Remember, Alice has never seen y before. Alice sends Eve

$$u = y^d \bmod n$$

Now, Eve computes

$$tu \bmod n = r^{-1}y^d \bmod n = r^{-1}x^d c^d \bmod n = c^d \bmod n = m$$

Eve now has m.

Scenario 2: Trent is a computer notary public. If Alice wants a document notarized, she sends it to Trent. Trent signs it with an RSA digital signature and sends it back. (No one-way hash functions are used here; Trent encrypts the entire message with his private key.)

Mallory wants Trent to sign a message he otherwise wouldn't. Maybe it has a phony timestamp; maybe it purports to be from another person. Whatever the reason, Trent would never sign it if he had a choice. Let's call this message m'.

First, Mallory chooses an arbitrary value x and computes $y = x^e \bmod n$. He can easily get e; it's Trent's public key and must be public to verify his signatures. Then he computes $m = ym' \bmod n$, and sends m to Trent to sign. Trent returns $m^d \bmod n$. Now Mallory calculates $(m^d \bmod n)x^{-1} \bmod n$, which equals $n'^d \bmod n$ and is the signature of m'.

Actually, Mallory can use several methods to accomplish these same things [423,458,486]. The weakness they all exploit is that exponentiation preserves the multiplicative structure of the input. That is:

$$(xm)^d \bmod n = x^d m^d \bmod n$$

Scenario 3: Eve wants Alice to sign m_3. She generates two messages, m_1 and m_2, such that

$$m_3 \equiv m_1 m_2 \pmod{n}$$

If Eve can get Alice to sign m_1 and m_2, she can calculate m_3:

$$m_3{}^d = (m_1{}^d \bmod n)(m_2{}^d \bmod n)$$

Moral: Never use RSA to sign a random document presented to you by a stranger. Always use a one-way hash function first. The ISO 9796 block format prevents this attack.

Common Modulus Attack on RSA

A possible RSA implementation gives everyone the same n, but different values for the exponents e and d. Unfortunately, this doesn't work. The most obvious problem is that if the same message is ever encrypted with two different exponents (both having the same modulus), and those two exponents are relatively prime (which they generally would be), then the plaintext can be recovered without either of the decryption exponents [1457].

Let m be the plaintext message. The two encryption keys are e_1 and e_2. The common modulus is n. The two ciphertext messages are:

$$c_1 = m^{e_1} \bmod n$$
$$c_2 = m^{e_2} \bmod n$$

The cryptanalyst knows n, e_1, e_2, c_1, and c_2. Here's how he recovers m.

Since e_1 and e_2 are relatively prime, the extended Euclidean algorithm can find r and s, such that

$$re_1 + se_2 = 1$$

Assuming r is negative (either r or s has to be, so just call the negative one r), then the extended Euclidean algorithm can be used again to calculate $c_1{}^{-1}$. Then

$$(c_1{}^{-1})^{-r} * C_2{}^s = m \bmod n$$

There are two other, more subtle, attacks against this type of system. One attack uses a probabilistic method for factoring n. The other uses a deterministic algorithm for calculating someone's secret key without factoring the modulus. Both attacks are described in detail in [449].

Moral: Don't share a common n among a group of users.

Low Encryption Exponent Attack against RSA

RSA encryption and signature verification are faster if you use a low value for e, but that can also be insecure [704]. If you encrypt $e(e + 1)/2$ linearly dependent messages with different public keys having the same value of e, there is an attack against the system. If there are fewer than that many messages, or if the messages are unrelated, there is no problem. If the messages are identical, then e messages are enough. The easiest solution is to pad messages with independent random values.

This also ensures that $m^e \bmod n \neq m^e$. Most real-world RSA implementations—PEM and PGP (see Sections 24.10 and 24.12), for example—do this.

Moral: Pad messages with random values before encrypting them; make sure m is about the same size as n.

Low Decryption Exponent Attack against RSA

Another attack, this one by Michael Wiener, will recover d, when d is up to one quarter the size of n and e is less than n [1596]. This rarely occurs if e and d are chosen at random, and cannot occur if e has a small value.

Moral: Choose a large value for d.

Lessons Learned

Judith Moore lists several restrictions on the use of RSA, based on the success of these attacks [1114,1115]:

- Knowledge of one encryption/decryption pair of exponents for a given modulus enables an attacker to factor the modulus.
- Knowledge of one encryption/decryption pair of exponents for a given modulus enables an attacker to calculate other encryption/decryption pairs without having to factor n.
- A common modulus should not be used in a protocol using RSA in a communications network. (This should be obvious from the previous two points.)
- Messages should be padded with random values to prevent attacks on low encryption exponents.
- The decryption exponent should be large.

Remember, it is not enough to have a secure cryptographic algorithm. The entire cryptosystem must be secure, and the cryptographic protocol must be secure. A failure in any of those three areas makes the overall system insecure.

Attack on Encrypting and Signing with RSA

It makes sense to sign a message before encrypting it (see Section 2.7), but not everyone follows this practice. With RSA, there is an attack against protocols that encrypt before signing [48].

Alice wants to send a message to Bob. First she encrypts it with Bob's public key; then she signs it with her private key. Her encrypted and signed message looks like:

$$(m^{e_B} \bmod n_B)^{d_A} \bmod n_A$$

Here's how Bob can claim that Alice sent him m' and not m. Realize that since Bob knows the factorization of n_B (it's his modulus), he can calculate discrete logarithms with respect to n_B. Therefore, all he has to do is to find an x such that

$$m'^x = m \bmod n_B$$

Then, if he can publish xe_B as his new public exponent and keep n_B as his modulus, he can claim that Alice sent him message m' encrypted in this new exponent.

This is a particularly nasty attack in some circumstances. Note that hash functions don't solve the problem. However, forcing a fixed encryption exponent for every user does.

Standards

RSA is a *de facto* standard in much of the world. The ISO almost, but not quite, created an RSA digital-signature standard; RSA is in an information annex to ISO 9796 [762]. The French banking community standardized on RSA [525], as have the Australians [1498]. The United States currently has no standard for public-key encryption, because of pressure from the NSA and patent issues. Many U.S. companies use PKCS (see Section 24.14), written by RSA Data Security, Inc. A draft ANSI banking standard specifies RSA [61].

Patents

The RSA algorithm is patented in the United States [1330], but not in any other country. PKP licenses the patent, along with other public-key cryptography patents (see Section 25.5). The U.S. patent will expire on September 20, 2000.

19.4 POHLIG-HELLMAN

The Pohlig-Hellman encryption scheme [1253] is similar to RSA. It is not a symmetric algorithm, because different keys are used for encryption and decryption. It is not a public-key scheme, because the keys are easily derivable from each other; both the encryption and decryption keys must be kept secret.

Like RSA,

$$C = P^e \bmod n$$
$$P = C^d \bmod n$$

where

$$ed \equiv 1 \text{ (mod some complicated number)}$$

Unlike RSA, n is not defined in terms of two large primes, it must remain part of the secret key. If someone had e and n, they could calculate d. Without knowledge of e or d, an adversary would be forced to calculate

$$e = \log_P C \bmod n$$

We have already seen that this is a hard problem.

Patents

The Pohlig-Hellman algorithm is patented in the United States [722] and also in Canada. PKP licenses the patent, along with other public-key cryptography patents (see Section 25.5).

19.5 RABIN

Rabin's scheme [1283,1601] gets its security from the difficulty of finding square roots modulo a composite number. This problem is equivalent to factoring. Here is one implementation of this scheme.

First choose two primes, p and q, both congruent to 3 mod 4. These primes are the private key; the product $n = pq$ is the public key.

To encrypt a message, M (M must be less than n), simply compute

$$C = M^2 \bmod n$$

Decrypting the message is just as easy, but slightly more annoying. Since the receiver knows p and q, he can solve the two congruences using the Chinese remainder theorem. Compute

$$m_1 = C^{(p+1)/4} \bmod p$$
$$m_2 = (p - C^{(p+1)/4}) \bmod p$$
$$m_3 = C^{(q+1)/4} \bmod q$$
$$m_4 = (q - C^{(q+1)/4}) \bmod q$$

Then choose an integer $a = q(q^{-1} \bmod p)$ and a integer $b = p(p^{-1} \bmod q)$. The four possible solutions are:

$$M_1 = (am_1 + bm_3) \bmod n$$
$$M_2 = (am_1 + bm_4) \bmod n$$
$$M_3 = (am_2 + bm_3) \bmod n$$
$$M_4 = (am_2 + bm_4) \bmod n$$

One of those four results, M_1, M_2, M_3, or M_4, equals M. If the message is English text, it should be easy to choose the correct M_i. On the other hand, if the message is a random-bit stream (say, for key generation or a digital signature), there is no way to determine which M_i is correct. One way to solve this problem is to add a known header to the message before encrypting.

Williams

Hugh Williams redefined Rabin's schemes to eliminate these shortcomings [1601]. In his scheme, p and q are selected such that

$$p \equiv 3 \bmod 8$$
$$q \equiv 7 \bmod 8$$

and

$$N = pq$$

Also, there is a small integer, S, such that $J(S,N) = -1$. (J is the Jacobi symbol—see Section 11.3). N and S are public. The secret key is k, such that

$$k = 1/2 * (1/4 * (p - 1) * (q - 1) + 1)$$

To encrypt a message M, compute c_1 such that $J(M,N) = (-1)^{c_1}$. Then, compute $M' = (S^{c_1} * M) \bmod N$. Like Rabin's scheme, $C = M'^2 \bmod N$. And $c_2 = M' \bmod 2$. The final ciphertext message is the triple:

$$(C, c_1, c_2)$$

To decrypt C, the receiver computes M'' using

$$C^k \equiv \pm M'' \pmod N$$

The proper sign of M'' is given by c_2. Finally,

$$M = (S^{c_1} * (-1)^{c_1} * M'') \bmod N$$

Williams refined this scheme further in [1603,1604,1605]. Instead of squaring the plaintext message, cube it. The large primes must be congruent to 1 mod 3; otherwise the public and private keys are the same. Even better, there is only one unique decryption for each encryption.

Both Rabin and Williams have an advantage over RSA in that they are provably as secure as factoring. However, they are completely insecure against a chosen-ciphertext attack. If you are going to use these schemes in instances where an attacker can mount this attack (for example, as a digital signature algorithm where an attacker can choose messages to be signed), be sure to use a one-way hash function before signing. Rabin suggested another way of defeating this attack: Append a different random string to each message before hashing and signing. Unfortunately, once you add a one-way hash function to the system it is no longer provably as secure as factoring [628], although adding hashing cannot weaken the system in any practical sense.

Other Rabin variants are [972,909,696,697,1439,989]. A two-dimensional variant is in [866,889].

19.6 ELGAMAL

The ElGamal scheme [518,519] can be used for both digital signatures and encryption; it gets its security from the difficulty of calculating discrete logarithms in a finite field.

To generate a key pair, first choose a prime, p, and two random numbers, g and x, such that both g and x are less than p. Then calculate

$$y = g^x \bmod p$$

The public key is y, g, and p. Both g and p can be shared among a group of users. The private key is x.

ElGamal Signatures

To sign a message, M, first choose a random number, k, such that k is relatively prime to $p - 1$. Then compute

$$a = g^k \bmod p$$

and use the extended Euclidean algorithm to solve for b in the following equation:

$$M = (xa + kb) \bmod (p - 1)$$

The signature is the pair: a and b. The random value, k, must be kept secret.
To verify a signature, confirm that

$$y^a a^b \bmod p = g^M \bmod p$$

Each ElGamal signature or encryption requires a new value of k, and that value must be chosen randomly. If Eve ever recovers a k that Alice used, she can recover Alice's private key, x. If Eve ever gets two messages signed or encrypted using the same k, even if she doesn't know what it is, she can recover x.

This is summarized in Table 19.5.

For example, choose $p = 11$ and $g = 2$. Choose private key $x = 8$. Calculate

$$y = g^x \bmod p = 2^8 \bmod 11 = 3$$

The public key is $y = 3$, $g = 2$, and $p = 11$.

To authenticate $M = 5$, first choose a random number $k = 9$. Confirm that gcd(9,10) = 1. Compute

$$a = g^k \bmod p = 2^9 \bmod 11 = 6$$

and use the extended Euclidean algorithm to solve for b:

$$M = (ax + kb) \bmod (p - 1)$$
$$5 = (8 * 6 + 9 * b) \bmod 10$$

The solution is $b = 3$, and the signature is the pair: $a = 6$ and $b = 3$.

Table 19.5
ElGamal Signatures

	Public Key:
p	prime (can be shared among a group of users)
g	$< p$ (can be shared among a group of users)
y	$= g^x \bmod p$
	Private Key:
x	$< p$
	Signing:
k	choose at random, relatively prime to $p - 1$
a	(signature) $= g^k \bmod p$
b	(signature) such that $M = (xa + kb) \bmod (p - 1)$
	Verifying:
Accept as valid if $y^a a^b \bmod p = g^M \bmod p$	

To verify a signature, confirm that

$$y^a a^b \bmod p = g^M \bmod p$$
$$3^6 6^3 \bmod 11 = 2^5 \bmod 11$$

A variant of ElGamal for signatures is in [1377]. Thomas Beth invented a variant of the ElGamal scheme suitable for proofs of identity [146]. There are variants for password authentication [312], and for key exchange [773]. And there are thousands more (see Section 20.4).

ElGamal Encryption

A modification of ElGamal can encrypt messages. To encrypt message M, first choose a random k, such that k is relatively prime to $p - 1$. Then compute

$$a = g^k \bmod p$$
$$b = y^k M \bmod p$$

The pair, a and b, is the ciphertext. Note that the ciphertext is twice the size of the plaintext.

To decrypt a and b, compute

$$M = b/a^x \bmod p$$

Since $a^x \equiv g^{kx} \pmod p$, and $b/a^x \equiv y^k M/a^x \equiv g^{xk} M/g^{xk} \equiv M \pmod p$, this all works (see Table 19.6). This is really the same as Diffie-Hellman key exchange (see Section 22.1), except that y is part of the key, and the encryption is multiplied by y^k.

Speed

Table 19.7 gives sample software speeds of ElGamal [918].

Table 19.6
ElGamal Encryption

	Public Key:
p	prime (can be shared among a group of users)
g	$< p$ (can be shared among a group of users)
y	$= g^x \bmod p$
	Private Key:
x	$< p$
	Encrypting:
k	choose at random, relatively prime to $p - 1$.
a (ciphertext)	$= g^k \bmod p$
b (ciphertext)	$= y^k M \bmod p$
	Decrypting:
M (plaintext)	$= b/a^x \bmod p$

Patents

ElGamal is unpatented. But, before you go ahead and implement the algorithm, realize that PKP feels that this algorithm is covered under the Diffie-Hellman patent [718]. However, the Diffie-Hellman patent will expire on April 29, 1997, making ElGamal the first public-key cryptography algorithm suitable for encryption and digital signatures unencumbered by patents in the United States. I can hardly wait.

19.7 McEliece

In 1978 Robert McEliece developed a public-key cryptosystem based on algebraic coding theory [1041]. The algorithm makes use of the existence of a class of error-correcting codes, known as **Goppa codes**. His idea was to construct a Goppa code and disguise it as a general linear code. There is a fast algorithm for decoding Goppa codes, but the general problem of finding a code word of a given weight in a linear binary code is **NP-complete**. A good description of this algorithm can be found in [1233]; see also [1562]. Following is just a quick summary.

Let $d_H(x,y)$ denote the Hamming distance between x and y. The numbers n, k, and t are system parameters.

The private key has three parts: G' is a $k * n$ generator matrix for a Goppa code that can correct t errors. P is an $n * n$ permutation matrix. S is a $k * k$ nonsingular matrix.

The public key is a $k * n$ matrix G: $G = SG'P$.

Plaintext messages are strings of k bits, in the form of k-element vectors over GF(2).

To encrypt a message, choose a random n-element vector over GF(2), z, with Hamming distance less than or equal to t.

$$c = mG + z$$

To decrypt the ciphertext, first compute $c' = cP^{-1}$. Then, using the decoding algorithm for the Goppa code, find m' such that $d_H(m'G, c')$ is less than or equal to t. Finally, compute $m = m'S^{-1}$.

In his original paper, McEliece suggested that $n = 1024$, $t = 50$, and $k = 524$. These are the minimum values required for security.

Table 19.7
ElGamal Speeds for Different Modulus Lengths with a 160-bit Exponent (on a SPARC II)

	512 bits	768 bits	1024 bits
Encrypt	0.33 sec	0.80 sec	1.09 sec
Decrypt	0.24 sec	0.58 sec	0.77 sec
Sign	0.25 sec	0.47 sec	0.63 sec
Verify	1.37 sec	5.12 sec	9.30 sec

Although the algorithm was one of the first public-key algorithms, and there were no successful cryptanalytic results against the algorithm, it has never gained wide acceptance in the cryptographic community. The scheme is two to three orders of magnitude faster than RSA, but has some problems. The public key is enormous: 2^{19} bits long. The data expansion is large: The ciphertext is twice as long as the plaintext.

Some attempts at cryptanalysis of this system can be found in [8,943,1559,306]. None of these were successful in the general case, although the similarity between the McEliece algorithm and knapsacks worried some.

In 1991, two Russian cryptographers claimed to have broken the McEliece system with some parameters [882]. Their paper contained no evidence to substantiate their claim, and most cryptographers discount the result. Another Russian attack, one that cannot be used directly against the McEliece system, is in [1447,1448]. Extensions to McEliece can be found in [424,1227,976].

Other Algorithms Based on Linear Error-Correcting Codes

The Niederreiter algorithm [1167] is closely related to the McEliece algorithm, and assumes that the public key is a random parity-check matrix of an error-correcting code. The private key is an efficient decoding algorithm for this matrix.

Another algorithm, used for identification and digital signatures, is based on syndrome decoding [1501]; see [306] for comments. An algorithm based on error-correcting codes [1621] is insecure [698,33,31,1560,32].

19.8 ELLIPTIC CURVE CRYPTOSYSTEMS

Elliptic curves have been studied for many years and there is an enormous amount of literature on the subject. In 1985, Neal Koblitz and V. S. Miller independently proposed using them for public-key cryptosystems [867,1095]. They did not invent a new cryptographic algorithm with elliptic curves over finite fields, but they implemented existing public-key algorithms, like Diffie-Hellman, using elliptic curves.

Elliptic curves are interesting because they provide a way of constructing "elements" and "rules of combining" that produce groups. These groups have enough familiar properties to build cryptographic algorithms, but they don't have certain properties that may facilitate cryptanalysis. For example, there is no good notion of "smooth" with elliptic curves. That is, there is no set of small elements in terms of which a random element has a good chance of being expressed by a simple algorithm. Hence, index calculus discrete logarithm algorithms do not work. See [1095] for more details.

Elliptic curves over the finite field GF(2^n) are particularly interesting. The arithmetic processors for the underlying field are easy to construct and are relatively simple to implement for n in the range of 130 to 200. They have the potential to provide faster public-key cryptosystems with smaller key sizes. Many public-key algorithms, like Diffie-Hellman, ElGamal, and Schnorr, can be implemented in elliptic curves over finite fields.

The mathematics here are complex and beyond the scope of this book. Those interested in this topic are invited to read the two references previously mentioned,

and the excellent book by Alfred Menezes [1059]. Two analogues of RSA work in elliptic curves [890,454]. Other papers are [23,119,1062,869,152,871,892,25,895,353, 1061,26,913,914,915]. Elliptic curve cryptosystems with small key lengths are discussed in [701]. Next Computer Inc.'s Fast Elliptic Encryption (FEE) algorithm also uses elliptic curves [388]. FEE has the nice feature that the private key can be any easy-to-remember string. There are proposed public-key cryptosystems using hyperelliptic curves [868,870,1441,1214].

19.9 LUC

Some cryptographers have developed generalizations of RSA that use various permutation polynomials instead of exponentiation. A variation called Kravitz-Reed, using irreducible binary polynomials [898], is insecure [451,589]. Winfried Müller and Wilfried Nöbauer use Dickson polynomials [1127,1128,965]. Rudolph Lidl and Müller generalized this approach in [966,1126] (a variant is called the Réidi scheme), and Nöbauer looked at its security in [1172,1173]. (Comments on prime generation with Lucas functions are in [969,967,968,598].) Despite all of this prior art, a group of researchers from New Zealand managed to patent this scheme in 1993, calling it LUC [1486,521,1487].

The nth Lucas number, $V_n(P,1)$, is defined as

$$V_n(P,1) = PV_{n-1}(P,1) - V_{n-2}(P,1)$$

There's a lot more theory to Lucas numbers; I'm ignoring all of it. A good theoretical treatment of Lucas sequences is in [1307,1308]. A particularly nice description of the mathematics of LUC is in [1494,708].

In any case, to generate a public-key/private-key key pair, first choose two large primes, p and q. Calculate n, the product of p and q. The encryption key, e, is a random number that is relatively prime to $p-1$, $q-1$, $p+1$, and $q+1$.

There are four possible decryption keys,

$$d = e^{-1} \bmod (\text{lcm}((p+1),(q+1)))$$
$$d = e^{-1} \bmod (\text{lcm}((p+1),(q-1)))$$
$$d = e^{-1} \bmod (\text{lcm}((p-1),(q+1)))$$
$$d = e^{-1} \bmod (\text{lcm}((p-1),(q-1)))$$

where lcm is the least common multiple.

The public key is d and n; the private key is e and n. Discard p and q.

To encrypt a message, P (P must be less than n), calculate

$$C = V_e(P,1) \pmod n$$

And to decrypt:

$$P = V_d(P,1) \pmod n, \text{ with the proper } d$$

At best, LUC is no more secure than RSA. And a recent, still-unpublished results show how to break LUC in at least some implementations. I just don't trust it.

19.10 FINITE AUTOMATON PUBLIC-KEY CRYPTOSYSTEMS

Chinese cryptographer Tao Renji has developed a public-key algorithm based on finite automata [1301,1302,1303,1300,1304,666]. Just as it is hard to factor the product of two large primes, it is also hard to factor the composition of two finite automata. This is especially so if one or both of them is nonlinear.

Much of this research took place in China in the 1980s and was published in Chinese. Renji is starting to write in English. His main result was that certain nonlinear automata (the quasilinear automata) possess weak inverses if, and only if, they have a certain echelon matrix structure. This property disappears if they are composed with another automaton (even a linear one). In the public-key algorithm, the secret key is an invertible quasilinear automaton and a linear automaton, and the corresponding public key can be derived by multiplying them out term by term. Data is encrypted by passing it through the public automaton, and decrypted by passing it through the inverses of its components (in some cases provided they have been set to a suitable initial state). This scheme works for both encryption and digital signatures.

The performance of such systems can be summed up by saying that like McEliece's system, they run much faster than RSA, but require longer keys. The keylength thought to give similar security to 512-bit RSA is 2792 bits, and to 1024-bit RSA is 4152 bits. For the former case, the system encrypts data at 20,869 bytes/sec and decrypts data at 17,117 bytes/sec, running on a 33 MHz 80486.

Renji has published three algorithms. The first is FAPKC0. This is a weak system which uses linear components, and is primarily illustrative. Two serious systems, FAPKC1 and FAPKC2, use one linear and one nonlinear component each. The latter is more complex, and was developed in order to support identity-based operation.

As for their strength, quite a lot of work has been done on them in China (where there are now over 30 institutes publishing cryptography and security papers). One can see from the considerable Chinese language literature that the problem has been studied.

One possible attraction of FAPKC1 and FAPKC2 is that they are not encumbered by any U.S. patents. Thus, once the Diffie-Hellman patent expires in 1997, they will unquestionably be in the public domain.

CHAPTER 20

Public-Key Digital Signature Algorithms

20.1 DIGITAL SIGNATURE ALGORITHM (DSA)

In August 1991, The National Institute of Standards and Technology (NIST) proposed the Digital Signature Algorithm (DSA) for use in their Digital Signature Standard (DSS). According to the *Federal Register* [538]:

> A Federal Information Processing Standard (FIPS) for Digital Signature Standard (DSS) is being proposed. This proposed standard specifies a public-key digital signature algorithm (DSA) appropriate for Federal digital signature applications. The proposed DSS uses a public key to verify to a recipient the integrity of data and identity of the sender of the data. The DSS can also be used by a third party to ascertain the authenticity of a signature and the data associated with it.
>
> This proposed standard adopts a public-key signature scheme that uses a pair of transformations to generate and verify a digital value called a signature.

And:

> This proposed FIPS is the result of evaluating a number of alternative digital signature techniques. In making the selection NIST has followed the mandate contained in section 2 of the Computer Security Act of 1987 that NIST develop standards to ". . . assure the cost-effective security and privacy of Federal information and, among technologies offering comparable protection, on selecting the option with the most desirable operating and use characteristics."
>
> Among the factors that were considered during this process were the level of security provided, the ease of implementation in both hardware and software, the ease of export from the U.S., the applicability of patents, impact on national security and law enforcement and the level of efficiency in both the signing and verification functions. A number of techniques were deemed to provide appropriate protection for Federal systems. The technique selected has the following desirable characteristics:

> NIST expects it to be available on a royalty-free basis. Broader use of this technique resulting from public availability should be an economic benefit to the government and the public.
>
> The technique selected provides for efficient implementation of the signature operations in smart card applications. In these applications the signing operations are performed in the computationally modest environment of the smart card while the verification process is implemented in a more computationally rich environment such as a personal computer, a hardware cryptographic module, or a mainframe computer.

Before it gets too confusing, let me review the nomenclature: DSA is the algorithm; the DSS is the standard. The standard employs the algorithm. The algorithm is part of the standard.

Reaction to the Announcement

NIST's announcement created a maelstrom of criticisms and accusations. Unfortunately, it was more political than academic. RSA Data Security, Inc., purveyors of the RSA algorithm, led the criticism against DSS. They wanted RSA, and not another algorithm, used as the standard. RSADSI makes a lot of money licensing the RSA algorithm, and a royalty-free digital signature standard would directly affect their bottom line. (Note: DSA is not necessarily free of patent infringements; I'll discuss that later.)

Before the algorithm was announced, RSADSI campaigned against a "common modulus," which might have given the government the ability to forge signatures. When the algorithm was announced without this common modulus, they attacked it on other grounds [154], both in letters to NIST and statements to the press. (Four letters to NIST appeared in [1326]. When reading them, keep in mind that at least two of the authors, Rivest and Hellman, had a financial interest in DSS's not being approved.)

Many large software companies that already licensed the RSA algorithm came out against the DSS. In 1982, the government had solicited public-key algorithms for a standard [537]. After that, there wasn't a peep out of NIST for nine years. Companies such as IBM, Apple, Novell, Lotus, Northern Telecom, Microsoft, DEC, and Sun had already spent large amounts of money implementing the RSA algorithm. They were not interested in losing their investment.

In all, NIST received 109 comments by the end of the first comment period on February 28, 1992.

Let's look at the criticisms against DSA, one by one.

1. DSA cannot be used for encryption or key distribution.

   True, but not the point of the standard. This is a signature standard. NIST should have a standard for public-key encryption. NIST is committing a grave injustice to the American people by not implementing a public-key

encryption standard. It is suspicious that this proposed digital signature standard cannot be used for encryption. (As it turns out, though, it can—see Section 23.3.) That does not mean that a signature standard is useless.

2. DSA was developed by the NSA, and there may be a trapdoor in the algorithm.

   Much of the initial comments were just paranoia: "NIST's denial of information with no apparent justification does not inspire confidence in DSS, but intensifies concern that there is a hidden agenda, such as laying the groundwork for a national public-key cryptosystem that is in fact vulnerable to being broken by NIST and/or NSA" [154]. One serious question about the security of DSA was raised by Arjen Lenstra and Stuart Haber at Bellcore. This will be discussed later.

3. DSA is slower than RSA [800].

   True, more or less. Signature generation speeds are the same, but signature verification can be 10 to 40 times slower with DSA. Key generation, however, is faster. But key generation is irrelevant; a user rarely does it. On the other hand, signature verification is the most common operation.

   The problem with this criticism is that there are many ways to play with the test parameters, depending on the results you want. Precomputations can speed up DSA signature generation, but don't always apply. Proponents of RSA use numbers optimized to make their calculations easier; proponents of DSA use their own optimizations. In any case, computers are getting faster all the time. While there is a speed difference, it will not be noticeable in most applications.

4. RSA is a *de facto* standard.

   Here are two examples of this complaint. From Robert Follett, the program director of standards at IBM [570]:

   > IBM is concerned that NIST has proposed a standard with a different digital signature scheme rather than adopting the international standard. We have been convinced by users and user organizations that the international standards using RSA will be a prerequisite to the sales of security products in the very near future.

   From Les Shroyer, vice president and director, corporate MIS and telecommunications, at Motorola [1444]:

   > We must have a single, robust, politically-accepted digital signature standard that is usable throughout the world, between both U.S. and non-U.S., and Motorola and non-Motorola entities. The lack of other viable digital signature technology for the last eight years has made RSA a de facto standard. . . . Motorola and many other companies . . . have committed millions of dollars to RSA. We have concern over the interoperability and support of two different standards, as that situation will lead to added costs, delays in deployment, and complication. . . .

Many companies wanted NIST to adopt the ISO 9796, the international digital signature standard that uses RSA [762]. While this is a valid complaint, it is not a sufficient justification to make it a standard. A royalty-free standard would better serve the U.S. public interest.

5. The DSA selection process was not public; sufficient time for analysis has not been provided.

   First NIST claimed that they designed the DSA; then they admitted that NSA helped them. Finally, they confirmed that NSA designed the algorithm. This worries many people; the NSA doesn't inspire trust. Even so, the algorithm is public and available for analysis; and NIST extended the time for analysis and comment.

6. DSA may infringe on other patents.

   It may. This will be discussed in the section on patent issues.

7. The key size is too small.

   This was the only valid criticism of DSS. The original implementation set the modulus at 512 bits [1149]. Since the algorithm gets its security from the difficulty of computing discrete logs in that modulus, this worried most cryptographers. There have since been advances in the problem of calculating discrete logarithms in a finite field, and 512 bits is too short for long-term security (see Section 7.2). According to Brian LaMacchia and Andrew Odlyzko, ". . . even 512-bit primes appear to offer only marginal security . . ." [934]. In response to this criticism, NIST made the key size variable, from 512 bits to 1024 bits. Not great, but better.

On May 19, 1994, the standard was finally issued [1154]. The issuing statement said [542]:

> This standard is applicable to all Federal departments and agencies for the protection of unclassified information. . . . This standard shall be used in designing and implementing public-key based signature schemes which Federal departments and agencies operate or which are operated for them under contract. Adoption and use of this standard is available to private and commercial organizations.

Before you run out and implement this standard in your next product, read the section on patent issues below.

Description of DSA

DSA is a variant of the Schnorr and ElGamal signature algorithms, and is fully described in [1154]. The algorithm uses the following parameters:

> p = a prime number L bits long, when L ranges from 512 to 1024 and is a multiple of 64. (In the original standard, the size of p was fixed at 512 bits [1149]. This was the source of much criticism and was changed by NIST [1154].)

q = a 160-bit prime factor of $p - 1$.

$g = h^{(p-1)/q} \bmod p$, where h is any number less than $p - 1$ such that $h^{(p-1)/q} \bmod p$ is greater than 1.

x = a number less than q.

$y = g^x \bmod p$.

The algorithm also makes use of a one-way hash function: $H(m)$. The standard specifies the Secure Hash Algorithm, discussed in Section 18.7.

The first three parameters, p, q, and g, are public and can be common across a network of users. The private key is x; the public key is y.

To sign a message, m:

(1) Alice generates a random number, k, less than q.

(2) Alice generates

$$r = (g^k \bmod p) \bmod q$$
$$s = (k^{-1}(H(m) + xr)) \bmod q$$

The parameters r and s are her signature; she sends these to Bob.

(3) Bob verifies the signature by computing

$$w = s^{-1} \bmod q$$
$$u_1 = (H(m) * w) \bmod q$$
$$u_2 = (rw) \bmod q$$
$$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$$

If $v = r$, then the signature is verified.

Proofs for the mathematical relationships are found in [1154]. Table 20.1 provides a summary.

Speed Precomputations

Table 20.2 gives sample software speeds of DSA [918].

Real-world implementations of DSA can often be speeded up through precomputations. Notice that the value r does not depend on the message. You can create a string of random k values, and then recompute r values for each of them. You can also recompute k^{-1} for each of those k values. Then, when a message comes along, you can compute s for a given r and k^{-1}.

This precomputation speeds up DSA considerably. Table 20.3 is a comparison of DSA and RSA computation times for a particular smart card implementation [1479].

Table 20.1
DSA Signatures

Public Key:

p 512-bit to 1024-bit prime (can be shared among a group of users)
q 160-bit prime factor of $p - 1$ (can be shared among a group of users)
g $= h^{(p-1)/q} \bmod p$, where h is less than $p - 1$ and $h^{(p-1)/q} \bmod p > 1$ (can be shared among a group of users)
y $= g^x \bmod p$ (a p-bit number)

Private Key:

x $< q$ (a 160-bit number)

Signing:

k choose at random, less than q
r (signature) $= (g^k \bmod p) \bmod q$
s (signature) $= (k^{-1} (H(m) + xr)) \bmod q$

Verifying:

$w = s^{-1} \bmod q$
$u_1 = (H(m) * w) \bmod q$
$u_2 = (rw) \bmod q$
$v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$
If $v = r$, then the signature is verified.

DSA Prime Generation

Lenstra and Haber pointed out that certain moduli are much easier to crack than others [950]. If someone forced a network to use one of these "cooked" moduli, then their signatures would be easier to forge. This isn't a problem for two reasons: These moduli are easy to detect and they are so rare that the chances of using one when choosing a modulus randomly are almost negligible—smaller, in fact, than the chances of accidentally generating a composite number using a probabilistic prime generation routine.

In [1154] NIST recommended a specific method for generating the two primes, p and q, where q divides $p - 1$. The prime p is L bits long, between 512 and 1024 bits

Table 20.2
DSA Speeds for Different Modulus Lengths with a 160-bit Exponent (on a SPARC II)

	512 bits	768 bits	1024 bits
Sign	0.20 sec	0.43 sec	0.57 sec
Verify	0.35 sec	0.80 sec	1.27 sec

Table 20.3
Comparison of RSA and DSA Computation Times

	DSA	RSA	DSA with Common p, q, g
Global Computations	Off-card (P)	N/A	Off-card (P)
Key Generation	14 sec	Off-card (S)	4 sec
Precomputation	14 sec	N/A	4 sec
Signature	.03 sec	15 sec	.03 sec
Verification	16 sec	1.5 sec	10 sec
	1–5 sec off-card (P)	1–3 sec off-card (P)	

Off-card computations were performed on an 80386 33 MHz, personal computer. (P) indicates public parameters off-card and (S) indicates secret parameters off-card. Both algorithms use a 512-bit modulus.

long, in some multiple of 64 bits. The prime q is 160 bits long. Let $L - 1 = 160n + b$, where L is the length of p, and n and b are two numbers and b is less than 160.

(1) Choose an arbitrary sequence of at least 160 bits and call it S. Let g be the length of S in bits.

(2) Compute $U = \text{SHA}(S) \oplus \text{SHA}((S + 1) \bmod 2^g)$, where SHA is the Secure Hash Algorithm (see Section 18.7).

(3) Form q by setting the most significant bit and the least significant bit of U to 1.

(4) Check whether q is prime.

(5) If q is not prime, go back to step (1).

(6) Let $C = 0$ and $N = 2$.

(7) For $k = 0, 1, \ldots, n$, let $V_k = \text{SHA}((S + N + k) \bmod 2^g)$

(8) Let W be the integer

$$W = V_0 + 2^{160}V_1 + \ldots + 2^{160(n-1)}V_{n-1} + 2^{160n}(V_n \bmod 2^b)$$

and let

$$X = W + 2^{L-1}$$

Note that X is an L-bit number.

(9) Let $p = X - ((X \bmod 2q) - 1)$. Note that p is congruent to 1 mod $2q$.

(10) If $p < 2^{L-1}$, then go to step (13).

(11) Check whether p is prime.

(12) If p is prime, go to step (15).

(13) Let $C = C + 1$ and $N = N + n + 1$.

(14) If $C = 4096$, then go to step (1). Otherwise, go to step (7).

(15) Save the value of S and the value of C used to generate p and q.

In [1154], the variable S is called the "seed," C is called the "counter," and N the "offset."

The point of this exercise is that there is a public means of generating p and q. For all practical purposes, this method prevents cooked values of p and q. If someone hands you a p and a q, you might wonder where that person got them. However, if someone hands you a value for S and C that generated the random p and q, you can go through this routine yourself. Using a one-way hash function, SHA in the standard, prevents someone from working backwards from a p and q to generate an S and C.

This security is better than what you get with RSA. In RSA, the prime numbers are kept secret. Someone could generate a fake prime or one of a special form that makes factoring easier. Unless you know the private key, you won't know that. Here, even if you don't know a person's private key, you can confirm that p and q have been generated randomly.

ElGamal Encryption with DSA

There have been allegations that the government likes the DSA because it is only a digital signature algorithm and can't be used for encryption. It is, however, possible to use the DSA function call to do ElGamal encryption.

Assume that the DSA algorithm is implemented with a single function call:

```
DSAsign (p,q,g,k,x,h,r,s)
```

You supply the numbers p, q, g, k, x, and h, and the function returns the signature parameters: r and s.

To do ElGamal encryption of message m with public key y, choose a random number, k, and call

```
DSAsign (p,p,g,k,0,0,r,s)
```

The value of r returned is a in the ElGamal scheme. Throw s away. Then, call

```
DSAsign (p,p,y,k,0,0,r,s)
```

Rename the value of r to be u; throw s away. Call

```
DSAsign (p,p,m,1,u,0,r,s)
```

Throw r away. The value of s returned is b in the ElGamal scheme. You now have the ciphertext, a and b.

Decryption is just as easy. Using secret key x, and ciphertext messages a and b, call

```
DSAsign (p,p,a,x,0,0,r,s)
```

The value r is $a^x \bmod p$. Call that e. Then call

```
DSAsign (p,p,1,e,b,0,r,s)
```

The value s is the plaintext message, m.

This method will not work with all implementations of DSA. Some may fix the values of p and q, or the lengths of some of the other parameters. Still, if the implementation is general enough, this is a way to encrypt using nothing more than digital signature function.

RSA Encryption with DSA

RSA encryption is even easier. With a modulus n, message m, and public key e, call

```
DSAsign (n,n,m,e,0,0,r,s)
```

The value of r returned is the ciphertext.

RSA decryption is the same thing. If d is the private key, then

```
DSAsign (n,n,m,d,0,0,r,s)
```

returns the plaintext as the value of r.

Security of DSA

At 512-bits, DSA wasn't strong enough for long-term security. At 1024 bits, it is.

The NSA, in its first public interview on the subject, commented to Joe Abernathy of *The Houston Chronicle* on allegations about a trapdoor in DSS [363]:

> Regarding the alleged trapdoor in the DSS. We find the term trapdoor somewhat misleading since it implies that the messages sent by the DSS are encrypted and with access via a trapdoor one could somehow decrypt (read) the message without the sender's knowledge.
>
> The DSS does not encrypt any data. The real issue is whether the DSS is susceptible to someone forging a signature and therefore discrediting the entire system. We state categorically that the chances of anyone—including NSA—forging a signature with the DSS when it is properly used and implemented is infinitesimally small.
>
> Furthermore, the alleged trapdoor vulnerability is true for *any* public key-based authentication system, including RSA. To imply somehow that this only affects the DSS (a popular argument in the press) is totally misleading. The issue is one of implementation and how one goes about selecting prime numbers. We call your attention to a recent EUROCRYPT conference which had a panel discussion on the issue of trapdoors in the DSS. Included on the panel was one of the Bellcore researchers who initially raised the trapdoor allegation, and our understanding is that the panel—including the person from Bellcore—concluded that the alleged trapdoor was not an issue for the DSS. Furthermore, the general consensus appeared to be that the trapdoor issue was trivial and had been overblown in the

press. However, to try to respond to the trapdoor allegation, at NIST's request, we have designed a prime generation process which will ensure that one can avoid selection of the relatively few weak primes which could lead to weakness in using the DSS. Additionally, NIST intends to allow for larger modulus sizes up to 1024 which effectively negates the need to even use the prime generation process to avoid weak primes. An additional very important point that is often overlooked is that with the DSS the primes are *public* and therefore can be subject to public examination. Not all public key systems provide for this same type of examination.

The integrity of any information security system requires attention to proper implementation. With the myriad of vulnerabilities possible given the differences among users, NSA has traditionally insisted on centralized trusted centers as a way to minimize risk to the system. While we have designed technical modifications to the DSS to meet NIST's requests for a more decentralized approach, we still would emphasize that portion of the *Federal Register* notice for the DSS which states:

> "While it is the intent of this standard to specify general security requirements for generating digital signatures, conformance to this standard does not assure that a particular implementation is secure. The responsible authority in each agency or department shall assure that an overall implementation provides an acceptable level of security. NIST will be working with government users to ensure appropriate implementations."

Finally, we have read all the arguments purporting insecurities with the DSS, and we remain unconvinced of their validity. The DSS has been subjected to intense evaluation within NSA which led to its being endorsed by our Director of Information Systems Security for use in signing unclassified data processed in certain intelligence systems and even for signing classified data in selected systems. We believe that this approval speaks to the lack of any credible attack on the integrity provided by the DSS given proper use and implementation. Based on the technical and security requirements of the U.S. government for digital signatures, we believe the DSS is the best choice. In fact, the DSS is being used in a pilot project for the Defense Message System to assure the authenticity of electronic messages of vital command and control information. This initial demonstration includes participation from the Joint Chiefs of Staff, the military services, and Defense Agencies and is being done in cooperation with NIST.

I'm not going to comment on the trustworthiness of the NSA. Take their comments for what you think they're worth.

Attacks against k

Each signature requires a new value of k, and that value must be chosen randomly. If Eve ever recovers a k that Alice used to sign a message, perhaps by exploiting some properties of the random-number generator that generated k, she can recover Alice's private key, x. If Eve ever gets two messages signed using the same k, even if she doesn't know what it is, she can recover x. And with x, Eve can generate undetectable forgeries of Alice's signature. In any implementation of the DSA, a good random-number generator is essential to the system's security [1468].

Dangers of a Common Modulus

Even though the DSS does not specify a common modulus to be shared by everyone, different implementations may. For example, the Internal Revenue Service is considering using the DSS for the electronic submission of tax returns. What if they require every taxpayer in the country to use a common p and q? Even though the standard doesn't require a common modulus, such an implementation accomplishes the same thing. A common modulus too easily becomes a tempting target for cryptanalysis. It is still too early to tell much about different DSS implementations, but there is some cause for concern.

Subliminal Channel in DSA

Gus Simmons discovered a subliminal channel in DSA [1468,1469] (see Section 23.3). This subliminal channel allows someone to embed a secret message in his signature that can only be read by another person who knows the key. According to Simmons, it is a "remarkable coincidence" that the "apparently inherent shortcomings of subliminal channels using the ElGamal scheme can all be overcome" in the DSS, and that the DSS "provides the most hospitable setting for subliminal communications discovered to date." NIST and NSA have not commented on this subliminal channel; no one knows if they even knew about it. Since this subliminal channel allows an unscrupulous implementer of DSS to leak a piece of the private key with each signature, it is important to never use an implementation of DSS if you don't trust the implementer.

Patents

David Kravitz, formerly of the NSA, holds a patent on DSA [897]. According to NIST [538]:

> NIST intends to make this DSS technique available world-wide on a royalty-free basis to the public interest. We believe this technique is patentable and that no other patents would apply to the DSS, but we cannot give firm assurances to such effect in advance of issuance of the patent.

Even so, three patent holders claim that the DSA infringes on their patents: Diffie-Hellman (see Section 22.1) [718], Merkle-Hellman (see Section 19.2) [720], and Schnorr (see Section 21.3) [1398]. The Schnorr patent is the most troublesome. The other two patents expire in 1997; the Schnorr patent is valid until 2008. The Schnorr algorithm was not developed with government money; unlike the PKP patents, the U.S. government has no rights to the Schnorr patent; and Schnorr patented his algorithm worldwide. Even if the U.S. courts rule in favor of DSA, it is unclear what other courts around the world would do. Is an international company going to adopt a standard that may be legal in some countries but infringes on a patent in others? This issue will take time to resolve; at the time of this writing it isn't even resolved in the United States.

In June 1993 NIST proposed to give PKP an exclusive patent license to DSA [541]. The agreement fell through after public outcry and the standard was issued without any deal. NIST said [542]:

> . . . NIST has addressed the possible patent infringement claims, and has concluded that there are no valid claims.

So the standard is official, lawsuits are threatened, and no one knows what to do. NIST has said that it would help defend people sued for patent infringement, if they were using DSA to satisfy a government contract. Everyone else, it seems, is on their own. ANSI has a draft banking standard that uses DSA [60]. NIST is working to standardize DSA within the government. Shell Oil has made DSA their international standard. I know of no other proposed DSA standards.

20.2 DSA VARIANTS

This variant makes computation easier on the signer by not forcing him to compute k^{-1} [1135]. All the parameters are as in DSA. To sign a message, m, Alice generates two random numbers, k and d, both less than q. The signature is

$$r = (g^k \bmod p) \bmod q$$
$$s = (H(m) + xr) * d \bmod q$$
$$t = kd \bmod q$$

Bob verifies the signature by computing

$$w = t/s \bmod q$$
$$u_1 = (H(m) * w) \bmod q$$
$$u_2 = (rw) \bmod q$$

If $r = ((g^{u1} * y^{u2}) \bmod p) \bmod q$, then the signature is verified.

This next variant makes computation easier on the verifier [1040,1629]. All the parameters are as in DSA. To sign a message, m, Alice generates a random number, k, less than q. The signature is

$$r = (g^k \bmod p) \bmod q$$
$$s = k * (H(m) + xr)^{-1} \bmod q$$

Bob verifies the signature by computing

$$u_1 = (H(m) * s) \bmod q$$
$$u_2 = (sr) \bmod q$$

If $r = ((g^{u1} * y^{u2}) \bmod p) \bmod q$, then the signature is verified.

Another DSA variant allows for batch verification; Bob can verify signatures in batches [1135]. If they are all valid, he is done. If one isn't valid, then he still has to find it. Unfortunately, it is not secure; either the signer or the verifier can easily create a set of bogus signatures that satisfy the batch criteria [974].

There is also a variant for DSA prime generation, one that embeds q and the parameters used to generate the primes within p. Whether this scheme reduces the security of DSA is still unknown.

(1) Choose an arbitrary sequence of at least 160 bits and call it S. Let g be the length of S in bits.

(2) Compute $U = \text{SHA}(S) \oplus \text{SHA}\ ((S + 1) \bmod 2^g)$, where SHA is the Secure Hash Algorithm (see Section 18.7).

(3) Form q by setting the most significant bit and the least significant bit of U to 1.

(4) Check whether q is prime.

(5) Let p be the concatenation of q, S, C, and SHA(S). C is set to 32 zero bits.

(6) $p = p - (p \bmod q) + 1$.

(7) $p = p + q$.

(8) If the C in p is $0 \times 7fffffff$, go to step (1).

(9) Check whether p is prime.

(10) If p is composite, go to step (7).

The neat thing about this variant is that you don't have to store the values of C and S used to generate p and q; they are embedded within p. For applications without a whole lot of memory, like smart cards, this can be a big deal.

20.3 GOST Digital Signature Algorithm

This is a Russian digital signature standard, officially called GOST R 34.10-94 [656]. The algorithm is very similar to DSA, and uses the following parameters

p = a prime number, either between 509 and 512 bits long, or between 1020 and 1024 bits long.

q = a 254- to 256-bit prime factor of $p - 1$.

a = any number less than $p - 1$ such that $a^q \bmod p = 1$.

x = a number less than q.

$y = a^x \bmod p$.

The algorithm also makes use of a one-way hash function: $H(x)$. The standard specifies GOST R 34.11-94 (see Section 18.11), a function based on the GOST symmetric algorithm (see Section 14.1) [657].

The first three parameters, p, q, and a, are public and can be common across a network of users. The private key is x; the public key is y.

To sign a message, m

(1) Alice generates a random number, k, less than q

(2) Alice generates

$$r = (a^k \bmod p) \bmod q$$
$$s = (xr + k(H(m))) \bmod q$$

If $H(m) \bmod q = 0$, then set it equal to 1. If $r = 0$, then choose another k and start again. The signature is two numbers: $r \bmod 2^{256}$ and $s \bmod 2^{256}$. She sends these to Bob.

(3) Bob verifies the signature by computing

$$v = H(m)^{q-2} \bmod q$$
$$z_1 = (sv) \bmod q$$
$$z_2 = ((q - r) * v) \bmod q$$
$$u = ((a^{z_1} * y^{z_2}) \bmod p) \bmod q$$

If $u = r$, then the signature is verified.

The difference between this scheme and DSA is that with DSA $s = (xr + k^{-1}(H(m))) \bmod q$, which leads to a different verification equation. Curious, though, is that q is 256 bits. Most Western cryptographers seem satisfied with a q of around 160 bits. Perhaps this is just a reflection of the Russian tendency to play it ultrasafe.

The standard has been in use since the beginning of 1995, and is not classified "for special use"—whatever that means.

20.4 DISCRETE LOGARITHM SIGNATURE SCHEMES

ElGamal, Schnorr (see Section 21.3), and DSA signature schemes are very similar. In fact, they are just three examples of a general digital signature scheme based on the Discrete Logarithm Problem. Along with thousands of other signature schemes, they are part of the same family [740,741,699,1184].

Choose p, a large prime number, and q, either $p - 1$ or a large prime factor of $p - 1$. Then choose g, a number between 1 and p such that $g^q \equiv 1 \pmod{p}$. All these numbers are public, and can be common to a group of users. The private key is x, less than q. The public key is $y = g^x \bmod q$.

To sign a message, m, first choose a random k less than and relatively prime to q. If q is also prime, any k less than q works. First compute

$$r = g^k \bmod p$$

The generalized **signature equation** now becomes

$$ak = b + cx \bmod q$$

The coefficients a, b, and c can be any of a variety of things. Each line in Table 20.4 gives six possibilities.

To verify the signature, the receiver must confirm that

$$r^a = g^b y^c \bmod p$$

This is called the **verification equation**.

Table 20.5 lists the signature and verifications possible from just the first line of potential values for a, b, and c, ignoring the effects of the $\pm$.

Table 20.4
Possible Permutations of a, b, and c ($r' = r \bmod q$)

$\pm r'$	$\pm s$	m
$\pm r'm$	$\pm s$	1
$\pm r'm$	$\pm ms$	1
$\pm mr'$	$\pm r's$	1
$\pm ms$	$\pm r's$	1

That's six different signature schemes. Adding the negative signs brings the total to 24. Using the other possible values listed for a, b, and c brings the total to 120.

ElGamal [518,519] and DSA [1154] are essentially based on equation (4). Other schemes are based on equation (2) [24,1629]. Schnorr [1396,1397] is closely related to equation (5), as is another scheme [1183]. And equation (1) can be modified to yield the scheme proposed in [1630]. The rest of the equations are new.

There's more. You can make any of these schemes more DSA-like by defining r as

$$r = (g^k \bmod p) \bmod q$$

Keep the same signature equation and make the verification equation

$$u_1 = a^{-1}b \bmod q$$
$$u_2 = a^{-1}c \bmod q$$
$$r = (g^{u_1}y^{u_2} \bmod p) \bmod q$$
$$(r \bmod q)^a = g^b y^c \bmod p$$

There are two other possibilities along these lines [740,741]; you can do this with each of the 120 schemes, bringing the total to 480 discrete-logarithm-based digital signature schemes.

But wait—there's more. Additional generalizations and variations can generate more than 13,000 variants (not all of them terribly efficient) [740,741].

One of the nice things about using RSA for digital signatures is a features called **message recovery**. When you verify an RSA signature you compute m. Then you compare the computed m with the message and see if the signature is valid for that

Table 20.5
Discrete Logarithm Signature Schemes

Signature Equation	Verification Equation
(1) $r'k = s + mx \bmod q$	$r^{r'} = g^s y^m \bmod p$
(2) $r'k = m + sx \bmod q$	$r^{r'} = g^m y^s \bmod p$
(3) $sk = r' + mx \bmod q$	$r^s = g^{r'} y^m \bmod p$
(4) $sk = m + r'x \bmod q$	$r^s = g^m y^{r'} \bmod p$
(5) $mk = s + r'x \bmod q$	$r^m = g^s y^{r'} \bmod p$
(6) $mk = r' + sx \bmod q$	$r^m = g^{r'} y^s \bmod p$

message. With the previous schemes, you can't recover m when you compute the signature; you need a candidate m that you use in a verification equation. Well, as it turns out it is possible to construct a message recovery variant for all the above signature schemes.

To sign, first compute

$$r = mg^k \bmod p$$

and replace m by 1 in the signature equation. Then you can reconstruct the verification equation such that m can be computed directly.

You can do the same with the DSA-like schemes:

$$r = (mg^k \bmod p) \bmod q$$

All the variants are equally secure, so it makes sense to choose a scheme that is easy to compute with. The requirement to compute inverses slows most of these schemes. As it turns out, a scheme in this pile allows computing both the signature equation and the verification equation without inverses and also gives message recovery. It is called the **p-NEW** scheme [1184].

$$r = mg^{-k} \bmod p$$
$$s = k - r'x \bmod q$$

And m is recovered (and the signature verified) by

$$m = g^s y^{r'} r \bmod p$$

Some variants sign two and three message blocks at the same time [740]; other variants can be used for blind signatures [741].

This is a remarkable piece of research. All of the various discrete-logarithm-based digital signature schemes have been put in one coherent framework. In my opinion this finally puts to rest any patent dispute between Schnorr [1398] and DSA [897]: DSA is not a derivative of Schnorr, nor even of ElGamal. All three are examples of this general construction, and this general construction is unpatented.

20.5 ONG-SCHNORR-SHAMIR

This signature scheme uses polynomials modulo n [1219,1220]. Choose a large integer n (you need not know the factorization of n). Then choose a random integer, k, such that k and n are relatively prime. Calculate h such that

$$h = -k^{-2} \bmod n = -(k^{-1})^2 \bmod n$$

The public key is h and n; k is the private key.

To sign a message, M, first generate a random number, r, such that r and n are relatively prime. Then calculate:

$$S_1 = 1/2 * (M/r + r) \bmod n$$
$$S_2 = k/2 * (M/r - r) \bmod n$$

The pair, S_1 and S_2, is the signature.

To verify a signature, confirm that

$$S_1^2 + h * S_2^2 \equiv M \pmod{n}$$

The version of the scheme described here is based on quadratic polynomials. When it was first proposed in [1217], a $100 reward was offered for successful cryptanalysis. It was proved insecure [1255,18], but its authors were not deterred. They proposed a modification of the algorithm based on cubic polynomials, which is also insecure [1255]. The authors then proposed a quartic version, which was also broken [524,1255]. A variant which fixes these problems is in [1134].

20.6 ESIGN

ESIGN is a digital signature scheme from NTT Japan [1205,583]. It is touted as being at least as secure and considerably faster than either RSA or DSA, with similar key and signature lengths.

The private key is a pair of large prime numbers, p and q. The public key is n, when

$$n = p^2q$$

H is a hash function that operates on a message, m, such that $H(m)$ is between 0 and $n - 1$. There is also a security parameter, k, which will be discussed shortly.

(1) Alice picks a random number x, where x is less than pq.

(2) Alice computes:

$$w, \text{ the least integer that is larger than or equal to}$$
$$(H(m) - x^k \bmod n)/pq$$
$$s = x + ((w/kx^{k-1}) \bmod p)pq$$

(3) Alice sends s to Bob.

(4) To verify the signature, Bob computes $s^k \bmod n$. He also computes a, which is the least integer larger than or equal to two times the number of bits of n divided by 3. If $H(m)$ is less than or equal to $s^k \bmod n$, and if $s^k \bmod n$ is less than $H(m) + 2^a$, then the signature is considered valid.

This algorithm works faster with precomputation. This precomputation can be done at any time and has nothing to do with the message being signed. After picking x, Alice could break step (2) into two partial steps. The first can be precomputed.

(2a) Alice computes:

$$u = x^k \bmod n$$
$$v = 1/(kx^{k-1}) \bmod p$$

(2b) Alice computes:

w = the least integer that is larger than or equal to
$(H(m) - u)/pq)$
$s = x + (wv \bmod p)pq$

For the size of numbers generally used, this precomputation speeds up the signature process by a factor of 10. Almost all the hard work is done in the precomputation stage. A discussion of modular arithmetic operations to speed ESIGN can be found in [1625,1624]. This algorithm can also be extended to work with elliptic curves [1206].

Security of ESIGN

When this algorithm was originally proposed, k was set to 2 [1215]. This was quickly broken by Ernie Brickell and John DeLaurentis [261], who then extended their attack to $k = 3$. A modified version of this algorithm [1203] was broken by Shamir [1204]. The variant proposed in [1204] was broken in [1553]. ESIGN is the current incarnation of this family of algorithms. Another new attack [963] does not work against ESIGN.

The authors currently recommend these values for k: 8, 16, 32, 64, 128, 256, 512, and 1024. They also recommend that p and q each be of at least 192 bits, making n at least 576 bits long. (I think n should be twice that length.) With these parameters, the authors conjecture that ESIGN is as secure as RSA or Rabin. And their analysis shows favorable speed comparison to RSA, ElGamal, and DSA [582].

Patents

ESIGN is patented in the United States [1208], Canada, England, France, Germany, and Italy. Anyone who wishes to license the algorithm should contact Intellectual Property Department, NTT, 1–6 Uchisaiwai-cho, 1-chome, Chiyada-ku, 100 Japan.

20.7 CELLULAR AUTOMATA

A new and novel idea, studied by Papua Guam [665], is the use of cellular automata in public-key cryptosystems. This system is still far too new and has not been studied extensively, but a preliminary examination suggests that it may have a cryptographic weakness similar to one seen in other cases [562]. Still, this is a promising area of research. Cellular automata have the property that, even if they are invertible, it is impossible to calculate the predecessor of an arbitrary state by reversing the rule for finding the successor. This sounds a whole lot like a trapdoor one-way function.

20.8 OTHER PUBLIC-KEY ALGORITHMS

Many other public-key algorithms have been proposed and broken over the years. The Matsumoto-Imai algorithm [1021] was broken in [450]. The Cade algorithm

was first proposed in 1985, broken in 1986 [774], and then strengthened in the same year [286]. In addition to these attacks, there are general attacks for decomposing polynomials over finite fields [605]. Any algorithm that gets its security from the composition of polynomials over a finite field should be looked upon with skepticism, if not outright suspicion.

The Yagisawa algorithm combines exponentiation mod p with arithmetic mod $p - 1$ [1623]; it was broken in [256]. Another public-key algorithm, Tsujii-Kurosawa-Itoh-Fujioka-Matsumoto [1548] is insecure [948]. A third system, Luccio-Mazzone [993], is insecure [717]. A signature scheme based on birational permutations [1425] was broken the day after it was presented [381]. Tatsuaki Okamoto has several signature schemes: one is provably as secure as the Discrete Logarithm Problem, and another is provably as secure as the Discrete Logarithm Problem *and* the Factoring Problem [1206]. Similar schemes are in [709].

Gustavus Simmons suggested J-algebras as a basis for public-key algorithms [1455,145]. This idea was abandoned after efficient methods for factoring polynomials were invented [951]. Special polynomial semigroups have also been studied [1619,962], but so far nothing has come of it. Harald Niederreiter proposed a public-key algorithm based on shift-register sequences [1166]. Another is based on Lyndon words [1476] and another on propositional calculus [817]. And a recent public-key algorithm gets its security from the matrix cover problem [82]. Tatsuaki Okamoto and Kazuo Ohta compare a number of digital signature schemes in [1212].

Prospects for creating radically new and different public-key cryptography algorithms seem dim. In 1988 Whitfield Diffie noted that most public-key algorithms are based on one of three hard problems [492,494]:

1. Knapsack: Given a set of unique numbers, find a subset whose sum is N.
2. Discrete logarithm: If p is a prime and g and M are integers, find x such that $g^x \equiv M \pmod{p}$.
3. Factoring: If N is the product of two primes, either
   a) factor N,
   b) given integers M and C, find d such that $M^d \equiv C \pmod{N}$,
   c) given integers e and C, find M such that $M^e \equiv C \pmod{N}$, or
   d) given an integer x, decide whether there exists an integer y such that $x \equiv y^2 \pmod{N}$.

According to Diffie [492,494], the Discrete Logarithm Problem was suggested by J. Gill, the Factoring Problem by Knuth, and the knapsack problem by Diffie himself.

This narrowness in the mathematical foundations of public-key cryptography is worrisome. A breakthrough in either the problem of factoring or of calculating discrete logarithms could render whole classes of public-key algorithms insecure. Diffie points out [492,494] that this risk is mitigated by two factors:

1. The operations on which public key cryptography currently depends—multiplying, exponentiating, and factoring—are all fundamental arithmetic phenom-

ena. They have been the subject of intense mathematical scrutiny for centuries and the increased attention that has resulted from their use in public key cryptosystems has on balance enhanced rather than diminished our confidence.

2. Our ability to carry out large arithmetic computations has grown steadily and now permits us to implement our systems with numbers sufficient in size to be vulnerable only to a dramatic breakthrough in factoring, logarithms, or root extraction.

As we have seen, not all public-key algorithms based on these problems are secure. The strength of any public-key algorithm depends on more than the computational complexity of the problem upon which it is based; a hard problem does not necessarily imply a strong algorithm. Adi Shamir listed three reasons why this is so [1415]:

1. Complexity theory usually deals with single isolated instances of a problem. A cryptanalyst often has a large collection of statistically related problems to solve—several ciphertexts encrypted with the same key.

2. The computational complexity of a problem is typically measured by its worst-case or average-case behavior. To be useful as a cipher, the problem must be hard to solve in almost all cases.

3. An arbitrarily difficult problem cannot necessarily be transformed into a cryptosystem, and it must be possible to insert trapdoor information into the problem so that a shortcut solution is possible with this information and only with this information.

CHAPTER 21

Identification Schemes

21.1 FEIGE-FIAT-SHAMIR

Amos Fiat's and Adi Shamir's authentication and digital signature scheme is discussed in [566,567]. Uriel Feige, Fiat, and Shamir modified the algorithm to a zero-knowledge proof of identity [544,545]. This is the best-known zero-knowledge proof of identity.

On July 9, 1986 the three authors submitted a U.S. patent application [1427]. Because of its potential military applications, the application was reviewed by the military. Occasionally the Patent Office responds not with a patent, but with something called a secrecy order. On January 6, 1987, three days before the end of their six-month period, the Patent Office imposed that order at the request of the Army. They stated that "... the disclosure or publication of the subject matter ... would be detrimental to the national security. ..." The authors were ordered to notify all Americans to whom the research had been disclosed that unauthorized disclosure could lead to two years' imprisonment, a $10,000 fine, or both. Furthermore, the authors had to inform the Commissioner of Patents and Trademarks of all foreign citizens to whom the information had been disclosed.

This was ludicrous. All through the second half of 1986, the authors had presented the work at conferences throughout Israel, Europe, and the United States. The authors weren't even American citizens, and all the work had been done at the Weizmann Institute in Israel.

Word spread through the academic community and the press. Within two days the secrecy order was rescinded; Shamir and others believe that the NSA pulled strings to rescind the order, although they officially had no comment. Further details of this bizarre story are in [936].

Simplified Feige-Fiat-Shamir Identification Scheme

Before issuing any private keys, the arbitrator chooses a random modulus, n, which is the product of two large primes. In real life, n should be at least 512 bits

long and probably closer to 1024 bits. This n can be shared among a group of provers. (Choosing a Blum integer makes computation easier, but it is not required for security.)

To generate Peggy's public and private keys, a trusted arbitrator chooses a number, v, where v is a quadratic residue mod n. In other words, choose v such that $x^2 \equiv v \pmod{n}$ has a solution and $v^{-1} \bmod n$ exists. This v is Peggy's public key. Then calculate the smallest s for which $s \equiv \text{sqrt}(v^{-1}) \pmod{n}$. This is Peggy's private key.

The identification protocol can now proceed.

(1) Peggy picks a random r, where r is less then n. She then computes $x = r^2 \bmod n$, and sends x to Victor.

(2) Victor sends Peggy a random bit, b.

(3) If $b = 0$, then Peggy sends Victor r. If $b = 1$, then Peggy sends Victor $y = r * s \bmod n$.

(4) If $b = 0$, Victor verifies that $x = r^2 \bmod n$, proving that Peggy knows sqrt (x). If $b = 1$, Victor verifies that $x = y^2 * v \bmod n$, proving that Peggy knows sqrt (v^{-1}).

This is a single round—called an **accreditation**—of the protocol. Peggy and Victor repeat this protocol t times, until Victor is convinced that Peggy knows s. It's a cut-and-choose protocol. If Peggy doesn't know s, she can pick r such that she can fool Victor if he sends her a 0, or she can pick r such that she can fool Victor if he sends her a 1. She can't do both. The odds of her fooling Victor once are 50 percent. The odds of her fooling him t times are 1 in 2^t.

Another way for Victor to attack the protocol would be trying to impersonate Peggy. He could initiate the protocol with another verifier, Valerie. In step (1), instead of choosing a random r, he would just reuse an old r that he saw Peggy use. However, the odds of Valerie choosing the same value for b in step (2) that Victor did in the protocol with Peggy are 1 in 2. So, the odds of his fooling Valerie are 50 percent. The odds of his fooling her t times are 1 in 2^t.

For this to work, Peggy must not reuse an r, ever. If she did, and Victor sent Peggy the other random bit in step (2), then he would have both of Peggy's responses. Then, from even one of these, he can calculate s and it's all over for Peggy.

Feige-Fiat-Shamir Identification Scheme

In their papers [544,545], Feige, Fiat and Shamir show how parallel construction can increase the number of accreditations per round and reduce Peggy and Victor's interactions.

First generate n as in the previous example, the product of two large primes. To generate Peggy's public and private keys, first choose k different numbers: v_1, v_2, . . . , v_k, where each v_i is a quadratic residue mod n. In other words, choose v_i such that $x^2 = v_i \bmod n$ has a solution and $v_i^{-1} \bmod n$ exists. This string, v_1, v_2, . . . , v_k, is the public key. Then calculate the smallest s_i such that $s_i = \text{sqrt}(v_i^{-1}) \bmod n$. This string, s_1, s_2, . . . , s_k, is the private key.

And the protocol is:

(1) Peggy picks a random r, when r is less than n. She then computes $x = r^2 \bmod n$, and sends x to Victor.

(2) Victor sends Peggy a random binary string k-bits long: $b_1, b_2, \ldots, b_k$.

(3) Peggy computes $y = r * (s_1^{b_1} * s_2^{b_2} * \ldots * s_k^{b_k}) \bmod n$. (She multiplies together whichever values of s_i that correspond to $b_i = 1$. If Victor's first bit is a 1, then s_1 is part of the product; if Victor's first bit is a 0, then s_1 is not part of the product, and so on.) She sends y to Victor.

(4) Victor verifies that $x = y^2 * (v_1^{b_1} * v_2^{b_2} * \ldots * v_k^{b_k}) \bmod n$. (He multiplies together the values of s_i based on the random binary string. If his first bit is a 1, then s_1 is part of the product; if his first bit is a 0, then s_1 is not part of the product, and so on.)

Peggy and Victor repeat this protocol t times, until Victor is convinced that Peggy knows $s_1, s_2, \ldots, s_k$.

The chance that Peggy can fool Victor is 1 in 2^{kt}. The authors recommend a 1 in 2^{20} chance of a cheater fooling Victor and suggest that $k = 5$ and $t = 4$. If you are more paranoid, increase these numbers.

An Example

Let's look at this protocol in action with small numbers.

If $n = 35$ (the two primes are 5 and 7), then the possible quadratic residues are:

1: $x^2 \equiv 1 \pmod{35}$ has the solutions: $x = 1, 6, 29$, or 34.

4: $x^2 \equiv 4 \pmod{35}$ has the solutions: $x = 2, 12, 23$, or 33.

9: $x^2 \equiv 9 \pmod{35}$ has the solutions: $x = 3, 17, 18$, or 32.

11: $x^2 \equiv 11 \pmod{35}$ has the solutions: $x = 9, 16, 19$, or 26.

14: $x^2 \equiv 14 \pmod{35}$ has the solutions: $x = 7$ or 28.

15: $x^2 \equiv 15 \pmod{35}$ has the solutions: $x = 15$ or 20.

16: $x^2 \equiv 16 \pmod{35}$ has the solutions: $x = 4, 11, 24$, or 31.

21: $x^2 \equiv 21 \pmod{35}$ has the solutions: $x = 14$ or 21.

25: $x^2 \equiv 25 \pmod{35}$ has the solutions: $x = 5$ or 30.

29: $x^2 \equiv 29 \pmod{35}$ has the solutions: $x = 8, 13, 22$ or 27.

30: $x^2 \equiv 30 \pmod{35}$ has the solutions: $x = 10$ or 25.

The inverses (mod 35) and their square roots are:

v	v^{-1}	$s = \text{sqrt}(v^{-1})$
1	1	1
4	9	3
9	4	2

11	16	4
16	11	9
29	29	8

Note that 14, 15, 21, 25, and 30 do not have inverses mod 35, because they are not relatively prime to 35. This makes sense, because there should be $(5 - 1) * (7 - 1)/4$ quadratic residues mod 35 relatively prime to 35: That is $\gcd(x,35) = 1$ (see Section 11.3).

So, Peggy gets the public key consisting of $k = 4$ values: {4,11,16,29}. The corresponding private key is {3,4,9,8}. Here's one round of the protocol.

(1) Peggy chooses a random $r = 16$, computes $16^2 \bmod 35 = 11$, and sends it to Victor.

(2) Victor sends Peggy a random binary string {1,1,0,1}.

(3) Peggy computes $16 * ((3^1) * (4^1) * (9^0) * (8^1)) \bmod 35 = 31$ and sends it to Victor.

(4) Victor verifies that $31^2 * ((4^1) * (11^1) * (16^0) * (29^1)) \bmod 35 = 11$.

Peggy and Victor repeat the protocol t times, each time with a different random r, until Victor is satisfied.

With small values like these, there's no real security. But when n is 512 bits long or more, Victor cannot learn anything about Peggy's secret key except the fact that she knows it.

Enhancements

It is possible to embed identification information into the protocol. Assume that I is a binary string representing Peggy's identification: her name, address, social security number, hat size, preferred brand of soft drink, and other personal information. Use a one-way hash function $H(x)$ to compute $H(I,j)$, where j is a small number concatenated onto I. Find a set of js where $H(I,j)$ is a quadratic residue mod n. These $H(I,j)$s become $v_1, v_2, \ldots, v_k$ (the js need not be quadratic residues). Peggy's public key is now I and the list of js. She sends I and the list of js to Victor before step (1) of the protocol (or perhaps Victor downloads them from a public bulletin board someplace), and Victor generates $v_1, v_2, \ldots, v_k$ from $H(I,j)$.

Now, after Victor successfully completes the protocol with Peggy, he is assured that Trent, who knows the factorization of the modulus, has certified the association between I and Peggy by giving her the square roots of the v_i derived from I. (See Section 5.2 for background information.)

Feige, Fiat, and Shamir include the following implementation remarks [544,545]:

> For nonperfect hash functions, it may be advisable to randomize I by concatenating it with a long random string, R. This string is chosen by the arbitrator and is revealed to Victor along with I.
>
> In typical implementations, k should be between 1 and 18. Larger values of k can reduce the time and communication complexity by reducing the number of rounds.

The value n should be at least 512 bits long. (Of course, there has been considerable progress in factoring since then.)

If each user chooses his own n and publishes it in a public key file, they can dispense with the arbitrator. However, this RSA-like variant makes the scheme considerably less convenient.

Fiat-Shamir Signature Scheme

Turning this identification scheme into a signature scheme is basically a matter of turning Victor into a hash function. The primary benefit of the Fiat-Shamir digital signature scheme over RSA is speed: Fiat-Shamir requires only 1 percent to 4 percent of the modular multiplications of RSA. For this protocol, we'll bring back Alice and Bob.

The setup is the same as the identification scheme. Choose n to be the product of two large primes. Generate the public key, $v_1, v_2, \ldots, v_k$, and the private key, $s_1, s_2, \ldots, s_k$, such that $s_i = \text{sqrt}\,(v_i^{-1}) \bmod n$.

(1) Alice picks t random integers between 1 and n: $r_1, r_2, \ldots, r_t$, and computes $x_1, x_2, \ldots, x_t$ such that $x_i = r_i^2 \bmod n$.

(2) Alice hashes the concatenation of the message and the string of x_is to generate a bit stream: $H(m, x_1, x_2, \ldots, x_t)$. She uses the first $k * t$ bits of this string as values of b_{ij}, where i goes from 1 to t, and j goes from 1 to k.

(3) Alice computes $y_1, y_2, \ldots, y_t$, where

$$y_i = r_i * (s_1^{b_{i1}} * s_2^{b_{i2}} * \ldots * s_k^{b_{ik}}) \bmod n$$

(For each i, she multiplies together the values of the s_j based on the random $b_{i,j}$ values. If $b_{i,1}$ is a 1, then s_1 is multiplied; if $b_{i,1}$ is a 0, then s_1 is not multiplied.)

(4) Alice sends Bob m, all the bit values of $b_{i,j}$, and all the values of y_i. He already has Alice's public key: $v_1, v_2, \ldots, v_k$.

(5) Bob computes $z_1, z_2, \ldots, z_t$, where

$$z_i = y_i^2 * (v_1^{b_{i1}} * v_2^{b_{i2}} * \ldots * v_k^{b_{ik}}) \bmod n$$

(Again, Bob multiplies based on the $b_{i,j}$ values.) Also note that z_i should be equal to x_i.

(6) Bob verifies that the first $k * t$ bits of $H(m, z_1, z_2, \ldots, z_t)$ are the $b_{i,j}$ values that Alice sent him.

As with the identification scheme, the security of this signature scheme is proportional to $1/2^{kt}$. It also depends on the difficulty of factoring n. Fiat and Shamir pointed out that forging a signature is easier when the complexity of factoring n is considerably lower than 2^{kt}. And, because of birthday-type attacks (see Section 18.1), they recommend that $k * t$ be increased from 20 to at least 72. They suggest $k = 9$ and $t = 8$.

Improved Fiat-Shamir Signature Scheme

Silvio Micali and Adi Shamir improved the Fiat-Shamir protocol in [1088]. They chose $v_1, v_2, \ldots, v_k$ to be the first k prime numbers. So

$$v_1 = 2,\ v_2 = 3,\ v_3 = 5, \text{ and so on.}$$

This is the public key.

The private key, $s_1, s_2, \ldots, s_k$ is a random square root, determined by

$$s_i = \text{sqrt}\ (v_i^{-1}) \bmod n$$

In this version, every person must have a different n. The modification makes it easier to verify signatures. The time required to generate signatures, and the security of those signatures, is unaffected.

Other Enhancements

There is also an N-party identification scheme, based on the Fiat-Shamir algorithm [264]. Two other improvements to the Fiat-Shamir scheme are proposed in [1218]. Another variant is [1368].

Ohta-Okamoto Identification Scheme

This protocol is a modification of the Feige-Fiat-Shamir identification scheme and gets its security from the difficulty of factoring [1198,1199]. The same authors also wrote a multisignature scheme (see Section 23.1), by which a number of different people can sequentially sign a message [1200]. This scheme has been proposed for smart-card implementation [850].

Patents

Fiat-Shamir is patented [1427]. Anyone interested in licensing the algorithm should contact Yeda Research and Development, The Weizmann Institute of Science, Rehovot 76100, Israel.

21.2 GUILLOU-QUISQUATER

Feige-Fiat-Shamir was the first practical identity-based protocol. It minimized computation by increasing the number of iterations and accreditations per iteration. For some implementations, like smart cards, this is less than ideal. Exchanges with the outside world are time-consuming, and the storage required for each accreditation can strain the limited resources of the card.

Louis Guillou and Jean-Jacques Quisquater developed a zero-knowledge identification algorithm more suited to applications like these [670,1280]. The exchanges between Peggy and Victor and the parallel accreditations in each exchange are both kept to an absolute minimum: There is only one exchange of one accreditation for each proof. For the same level of security, the computation required by Guillou-Quisquater is greater than by Feige-Fiat-Shamir by a factor of three. And

like Feige-Fiat-Shamir, this identification algorithm can be converted to a digital signature algorithm.

Guillou-Quisquater Identification Scheme

Peggy is a smart card who wants to prove her identity to Victor. Peggy's identity consists of a set of credentials: a data string consisting of the card's name, validity period, a bank account number, and whatever else the application warrants. This bit string is called J. (Actually, the credentials can be a longer string and hashed to a J value. This complexity does not modify the protocol in any way.) This is analogous to the public key. Other public information, shared by all "Peggys" who could use this application, is an exponent v and a modulus n, where n is the product of two secret primes. The private key is B, calculated such that $JB^v \equiv 1 \pmod{n}$.

Peggy sends Victor her credentials, J. Now, she wants to prove to Victor that those credentials are hers. To do this, she has to convince Victor that she knows B. Here's the protocol:

(1) Peggy picks a random integer r, such that r is between 1 and $n - 1$. She computes $T = r^v \bmod n$ and sends it to Victor.

(2) Victor picks a random integer, d, such that d is between zero and $v - 1$. He sends d to Peggy.

(3) Peggy computes $D = rB^d \bmod n$, and sends it to Victor.

(4) Victor computes $T' = D^v J^d \bmod n$. If $T \equiv T' \pmod{n}$, then the authentication succeeds.

The math isn't that complex:

$$T' = D^v J^d = (rB^d)^v J^d = r^v B^{dv} J^d = r^v (JB^v)^d = r^v \equiv T \pmod{n}$$

since B was constructed to satisfy

$$JB^v \equiv 1 \pmod{n}$$

Guillou-Quisquater Signature Scheme

This identification can be converted to a signature scheme, also suited for smartcard implementation [671,672].

The public and private key setup is the same as before. Here's the protocol:

(1) Alice picks a random integer r, such that r is between 1 and $n - 1$. She computes $T = r^v \bmod n$.

(2) Alice computes $d = H(M,T)$, where M is the message being signed and $H(x)$ is a one-way hash function. The d produced by the hash function must be between 0 and $v - 1$ [1280]. If the output of the hash function is not within this range, it must be reduced modulo v.

(3) Alice computes $D = rB^d \bmod n$. The signature consists of the message, M, the two calculated values, d and D, and her credentials, J. She sends this signature to Bob.

(4) Bob computes $T' = D^v J^d \bmod n$. He then computes $d' = H(M,T')$. If $d = d'$, then Alice must know B and the signature is valid.

Multiple Signatures

What if many people want to sign the same document? The easy solution has each of them signing separately, but this signature scheme can do better than that. Here Alice and Bob sign the same document and Carol verifies the signatures, but any number of people can be involved in the signature process. As before, Alice and Bob have their own unique J and B values: (J_A, B_A) and (J_B, B_B). The values n and v are common to the system.

(1) Alice picks a random integer, r_A, such that r_A is between 1 and $n - 1$. She computes $T_A = r_A^v \bmod n$ and sends T_A to Bob.

(2) Bob picks a random integer, r_B, such that r_B is between 1 and $n - 1$. He computes $T_B = r_B^v \bmod n$ and sends T_B to Alice.

(3) Alice and Bob each compute $T = (T_A T_B) \bmod n$.

(4) Alice and Bob each compute $d = H(M,T)$, where M is the message being signed and $H(x)$ is a one-way hash function. The d produced by the hash function must be between 0 and $v - 1$ [1280]. If the output of the hash function is not within this range, it must be reduced modulo v.

(5) Alice computes $D_A = r_A B_A^d \bmod n$ and sends D_A to Bob.

(6) Bob computes $D_B = r_B B_B^d \bmod n$ and sends D_B to Alice.

(7) Alice and Bob each compute $D = D_A D_B \bmod n$. The signature consists of the message, M, the two calculated values, d and D, and both of their credentials: J_A and J_B.

(8) Carol computes $J = J_A J_B \bmod n$.

(9) Carol computes $T' = D^v J^d \bmod n$. She then computes $d' = H(M,T')$. If $d \equiv d'$, then the multiple signature is valid.

This protocol can be extended to any number of people. For multiple people to sign, they all multiply their individual T_i values together in step (3), and their individual D_i values together in step (7). To verify a multiple signature, multiply all the signers J_i values together in step (8). Either all the signatures are valid or there is at least one invalid signature.

21.3 SCHNORR

Claus Schnorr's authentication and signature scheme [1396,1397] gets its security from the difficulty of calculating discrete logarithms. To generate a key pair, first

choose two primes, p and q, such that q is a prime factor of $p - 1$. Then, choose an a not equal to 1, such that $a^q \equiv 1 \pmod{p}$. All these numbers can be common to a group of users and can be freely published.

To generate a particular public-key/private-key key pair, choose a random number less than q. This is the private key, s. Then calculate $v = a^{-s} \bmod p$. This is the public key.

Authentication Protocol

(1) Peggy picks a random number, r, less than q, and computes $x = a^r \bmod p$. This is the preprocessing stage and can be done long before Victor is present.

(2) Peggy sends x to Victor.

(3) Victor sends Peggy a random number, e, between 0 and $2^t - 1$. (I'll discuss t in a moment.)

(4) Peggy computes $y = (r + se) \bmod q$ and sends y to Victor.

(5) Victor verifies that $x = a^y v^e \bmod p$.

The security is based on the parameter t. The difficulty of breaking the algorithm is about 2^t. Schnorr recommended that p be about 512 bits, q be about 140 bits, and t be 72.

Digital Signature Protocol

Schnorr can also be used as a digital signature protocol on a message, M. The public-key/private-key key pair is the same, but we're now adding a one-way hash function, $H(M)$.

(1) Alice picks a random number, r, less than q, and computes $x = a^r \bmod p$. This computation is the preprocessing stage.

(2) Alice concatenates M and x, and hashes the result:

$$e = H(M,x)$$

(3) Alice computes $y = (r + se) \bmod q$. The signature is e and y; she sends these to Bob.

(4) Bob computes $x' = a^y v^e \bmod p$. He then confirms that the concatenation of M and x' hashes to e.

$$e = H(M,x')$$

If it does, he accepts the signature as valid.

In his paper, Schnorr cites these novel features of his algorithm:

> Most of the computation for signature generation can be completed in a preprocessing stage, independent of the message being signed. Hence, it can be done dur-

ing idle time and not affect the signature speed. An attack against this preprocessing stage is discussed in [475], but I don't think it's practical.

For the same level of security, the length of signatures is less for Schnorr than for RSA. For example, with a 140-bit q, signatures are only 212-bits long, less than half the length of RSA signatures. Schnorr's signatures are also much shorter than ElGamal signatures.

Of course, practical considerations may make even fewer bits suitable for a given scheme: For example, an identification scheme where the cheater must perform an on-line attack in only a few seconds, versus a signature scheme where the cheater can calculate for years off-line to come up with a forgery.

A modification of this algorithm, by Ernie Brickell and Kevin McCurley, enhances its security [265].

Patents

Schnorr is patented in the United States [1398] and in many other countries. In 1993, PKP acquired the worldwide rights to the patent (see Section 25.5). The U.S. patent expires on February 19, 2008.

21.4 CONVERTING IDENTIFICATION SCHEMES TO SIGNATURE SCHEMES

There is a standard method of converting an identification scheme into a signature scheme: Replace Victor with a one-way hash function. The message is not hashed before it is signed; instead the hashing is incorporated into the signing algorithm. In principle, this can be done with any identification scheme.

CHAPTER 22

Key-Exchange Algorithms

22.1 DIFFIE-HELLMAN

Diffie-Hellman was the first public-key algorithm ever invented, way back in 1976 [496]. It gets its security from the difficulty of calculating discrete logarithms in a finite field, as compared with the ease of calculating exponentiation in the same field. Diffie-Hellman can be used for key distribution—Alice and Bob can use this algorithm to generate a secret key—but it cannot be used to encrypt and decrypt messages.

The math is simple. First, Alice and Bob agree on a large prime, n and g, such that g is primitive mod n. These two integers don't have to be secret; Alice and Bob can agree to them over some insecure channel. They can even be common among a group of users. It doesn't matter.

Then, the protocol goes as follows:

(1) Alice chooses a random large integer x and sends Bob

$$X = g^x \bmod n$$

(2) Bob chooses a random large integer y and sends Alice

$$Y = g^y \bmod n$$

(3) Alice computes

$$k = Y^x \bmod n$$

(4) Bob computes

$$k' = X^y \bmod n$$

Both k and k' are equal to $g^{xy} \bmod n$. No one listening on the channel can compute that value; they only know n, g, X, and Y. Unless they can compute the discrete log-

arithm and recover x or y, they do not solve the problem. So, k is the secret key that both Alice and Bob computed independently.

The choice of g and n can have a substantial impact on the security of this system. The number $(n - 1)/2$ should also be a prime [1253]. And most important, n should be large: The security of the system is based on the difficulty of factoring numbers the same size as n. You can choose any g, such that g is primitive mod n; there's no reason not to choose the smallest g you can—generally a one-digit number. (And actually, g does not have to be primitive; it just has to generate a large subgroup of the multiplicitive group mod n.)

Diffie-Hellman with Three or More Parties

The Diffie-Hellman key-exchange protocol can easily be extended to work with three or more people. In this example, Alice, Bob, and Carol together generate a secret key.

(1) Alice chooses a random large integer x and computes

$$X = g^x \bmod n$$

(2) Bob chooses a random large integer y and sends Carol

$$Y = g^y \bmod n$$

(3) Carol chooses a random large integer z and sends Alice

$$Z = g^z \bmod n$$

(4) Alice sends Bob

$$Z' = Z^x \bmod n$$

(5) Bob sends Carol

$$X' = X^y \bmod n$$

(6) Carol sends Alice

$$Y' = Y^z \bmod n$$

(7) Alice computes

$$k = Y'^x \bmod n$$

(8) Bob computes

$$k = Z'^y \bmod n$$

(9) Carol computes

$$k = X'^y \bmod n$$

The secret key, k, is equal to $g^{xyz} \bmod n$, and no one else listening in on the communications can compute that value. The protocol can be easily extended to four or more people; just add more people and more rounds of computation.

Extended Diffie-Hellman

Diffie-Hellman also works in commutitive rings [1253]. Z. Shmuley and Kevin McCurley studied a variant of the algorithm where the modulus is a composite number [1442,1038]. V. S. Miller and Neal Koblitz extended this algorithm to elliptic curves [1095,867]. Taher ElGamal used the basic idea to develop an encryption and digital signature algorithm (see Section 19.6).

This algorithm also works in the Galois field $GF(2^k)$ [1442,1038]. Some implementations take this approach [884,1631,1632], because the computation is much quicker. Similarly, cryptanalytic computation is equally fast, so it is important to carefully choose a field large enough to ensure security.

Hughes

This variant of Diffie-Hellman allows Alice to generate a key and send it to Bob [745].

(1) Alice chooses a random large integer x and generates

$$k = g^x \bmod n$$

(2) Bob chooses a random large integer y and sends Alice

$$Y = g^y \bmod n$$

(3) Alice sends Bob

$$X = Y^x \bmod n$$

(4) Bob computes

$$z = y^{-1}$$
$$k' = X^z \bmod n$$

If everything goes correctly, $k = k'$.

The advantage of this protocol over Diffie-Hellman is that k can be computed before any interaction, and Alice can encrypt a message using k prior to contacting Bob. She can send it to a variety of people and interact with them to exchange the key individually later.

Key Exchange Without Exchanging Keys

If you have a community of users, each could publish a public key, $X = g^x \bmod n$, in a common database. If Alice wants to communicate with Bob, she just has to retrieve Bob's public key and generate their shared secret key. She could then encrypt a message with that key and send it to Bob. Bob would retrieve Alice's public key to generate the shared secret key.

Each pair of users would have a unique secret key, and no prior communication between users is required. The public keys have to be certified to prevent spoofing attacks and should be changed regularly, but otherwise this is a pretty clever idea.

Patents

The Diffie-Hellman key-exchange algorithm is patented in the United States [718] and Canada [719]. A group called Public Key Partners (PKP) licenses the patent, along with other public-key cryptography patents (see Section 25.5). The U.S. patent will expire on April 29, 1997.

22.2 STATION-TO-STATION PROTOCOL

Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack. One way to prevent this problem is to have Alice and Bob sign their messages to each other [500].

This protocol assumes that Alice has a certificate with Bob's public key and that Bob has a certificate with Alice's public key. These certificates have been signed by some trusted authority outside this protocol. Here's how Alice and Bob generate a secret key, k.

(1) Alice generates a random number, x, and sends it to Bob.

(2) Bob generates a random number, y. Using the Diffie-Hellman protocol he computes their shared key based on x and y: k. He signs x and y, and encrypts the signature using k. He then sends that, along with y, to Alice.

$$y, E_k[S_B[x,y]]$$

(3) Alice also computes k. She decrypts the rest of Bob's message and verifies his signature. Then she sends Bob a signed message consisting of x and y, encrypted in their shared key.

$$E_k[S_A[x,y]]$$

(4) Bob decrypts the message and verifies Alice's signature.

22.3 SHAMIR'S THREE-PASS PROTOCOL

This protocol, invented by Adi Shamir but never published, enables Alice and Bob to communicate securely without any advance exchange of either secret keys or public keys [1008].

This assumes the existence of a symmetric cipher that is commutative, that is:

$$E_A[E_B[P]] = E_B[E_A[P]]$$

Alice's secret key is A; Bob's secret key is B. Alice wants to send a message, M, to Bob. Here's the protocol.

(1) Alice encrypts M with her key and sends Bob

$$C_1 = E_A[M]$$

(2) Bob encrypts C_1 with his key and sends Alice

$$C_2 = E_B[E_A[M]]$$

(3) Alice decrypts C_2 with her key and sends Bob

$$C_3 = D_A[E_B[E_A[M]]] = D_A[E_A[E_B[M]]] = E_B[M]$$

(4) Bob decrypts C_3 with his key to recover M.

One-time pads are commutative and have perfect secrecy, but they will not work with this protocol. With a one-time pad, the three ciphertext messages would be:

$$C_1 = P \oplus A$$
$$C_2 = P \oplus A \oplus B$$
$$C_3 = P \oplus B$$

Eve, who can record the three messages as they pass between Alice and Bob, simply XORs them together to retrieve the message:

$$C_1 \oplus C_2 \oplus C_3 = (P \oplus A) \oplus (M \oplus A \oplus B) \oplus (M \oplus B) = M$$

This clearly won't work.

Shamir (and independently, Jim Omura) described an encryption algorithm that will work with this protocol, one similar to RSA. Let p be a large prime for which $p - 1$ has a large prime factor. Choose an encryption key, e, such that e is relatively prime to $p - 1$. Calculate d such that $de \equiv 1 \pmod{p-1}$.

To encrypt a message, calculate

$$C = M^e \bmod p$$

To decrypt a message, calculate

$$M = C^d \bmod p$$

There seems to be no way for Eve to recover M without solving the discrete logarithm problem, but this has never been proved.

Like Diffie-Hellman, this protocol allows Alice to initiate secure communication with Bob without knowing any of his keys. For Alice to use a public-key algorithm, she has to know his public key. With Shamir's three-pass protocol, she just sends him a ciphertext message. The same thing with a public-key algorithm looks like:

(1) Alice asks Bob (or a KDC) for his public key.
(2) Bob (or the KDC) sends Alice his public key.
(3) Alice encrypts M with Bob's public key and sends it to Bob.

Shamir's three-pass protocol will fall to a man-in-the-middle attack.

22.4 COMSET

COMSET (COMmunications SETup) is a mutual identification and key exchange protocol developed for the RIPE project [1305] (see Section 25.7). Using public-key

cryptography, it allows Alice and Bob to identify themselves to each other and also to exchange a secret key.

The mathematical principle behind COMSET is Rabin's scheme [1283] (see Section 19.5). The scheme itself was originally proposed in [224]. See [1305] for details.

22.5 ENCRYPTED KEY EXCHANGE

The Encrypted Key Exchange (EKE) protocol was designed by Steve Bellovin and Michael Merritt [109]. It provides security and authentication on computer networks, using both symmetric and public-key cryptography in a novel way: A shared secret key is used to encrypt a randomly generated public key.

The Basic EKE Protocol

Alice and Bob (two users, a user and the host, or whoever) share a common password, P. Using this protocol, they can authenticate each other and generate a common session key, K.

(1) Alice generates a random public-key/private-key key pair. She encrypts the public key, K', using a symmetric algorithm and P as the key: $E_P(K')$. She sends Bob

$$A, E_P(K')$$

(2) Bob knows P. He decrypts the message to obtain K'. Then, he generates a random session key, K, and encrypts it with the public key he received from Alice and P as the key. He sends Alice

$$E_P(E_{K'}(K))$$

(3) Alice decrypts the message to obtain K. She generates a random string, R_A, encrypts it with K, and sends Bob

$$E_K(R_A)$$

(4) Bob decrypts the message to obtain R_A. He generates another random string, R_B, encrypts both strings with K, and sends Alice the result.

$$E_K(R_A, R_B)$$

(5) Alice decrypts the message to obtain R_A and R_B. Assuming the R_A she received from Bob is the same as the one she sent to Bob in step (3), she encrypts R_B with K and sends it to Bob.

$$E_K(R_B)$$

(6) Bob decrypts the message to obtain R_B. Assuming the R_B he received from Alice is the same one he sent to Alice in step (4), the protocol is complete. Both parties now communicate using K as the session key.

At step (3), both Alice and Bob know K' and K. K is the session key and can be used to encrypt all other messages between Alice and Bob. Eve, sitting between Alice and

Bob, only knows $E_P[K']$, $E_P[E_{K'}[K]]$, and some messages encrypted with K. In other protocols, Eve could make guesses at P (people choose bad passwords all the time, and if Eve is clever she can make some good guesses) and then test her guesses. In this protocol, Eve cannot test her guess without cracking the public-key algorithm as well. And if both K' and K are chosen randomly, this can be an insurmountable problem.

The challenge-response portion of the protocol, steps (3) through (6), provides validation. Steps (3) through (5) prove to Alice that Bob knows K; steps (4) through (6) prove to Bob that Alice knows K. The Kerberos protocol timestamp exchange accomplishes the same thing.

EKE can be implemented with a variety of public-key algorithms: RSA, ElGamal, Diffie-Hellman. There are security problems with implementing EKE with a knapsack algorithm (aside from the inherent insecurity of knapsack algorithms): The normal distribution of the ciphertext messages negates the benefits of EKE.

Implementing EKE with RSA

The RSA algorithm seems perfect for this application, but there are some subtle problems. The authors recommend encrypting only the encryption exponent in step (1) and sending the modulus in the clear. An explanation of the reasoning behind this recommendation, as well as other subtleties involved in using RSA, is in [109].

Implementing EKE with ElGamal

Implementing EKE with the ElGamal algorithm is straightforward, and there is even a simplification of the basic protocol. Using the notation from Section 19.6, g and p are parts of the public key and are common to all users. The private key is a random number r. The public key is $g^r \bmod p$. The message Alice sends to Bob in step (1) becomes

$$\text{Alice}, g^r \bmod p$$

Note that this public key does not have to be encrypted with P. This is not true in general, but it is true for the ElGamal algorithm. Details are in [109].

Bob chooses a random number, R (for the ElGamal algorithm and independent of any random numbers chosen for EKE), and the message he sends to Alice in step (2) becomes

$$E_P[g^R \bmod p, Kg^{Rr} \bmod p]$$

Refer back to Section 19.6 for restrictions on choosing the variables for ElGamal.

Implementing EKE with Diffie-Hellman

With the Diffie-Hellman protocol, K is generated automatically. The final protocol is even simpler. A value for g and n is set for all users on the network.

(1) Alice picks a random number, r_A, and sends Bob

$$A, g^{r_A} \bmod n$$

With Diffie-Hellman, Alice does not have to encrypt her first message with P.

(2) Bob picks a random number, r_B, and calculates

$$K = g^{r_A * r_B} \bmod n$$

He generates a random string R_B, then calculates and sends Alice:

$$E_P\{g^{r_B} \bmod n\},\ E_K\{R_B\}$$

(3) Alice decrypts the first half of Bob's message to obtain $g^{r_B} \bmod n$. Then she calculates K and uses K to decrypt R_B. She generates another random string, R_A, encrypts both strings with K, and sends Bob the result.

$$E_K\{R_A, R_B\}$$

(4) Bob decrypts the message to obtain R_A and R_B. Assuming the R_B he received from Alice is the same as the one he sent to Alice in step (2), he encrypts R_A with K and sends it to Alice.

$$E_K\{R_A\}$$

(5) Alice decrypts the message to maintain R_A. Assuming the R_A she received from Bob is the same as the one she sent to Bob in step (3), the protocol is complete. Both parties now communicate using K as the session key.

Strengthening EKE

Bellovin and Merritt suggest an enhancement of the challenge-and-response portion of the protocol—to prevent a possible attack if a cryptanalyst recovers an old K value.

Look at the basic EKE protocol. In step (3), Alice generates another random number, S_A, and sends Bob

$$E_K\{R_A, S_A\}$$

In step (4), Bob generates another random number, S_B, and sends Alice

$$E_K\{R_A, R_B, S_B\}$$

Alice and Bob now can both calculate the true session key, $S_A \oplus S_B$. This key is used for all future messages between Alice and Bob; K is just used as a key-exchange key.

Look at the levels of protection EKE provides. A recovered value of S gives Eve no information about P, because P is never used to encrypt anything that leads directly to S. A cryptanalytic attack on K is also not feasible; K is used only to encrypt random data, and S is never encrypted alone.

Augmented EKE

The EKE protocol suffers from one serious disadvantage: It requires that both parties possess the P. Most password-based authentication systems store a one-way hash of the user's password, not the password itself (see Section 3.2). The Augmented EKE (A-EKE) protocol uses a one-way hash of the user's password as the

superencryption key in the Diffie-Hellman variant of EKE. The user then sends an extra message based on the original password; this message authenticates the newly chosen session key.

Here's how it works. As usual, Alice and Bob want to authenticate each other and generate a common key. They agree on some digital signature scheme where any number can serve as the private key, and where the public key is derived from the private key, rather than being generated along with it. The ElGamal and DSA algorithms work well for this. Alice's password P (or perhaps some simple hash of it) will serve as the private key and as P'.

(1) Alice picks her random exponent R_a and transmits

$$E_{P'}(g^{r_A} \bmod n)$$

(2) Bob, who knows only P' and cannot derive P from it, chooses R_b and sends

$$E_P(g^{r_A} \bmod n)$$

(3) Both Alice and Bob calculate the shared session key $K = g^{r_A \cdot r_B} \bmod n$. Finally, Alice proves that she knows P itself, and not just P', by sending

$$E_K(S_P(K))$$

Bob, who knows both K and P', can decrypt and validate the signature. Only Alice could have sent this message, since only she knows P; an intruder who obtains a copy of Bob's password file can try guessing at P, but cannot otherwise sign the session key.

The A-EKE scheme does not work with the public-key variant of EKE, since in it one party chooses the session key and imposes it on the other. This permits a man-in-the-middle attack by an attacker who has captured P'.

Applications of EKE

Bellovin and Merritt suggest using this protocol for secure public telephones [109]:

> Let us assume that encrypting public telephones are deployed. If someone wishes to use one of these phones, some sort of keying information must be provided. Conventional solutions . . . require that the caller possess a physical key. This is undesirable in many situations. EKE permits use of a short, keypad-entered password, but uses a much longer session key for the call.
>
> EKE would also be useful with cellular phones. Fraud has been a problem in the cellular industry; EKE can defend against it (and ensure the privacy of the call) by rendering a phone useless if a PIN has not been entered. Since the PIN is not stored within the phone, it is not possible to retrieve one from a stolen unit.

EKE's primary strength is that both symmetric and public-key cryptography work together in a manner that strengthens them both:

> From a general perspective, EKE functions as a *privacy amplifier*. That is, it can be used to strengthen comparatively weak symmetric and asymmetric systems

when used together. Consider, for example, the key size needed to maintain security when using exponential key exchange. As LaMacchia and Odlyzko have shown [934], even modulus sizes once believed to be safe (to wit, 192 bits) are vulnerable to an attack requiring only a few minutes of computer time. But their attack is not feasible if one must first guess a password before applying it.

Conversely, the difficulty of cracking exponential key exchange can be used to frustrate attempts at password-guessing. Password-guessing attacks are feasible because of how rapidly each guess may be verified. If performing such verification requires solving an exponential key exchange, the total time, if not the conceptual difficulty, increases dramatically.

EKE is patented [111].

22.6 FORTIFIED KEY NEGOTIATION

This scheme also protects key-negotiation schemes from poorly chosen passwords and man-in-the-middle attacks [47,983]. It uses a hash function of two variables that has a very special property: It has many collisions on the first variable while having effectively no collisions on the second variable.

$$H'(x, y) = H(H(k, x) \bmod 2^m, x),$$
where $H(k, x)$ is an ordinary hash function on k and x

Here's the protocol. Alice and Bob share a secret password, P, and have just exchanged a secret key, K, using Diffie-Hellman key exchange. They use P to check that their two session keys are the same (and that Eve is not attempting a man-in-the-middle attack), without giving P away to Eve.

(1) Alice sends Bob

$$H'(P, K)$$

(2) Bob computes $H'(P, K)$ and compares his result with what he received from Alice. If they match he sends Alice

$$H'(H(P, K))$$

(3) Alice computes $H'(H(P, K))$ and compares her result with what she received from Bob.

If Eve is trying a man-in-the-middle attack, she shares one key, K_1, with Alice, and another key, K_2, with Bob. To fool Bob in step (2), she has to figure out the shared password and then send Bob $H * (P, K_2)$. With a normal hash function she can try common passwords until she guesses the correct one, and then successfully infiltrate the protocol. But with this hash function, many passwords are likely to produce the same value when hashed with K_1. So when she finds a match, she will probably have the wrong password, and hence Bob will not be fooled.

22.7 Conference Key Distribution and Secret Broadcasting

Alice wants to broadcast a message, M, from a single transmitter. However, she doesn't want it to be intelligible by every listener. In fact, she only wants a select subset of listeners to be able to recover M. Everyone else should get nonsense.

Alice can share a different key (secret or public) with each listener. She encrypts the message in some random key, K. Then she encrypts a copy of K with each of the keys of her intended recipients. Finally, she broadcasts the encrypted message and then all of the encrypted Ks. Bob, who is listening, either tries to decrypt all the Ks with his secret key, looking for one that is correct, or, if Alice doesn't mind everyone knowing who her message is for, he looks for his name followed by an encrypted key. Multiple-key cryptography, previously discussed, also works.

Another method is suggested in [352]. First, each listener shares a secret key with Alice, one that is larger than any possible encrypted message. All of those keys should be pairwise prime. She encrypts the message in a random key, K. Then, she computes a single integer, R, such that R modulo a secret key is congruent to K when that secret key is supposed to decrypt the message, and R modulo a secret key is otherwise congruent to zero.

For example, if Alice wants the secret to be received by Bob, Carol, and Ellen, but not by Dave and Frank, she encrypts the message with K and then computes R such that

$$R \equiv K \pmod{K_B}$$
$$R \equiv K \pmod{K_C}$$
$$R \equiv 0 \pmod{K_D}$$
$$R \equiv K \pmod{K_E}$$
$$R \equiv 0 \pmod{K_F}$$

This is a straightforward algebra problem, one that Alice can solve easily. When listeners receive the broadcast, they compute the received key modulo their secret key. If they were intended to receive the message, they recover the key. Otherwise, they recover nothing.

Yet a third way, using a threshold scheme (see Section 3.7), is suggested in [141]. Like the others, every potential receiver gets a secret key. This key is a shadow in a yet-uncreated threshold scheme. Alice saves some secret keys for herself, adding some randomness to the system. Let's say there are k people out there.

Then, to broadcast M, Alice encrypts M with key K and does the following.

(1) Alice chooses a random number, j. This number serves to hide the number of recipients of the message. It doesn't have to be very large; it can be as small as 0.

(2) Alice creates a $(k + j + 1, 2k + j + 1)$ threshold scheme, with:

K as the secret.

The secret keys of the intended recipients as shadows.

The secret keys of nonrecipients not as shadows.

j randomly chosen shadows, not the same as any of the secret keys.

(3) Alice broadcasts $k + j$ randomly chosen shadows, none of which is any of the shadows listed in step (2).

(4) All listeners who receive the broadcast add their shadow to the $k + j$ shadows they received. If adding their shadow allows them to calculate the secret, then they have recovered the key. If it does not, then they haven't.

Another approach can be found in [885,886,1194]. For yet another approach, see [1000].

Conference Key Distribution

This protocol allows a group of n users to agree on a secret key using only insecure channels. The group shares two large primes, p and q, and a generator g the same size as q.

(1) User i, where i goes from 1 to n, chooses a random r_i less than q, and broadcasts

$$z_i = g^{r_i} \bmod p$$

(2) Every user verifies that $z_i^q \equiv 1 \pmod{p}$, for all i from 1 to n.

(3) User i broadcasts

$$x_i = (z_{i+1}/z_{i-1})^{r_i} \bmod p$$

(4) User i computes

$$K = (z_{i-1})^{nr_i} * x_i^{n-1} * x_{i+1}^{n-2} * \ldots * x_{i-2} \bmod p$$

All index computations in the above protocol—$i - 1$, $i - 2$, and $i + 1$—should be computed mod n. At the end of the protocol, all honest users have the same K. No one else gets anything. However, this protocol falls to a man-in-the-middle attack. Another protocol, not quite as pretty, is in [757].

Tatebayashi-Matsuzaki-Newman

This key distribution protocol is suitable for networks [1521]. Alice wants to generate a session key with Bob using Trent, the KDC. All parties know Trent's public key, n. Trent knows the two large primes that n factors to, and hence can easily take

cube roots modulo n. A lot of the details are left out of the following protocol, but you get the idea.

(1) Alice chooses a random number, r_A, and sends Trent

$$r_A^3 \bmod n$$

(2) Trent tells Bob that someone wants to exchange a key with him.

(3) Bob chooses a random number, r_B, and sends Trent

$$r_B^3 \bmod n$$

(4) Trent uses his private key to recover r_A and r_B. He sends Alice

$$r_A \oplus r_B$$

(5) Alice calculates

$$(r_A \oplus r_B) \oplus r_A = r_B$$

She uses this r_B to communicate securely with Bob.

This protocol looks good, but it has a horrible flaw. Carol can listen in on step (3) and use that information, with the help of an unsuspecting Trent and another malicious user (Dave), to recover r_B [1472].

(1) Carol chooses a random number, r_C, and sends Trent

$$r_B^3 r_C^3 \bmod n$$

(2) Trent tells Dave that someone wants to exchange a key with him.

(3) Dave chooses a random number, r_D, and sends Trent

$$r_D^3 \bmod n$$

(4) Trent uses his private key to recover r_C and r_D. He sends Carol

$$(r_B r_C) \bmod n \oplus r_D$$

(5) Dave sends r_D to Carol.

(6) Carol uses r_C and r_D to recover r_B. She uses r_B to eavesdrop on Alice and Bob.

This is not good.

CHAPTER 23

Special Algorithms for Protocols

23.1 MULTIPLE-KEY PUBLIC-KEY CRYPTOGRAPHY

This is a generalization of RSA (see Section 19.3) [217,212]. The modulus, n, is the product of two primes, p and q. However, instead of choosing e and d such that $ed \equiv 1 \bmod (p-1)(q-1)$, choose t keys, K_i, such that

$$K_1 * K_2 * \ldots * K_t \equiv 1 \bmod (p-1)(q-1)$$

Since

$$M^{K_1 * K_2 * \ldots * K_t} = M$$

this is a multiple-key scheme as described in Section 3.5.

If, for example, there are five keys, a message encrypted with K_3 and K_5 can be decrypted with K_1, K_2, and K_4:

$$C = M^{K_3 * K_5} \bmod n$$
$$M = C^{K_1 * K_2 * K_4} \bmod n$$

One use for this is multisignatures. Imagine a situation where both Alice and Bob have to sign a document for it to be valid. Use three keys: K_1, K_2, and K_3. The first two are issued one each to Alice and Bob, and the third is made public.

(1) First Alice signs M and sends it to Bob.

$$M' = M^{K_1} \bmod n$$

(2) Bob can recover M from M'.

$$M = M'^{K_2 * K_3} \bmod n$$

(3) He can also add his signature.

$$M'' = M'^{K_2} \bmod n$$

(4) Anyone can verify the signature with K_3, the public key.

$$M = M''^{K_3} \bmod n$$

Note that a trusted party is needed to set this system up and distribute the keys to Alice and Bob. Another scheme with the same problem is [484]. Yet a third scheme is [695,830,700], but the effort in verification is proportional to the number of signers. Newer schemes [220,1200] based on zero-knowledge identification schemes solve both shortcomings of the previous systems.

23.2 SECRET-SHARING ALGORITHMS

Back in Section 3.7 I discussed the idea behind secret-sharing schemes. The four different algorithms that follow are all particular cases of a general theoretical framework [883].

LaGrange Interpolating Polynomial Scheme

Adi Shamir uses polynomial equations in a finite field to construct a threshold scheme [1414]. Choose a prime, p, which is both larger than the number of possible shadows and larger than the largest possible secret. To share a secret, generate an arbitrary polynomial of degree $m - 1$. For example, if you want to create a $(3,n)$-threshold scheme (three shadows are necessary to reconstruct M), generate a quadratic polynomial

$$(ax^2 + bx + M) \bmod p$$

where p is a random prime larger than any of the coefficients. The coefficients a and b are chosen randomly; they are kept secret and discarded after the shadows are handed out. M is the message. The prime must be made public.

The shadows are obtained by evaluating the polynomial at n different points:

$$k_i = F(x_i)$$

In other words, the first shadow could be the polynomial evaluated at $x = 1$, the second shadow could be the polynomial evaluated at $x = 2$, and so forth.

Since the quadratic polynomial has three unknown coefficients, a, b, and M, any three shadows can be used to create three equations. Two shadows cannot. One shadow cannot. Four or five shadows are redundant.

For example, let M be 11. To construct a (3, 5)-threshold scheme, where any three of five people can reconstruct M, first generate a quadratic equation (7 and 8 were chosen randomly):

$$F(x) = (7x^2 + 8x + 11) \bmod 13$$

The five shadows are:

$$k_1 = F(1) = 7 + 8 + 11 \equiv 0 \pmod{13}$$
$$k_2 = F(2) = 28 + 16 + 11 \equiv 3 \pmod{13}$$
$$k_3 = F(3) = 63 + 24 + 11 \equiv 7 \pmod{13}$$

$$k_4 = F(4) = 112 + 32 + 11 \equiv 12 \pmod{13}$$
$$k_5 = F(5) = 175 + 40 + 11 \equiv 5 \pmod{13}$$

To reconstruct M from three of the shadows, for example k_2, k_3, and k_5, solve the set of linear equations:

$$a * 2^2 + b * 2 + M \equiv 3 \pmod{13}$$
$$a * 3^2 + b * 3 + M \equiv 7 \pmod{13}$$
$$a * 5^2 + b * 5 + M \equiv 5 \pmod{13}$$

The solution will be $a = 7$, $b = 8$, and $M = 11$. So M is recovered.

This sharing scheme can be easily implemented for larger numbers. If you want to divide the message into 30 equal parts such that any six can get together and reproduce the message, give each of the 30 people the evaluation of a polynomial of degree 5.

$$F(x) = ax^5 + bx^5 + cx^4 + dx^3 + ex^2 + fx + M \pmod{p}$$

Six people can solve for the six unknowns (including M); five people cannot learn anything about M.

The most mind-boggling aspect of secret sharing is that if the coefficients are picked randomly, five people with infinite computing power can't learn anything more than the length of the message (which each of them knows anyway). This is as secure as a one-time pad; an attempt at exhaustive search (that is, trying all possible sixth shadows) will reveal that any conceivable message could be the secret. This is true for all the secret-sharing schemes presented here.

Vector Scheme

George Blakley invented a scheme using points in space [182]. The message is defined as a point in m-dimensional space. Each shadow is the equation of an $(m - 1)$-dimensional hyperplane that includes the point. The intersection of any m of the hyperplanes exactly determines the point.

For example, if three shadows are required to reconstruct the message, then it is a point in three-dimensional space. Each shadow is a different plane. With one shadow, you know the point is somewhere on the plane. With two shadows, you know the point is somewhere on the line formed where the two planes intersect. With three shadows, you can determine the point exactly: the intersection of the three planes.

Asmuth-Bloom

This scheme uses prime numbers [65]. For an (m, n)-threshold scheme, choose a large prime, p, greater than M. Then choose n numbers less than p, $d_1, d_2, \ldots, d_n$, such that:

1. The d values are in increasing order; $d_i < d_{i+1}$
2. Each d_i is relatively prime to every other d_i
3. $d_1 * d_2 * \ldots * d_m > p * d_{n-m+2} * d_{n-m+3} * \ldots * d_n$

To distribute the shadows, first choose a random value r and compute

$$M' = M + rp$$

The shadows, k_i, are

$$k_i = M' \bmod d_i$$

Any m shadows can get together and reconstruct M using the Chinese remainder theorem, but any $m - 1$ cannot. See [65] for details.

Karnin-Greene-Hellman

This scheme uses matrix multiplication [818]. Choose $n + 1$ m-dimensional vectors, $V_0, V_1, \ldots, V_n$, such that any possible $m * m$ matrix formed out of those vectors has rank m. The vector U is a row vector of dimension $m + 1$.

M is the matrix product $U \cdot V_0$. The shadows are the products $U \cdot V_i$, where i is a number from 1 to n.

Any m shadows can be used to solve the $m * m$ system of linear equations, where the unknowns are the coefficients of U. UV_0 can be computed from U. Any $m - 1$ shadows cannot solve the system of linear equations and therefore cannot recover the secret.

Advanced Threshold Schemes

The previous examples illustrate only the simplest threshold schemes: Divide a secret into n shadows such that any m can be used to recover the secret. These algorithms can be used to create far more complicated schemes. The following examples will use Shamir's algorithm, although any of the others will work.

To create a scheme in which one person is more important than another, give that person more shadows. If it takes five shadows to recreate a secret and one person has three shadows while everyone else has only one, then that person and two other people can recreate the secret. Without that person, it takes five to recreate the secret.

Two or more people could get multiple shadows. Each person could have a different number of shadows. No matter how the shadows are distributed, any m of them can be used to reconstruct the secret. Someone with $m - 1$ shadows, be it one person or a roomful of people, cannot do it.

In other types of schemes, imagine a scenario with two hostile delegations. You can share the secret so that two people from the 7 in Delegation A and 3 people from the 12 in Delegation B are required to reconstruct the secret. Make a polynomial of degree 3 that is the product of a linear expression and a quadratic expression. Give everyone from Delegation A a shadow that is the result of an evaluation of the linear equation; give everyone from Delegation B a shadow that is the evaluation of the quadratic equation.

Any two shadows from Delegation A can be used to reconstruct the linear equation, but no matter how many other shadows the group has, they cannot get any information about the secret. The same is true for Delegation B: They can get three shadows together to reconstruct the quadratic equation, but they cannot get any

more information necessary to reconstruct the secret. Only when the two delegations share their equations can they be multiplied to reconstruct the secret.

In general, any type of sharing scheme that can be imagined can be implemented. All you have to do is to envision a system of equations that corresponds to the particular scheme. Some excellent papers on generalized secret-sharing schemes are [1462,1463,1464].

Sharing a Secret with Cheaters

This algorithm modifies the standard (m, n)-threshold scheme to detect cheaters [1529]. I demonstrate this using the LaGrange scheme, although it works with the others as well.

Choose a prime, p, that is both larger than n and larger than

$$(s-1)(m-1)/e+m$$

where s is the largest possible secret and e is the probability of successful cheating. You can make e as small as you want; it just makes the computation more complex. Construct your shadows as before, except instead of using $1, 2, 3, \ldots, n$ for x_i, choose random numbers between 1 and $p-1$ for x_i.

Now, when Mallory sneaks into the secret reconstruction meeting with his false share, his share has a high probability of not being possible. An impossible secret is, of course, a fake secret. See [1529] for the math.

Unfortunately, while Mallory is exposed as a cheater, he still learns the secret (assuming that there are m other valid shares). Another protocol, from [1529,975], prevents that. The basic idea is to have a series of k secrets, such that none of the participants knows beforehand which is correct. Each secret is larger than the one before, except for the real secret. The participants combine their shadows to generate one secret after the other, until they create a secret that is less than the previous secret. That's the correct one.

This scheme will expose cheaters early, before the secret is generated. There are complications when the participants deliver their shadows one at a time; refer to the papers for details. Other papers on the detection and prevention of cheaters in threshold schemes are [355,114,270].

23.3 Subliminal Channel

Ong-Schnorr-Shamir

This subliminal channel (see Section 4.2), designed by Gustavus Simmons [1458,1459,1460], uses the Ong-Schnorr-Shamir identification scheme (see Section 20.5). As in the original scheme, the sender (Alice) chooses a public modulus, n, and a private key, k, such that n and k are relatively prime. Unlike the original scheme, k is shared between Alice and Bob, the recipient of the subliminal message.

The public key is calculated:

$$h = -k^2 \bmod n$$

If Alice wants to send the subliminal message M by means of the innocuous message M', she first confirms that M' and n are relatively prime, and that M and n are relatively prime.

Alice calculates

$$S_1 = 1/2 * ((M'/M + M)) \bmod n$$
$$S_2 = k/2 * ((M'/M - M)) \bmod n$$

Together, the pair, S_1 and S_2, is the signature under the traditional Ong-Schnorr-Shamir scheme and the carrier of the subliminal message.

Walter the warden (remember him?) can authenticate the message as described by the Ong-Schnorr-Shamir signature scheme, but Bob can do better. He can authenticate the message (it is always possible that Walter can make his own messages). He confirms that

$$S_1^2 - S_2^2/k^2 \equiv M' \pmod n$$

If the message is authentic, the receiver can recover the subliminal message using this formula:

$$M = M'/(S_1 + S_2 k^{-1}) \bmod n$$

This works, but remember that the basic Ong-Schnorr-Shamir has been broken.

ElGamal

Simmons's second subliminal channel [1459], described in [1407,1473], is based on the ElGamal signature scheme (see Section 19.6).

Key generation is the same as the basic ElGamal signature scheme. First choose a prime, p, and two random numbers, g and r, such that both g and r are less than p. Then calculate

$$K = g^r \bmod p$$

The public key is K, g, and p. The private key is r. Besides Alice, Bob also knows r; it is the key that is used to send and read the subliminal message in addition to being the key used to sign the innocuous message.

To send a subliminal message M using the innocuous message M', M and p must be all relatively prime to each other, and M and $p - 1$ must be relatively prime. Alice calculates

$$X = g^M \bmod p$$

and solves the following equation for Y (using the extended Euclidean algorithm):

$$M' = rX + MY \bmod (p - 1)$$

As in the basic ElGamal scheme, the signature is the pair: X and Y.

Walter can verify the ElGamal signature. He confirms that

$$K^X X^Y \equiv g^{M'} \pmod p$$

Bob can recover the subliminal message. First he confirms that

$$(g^r)^X X^Y \equiv g^{M'} \pmod p$$

If it does, he accepts the message as genuine (not from Walter).

Then, to recover M, he computes

$$M = (Y^{-1}(M' - rX)) \bmod (p - 1)$$

For example, let $p = 11$ and $g = 2$. The private key, r, is chosen to be 8. This means the public key, which Walter can use to verify the signature, is $g^r \bmod p = 2^8 \bmod 11 = 3$.

To send the subliminal message $M = 9$, using innocuous message $M' = 5$, Alice confirms that 9 and 11 are relatively prime and that 5 and 11 are relatively prime. She also confirms that 9 and $11 - 1 = 10$ are relatively prime. They are, so she calculates

$$X = g^M \bmod p = 2^9 \bmod 11 = 6$$

Then, she solves the following equation for Y:

$$5 = 8 * 6 + 9 * Y \bmod 10$$

$Y = 3$, so the signature is the pair, X and Y: 6 and 3.

Bob confirms that

$$(g^r)^X X^Y \equiv g^{M'} \pmod p$$
$$(2^8)^6 6^3 \equiv 2^5 \pmod{11}$$

It does (do the math yourself if you don't trust me), so he then recovers the subliminal message by calculating

$$M = (Y^{-1}(M' - rX)) \bmod (p - 1) = 3^{-1}(5 - 8 * 6) \bmod 10 = 7(7) \bmod 10 = 49 \bmod 10 = 9$$

ESIGN

A subliminal channel can be added to ESIGN [1460] (see Section 20.6).

In ESIGN, the secret key is a pair of large prime numbers, p and q, and the public key is $n = p^2q$. With a subliminal channel, the private key is three primes, p, q, and r, and the public key is n, such that

$$n = p^2qr$$

The variable, r, is the extra piece of information that Bob needs to read the subliminal message.

To sign a normal message, Alice first picks a random number, x, such that x is less than pqr and computes:

$$w, \text{ the least integer that is larger than } (H(m) - x^k \bmod n)/pqr)$$
$$s = x + ((w/kx^{k-1}) \bmod p)pqr$$

$H(m)$ is the hash of the message; k is a security parameter. The value s is the signature.

To verify the signature, Bob computes $s^k \bmod n$. He also computes a, which is the least integer larger than the number of bits of n divided by 3. If $H(m)$ is less than or equal to $s^k \bmod n$, and if $s^k \bmod n$ is less than $H(m) + 2^a$, then the signature is considered valid.

To send a subliminal message, M, using the innocuous message, M', Alice calcu-

lates s using M in place of $H(m)$. This means that the message must be smaller than p^2qr. She then chooses a random value, u, and calculates

$$x' = M' + ur$$

Then, use this x' value as the "random number" x to sign M'. This second s value is sent as a signature.

Walter can verify that s (the second s) is a valid signature of M'.

Bob can also authenticate the message in the same way. But, since he also knows r, he can calculate

$$s = x' + ypqr = M + ur + ypqr \equiv M \pmod{r}$$

This implementation of a subliminal channel is far better than the previous two. In the Ong-Schnorr-Shamir and ElGamal implementations, Bob has Alice's private key. Besides being able to read subliminal messages from Alice, Bob can impersonate Alice and sign normal documents. Alice can do nothing about this; she must trust Bob to set up this subliminal channel.

The ESIGN scheme doesn't suffer from this problem. Alice's private key is the set of three primes: p, q, and r. Bob's secret key is just r. He knows $n = p^2qr$, but to recover p and q he has to factor that number. If the primes are large enough, Bob has just as much trouble impersonating Alice as would Walter or anyone else.

DSA

There is also a subliminal channel in DSA (see Section 20.1) [1468,1469,1473]. In fact, there are several. The simplest subliminal channel involves the choice of k. It is supposed to be a 160-bit random number. However, if Alice chooses a particular k, then Bob, who also knows Alice's private key, can recover it. Alice can send Bob a 160-bit subliminal message in each DSA signature; everyone else simply verifies Alice's signature. Another complication: Since k should be random, Alice and Bob need to share a one-time pad and encrypt the subliminal message with the one-time pad to generate a k.

DSA has subliminal channels that do not require Bob to know Alice's private key. These also involve choosing particular values of k, but cannot be used to send 160 bits of information. This scheme, presented in [1468,1469], allows Alice and Bob to exchange one bit of subliminal information per signed message.

(1) Alice and Bob agree on a random prime, P (different from the parameter p in the signature scheme). This is their secret key for the subliminal channel.

(2) Alice signs an innocuous message, M. If she wants to send Bob the subliminal bit, 1, she makes sure the r parameter of the signature is a quadratic residue modulo P. If she wants to send him a 0, she makes sure the r parameter is a quadratic nonresidue modulo P. She does this by signing the message with random k values until she gets a signature with an r with the requisite property. Since quadratic residues and quadratic nonresidues are equally likely, this shouldn't be too difficult.

(3) Alice sends the signed message to Bob.

(4) Bob verifies the signature to make sure the message is authentic. Then he checks whether r is a quadratic residue or a quadratic nonresidue modulo P and recovers the subliminal bit.

Sending multiple bits via this method involves making r either a quadratic residue or a quadratic nonresidue modulo a variety of parameters. See [1468,1469] for details.

This scheme can be easily extended to send multiple subliminal bits per signature. If Alice and Bob agree on two random primes, P and Q, Alice can send two bits by choosing a random k such that r is either a quadratic residue mod P or a quadratic nonresidue mod P, and either a quadratic residue mod Q or a quadratic nonresidue mod Q. A random value of k has a 25 percent chance of producing an r of the correct form.

Here's how Mallory, an unscrupulous implementer of DSA, can have the algorithm leak 10 bits of Alice's private key every time she signs a document.

(1) Mallory puts his implementation of DSA in a tamperproof VLSI chip, so that no one can examine its inner workings. He creates a 14-bit subliminal channel in his implementation of DSA. That is, he chooses 14 random primes, and has the chip choose a value of k such that r is either a quadratic residue or a quadratic nonresidue modulo each of those 14 primes, depending on the subliminal message.

(2) Mallory distributes the chips to Alice, Bob, and everyone else who wants them.

(3) Alice signs a message normally, using her 160-bit private key, x.

(4) The chip randomly chooses a 10-bit block of x: the first 10 bits, the second 10 bits, and so on. Since there are 16 possible 10-bit blocks, a 4-bit number can identify which block it is. This 4-bit identifier, plus the 10 bits of the key, is the 14-bit subliminal message.

(5) The chip tries random values of k until it finds one that has the correct quadratic residue properties to send the subliminal message. The odds of a random k being of the correct form are 1 in 16,384. Assuming the chip can test 10,000 values of k per second, it will find one in less than two seconds. This computation does not involve the message and can be performed offline, before Alice wants to sign a message.

(6) The chip signs the message normally, using the value of k chosen in step (5).

(7) Alice sends the digital signature to Bob, or publishes it on the network, or whatever.

(8) Mallory recovers r and, because he knows the 14 primes, decrypts the subliminal message.

It's scary that even if Alice knows what is happening, she cannot prove it. As long as those 14 secret primes stay secret, Mallory is safe.

Foiling the DSA Subliminal Channel

The subliminal channel relies on the fact that Alice can choose k to transmit subliminal information. To foil the subliminal channel, Alice cannot be allowed to choose k. However, neither can anyone else; if someone else were allowed to choose k, it would allow that person to forge Alice's signature. The only solution is for Alice to jointly generate k with another party, call him Bob, in such a way that Alice cannot control a single bit of k and Bob cannot know a single bit of k. At the end of the protocol, Bob should be able to verify that Alice used the k that they jointly generated.

Here's the protocol [1470,1472,1473]:

(1) Alice chooses k' and sends Bob

$$u = g^{k'} \bmod p$$

(2) Bob chooses k'' and sends it to Alice.

(3) Alice calculates $k = k'k'' \bmod (p-1)$. She uses k to sign her message, M, with the DSA and sends Bob the signature: r and s.

(4) Bob verifies that

$$(u^{k''} \bmod p) \bmod q = r$$

If it does, he knows that k was used to sign M.

After step (4), Bob knows that no subliminal information can be embedded in r. If he is a trusted party, he can certify that Alice's signature is subliminal-free. Others will have to trust his certification; Bob cannot prove this fact to a third party with a transcript of the protocol.

A surprising result is that if Bob wants to, he can use this protocol to create his own subliminal channel. Bob can embed a subliminal message in one of Alice's signatures by choosing k'' with certain characteristics. When Simmons discovered this, he dubbed it the "Cuckoo's Channel." Details on how the Cuckoo's Channel works, and a three-pass protocol for generating k that prevents it, are discussed in [1471,1473].

Other Schemes

Any signature scheme can be converted into a subliminal channel [1458,1460, 1406]. A protocol for embedding a subliminal channel in the Fiat-Shamir and Feige-Fiat-Shamir protocols, as well as possible abuses of the subliminal channel, can be found in [485].

23.4 UNDENIABLE DIGITAL SIGNATURES

This undeniable signature algorithm (see Section 4.3) is by David Chaum [343,327]. First, a large prime, p, and a primitive element, g, are made public, and used by a group of signers. Alice has a private key, x, and a public key, $g^x \bmod p$.

To sign a message, Alice computes $z = m^x \bmod p$. That's all she has to do. Verification is a little more complicated.

(1) Bob chooses two random numbers, a and b, both less than p, and sends Alice:

$$c = z^a (g^x)^b \bmod p$$

(2) Alice computes $t = x^{-1} \bmod (p - 1)$, and sends Bob:

$$d = c^t \bmod p$$

(3) Bob confirms that

$$d \equiv m^a g^b \pmod p$$

If it is, he accepts the signature as genuine.

Imagine that Alice and Bob went through this protocol, and Bob is now convinced that Alice signed the message. Bob wants to convince Carol, so he shows her a transcript of the protocol. Dave, however, wants to convince Carol that some other person signed the document. He creates a fake transcript of the protocol. First he generates the message in step (1). Then in step (3) he generates d and the fake transmission from this other person in step (2). Finally, he creates the message in step (2). To Carol, both Bob's and Dave's transcripts are identical. She cannot be convinced of the signature's validity unless she goes through the protocol herself.

Of course, if she were watching over Bob's shoulder as he completed the protocol, she would be convinced. Carol has to see the steps done in order, just as Bob does.

There may be a problem with this signature scheme, but I know of no details. Please pay attention to the literature before you use it.

Another protocol not only has a confirmation protocol—Alice can convince Bob that her signature is valid—but it also has a disavowal protocol; Alice can use a zero-knowledge interactive protocol to convince him that the signature is not valid, if it is not [329].

Like the previous protocol, a group of signers use a shared public large prime, p, and a primitive element, g. Alice has a unique private key, x, and a public key, $g^x \bmod p$. To sign a message, Alice computes $z = m^x \bmod p$.

To verify a signature:

(1) Bob chooses two random numbers, a and b, both less than p, and sends Alice:

$$c = m^a g^b \bmod p$$

(2) Alice chooses a random number, q, less than p, and computes and sends to Bob:

$$s_1 = cg^q \bmod p,\ s_2 = (cg^q)^x \bmod p$$

(3) Bob sends Alice a and b, so that Alice can confirm that Bob did not cheat in step (1).

(4) Alice sends Bob q, so that Bob can use m^x and reconstruct s_1 and s_2. If

$$s_1 \equiv cg^q \pmod{p}$$
$$s_2 \equiv (g^x)^{b+q} z^a \pmod{p}$$

then the signature is valid.

Alice can also disavow a signature, z, for a message, m. See [329] for details.

Additional protocols for undeniable signatures can be found in [584,344]. Lein Harn and Shoubao Yang proposed a group undeniable signature scheme [700].

Convertible Undeniable Signatures

An algorithm for a **convertible undeniable signature**, which can be verified, disavowed, and also converted to a conventional digital signature is given in [213]. It's based on the ElGamal digital signature algorithm.

Like ElGamal, first choose two primes, p and q, such that q divides $p - 1$. Now you have to create a number, g, less than q. First choose a random number, h, between 2 and $p - 1$. Calculate

$$g = h^{(p-1)/q} \bmod p$$

If g equals the 1, choose another random h. If it doesn't, stick with the g you have.

The private keys are two different random numbers, x and z, both less than q. The public keys are p, q, g, y, and u, where

$$y = g^x \bmod p$$
$$u = g^z \bmod p$$

To compute the convertible undeniable signature of message m (which is actually the hash of a message), first choose a random number, t, between 1 and $q - 1$. Then compute

$$T = g^t \bmod p$$

and

$$m' = Ttzm \bmod q.$$

Now, compute the standard ElGamal signature on m'. Choose a random number, R, such that R is less than and relatively prime to $p - 1$. Then compute $r = g^R \bmod p$, and use the extended Euclidean algorithm to compute s, such that

$$m' \equiv rx + Rs \pmod{q}$$

The signature is the ElGamal signature (r, s), and T.

Here's how Alice verifies her signature to Bob:

(1) Bob generates two random numbers, a and b. He computes $c = T^{Tma} g^b \bmod p$ and sends that to Alice.

(2) Alice generates a random number, k, and computes $h_1 = cg^k \bmod p$, and $h_2 = h_1^z \bmod p$, and sends both of those numbers to Bob.

(3) Bob sends Alice a and b.

(4) Alice verifies that $c = T^{Tma}g^b \bmod p$. She sends k to Bob.

(5) Bob verifies that $h_1 = T^{Tma}g^{b+k} \bmod p$, and that $h_2 = y^{ta}r^{sa}u^{b+k} \bmod p$.

Alice can convert all of her undeniable signatures to normal signatures by publishing z. Now, anyone can verify her signature without her help.

Undeniable signature schemes can be combined with secret-sharing schemes to create **distributed convertible undeniable signatures** [1235]. Someone can sign a message, then distribute the ability to confirm that the signature is valid. He might, for example, require three out of five people to participate in the protocol in order to convince Bob that the signature is valid. Improvements on this notion deleted the requirement for a trusted dealer [700,1369].

23.5 Designated Confirmer Signatures

Here's how Alice can sign a message and Bob can verify it, such that Carol can verify Alice's signature at some later time to Dave (see Section 4.4) [333].

First, a large prime, p, and a primitive element, g, are made public and used by a group of users. The product of two primes, n, is also public. Carol has a private key, z, and a public key is $h = g^x \bmod p$.

In this protocol Alice can sign m such that Bob is convinced that the signature is valid, but cannot convince a third party.

(1) Alice chooses a random x and computes

$$a = g^x \bmod p$$
$$b = h^x \bmod p$$

She computes the hash of m, $H(m)$, and the hash of a and b concatenated, $H(a,b)$. She then computes

$$j = (H(m) \oplus H(a, b))^{1/3} \bmod n$$

and sends a, b, and j to Bob.

(2) Bob chooses two random numbers, s and t, both less than p, and sends Alice

$$c = g^s h^t \bmod p$$

(3) Alice chooses a random q less than p, and sends Bob

$$d = g^q \bmod p$$
$$e = (cd)^x \bmod p$$

(4) Bob sends Alice s and t.

(5) Alice confirms that

$$g^s h^t \equiv c \pmod p$$

Then she sends Bob q.

(6) Bob confirms

$$d \equiv g^q \pmod{p}$$
$$e/a^q \equiv a^s b^t \pmod{p}$$
$$H(m) \oplus H(a, b) \equiv j^{1/3} \bmod n$$

If they all check out, he accepts the signature as genuine.

Bob cannot use a transcript of this proof to convince Dave that the signature is genuine, but Dave can conduct a protocol with Alice's designated confirmer, Carol. Here's how Carol convinces Dave that a and b constitute a valid signature.

(1) Dave chooses a random u and v, both less than p, and sends Carol

$$k = g^u a^v \bmod p$$

(2) Carol chooses a random w, less than p, and sends Dave

$$l = g^w \bmod p$$
$$y = (kl)^z \bmod p$$

(3) Dave sends Carol u and v.

(4) Carol confirms that

$$g^u a^v \equiv k \pmod{p}$$

Then she sends Dave w.

(5) Dave confirms that

$$g^w \equiv l \pmod{p}$$
$$y/h^w \equiv h^u b^v \pmod{p}$$

If they both check out, he accepts the signature as genuine.

In another protocol Carol can convert the designated-confirmer protocol into a conventional digital signature. See [333] for details.

23.6 COMPUTING WITH ENCRYPTED DATA

The Discrete Logarithm Problem

There is a large prime, p, and a generator, g. Alice has a particular value for x, and wants to know e, such that

$$g^e \equiv x \pmod{p}$$

This is a hard problem, and Alice lacks the computational power to compute the result. Bob has the power to solve the problem—he represents the government, or a large computing organization, or whatever. Here's how Bob can do it without Alice revealing x [547,4]:

(1) Alice chooses a random number, r, less than p.

(2) Alice computes

$$x' = xg^r \bmod p$$

(3) Alice asks Bob to solve

$$g^{e'} \equiv x' \pmod{p}$$

(5) Bob computes e' and sends it to Alice.

(6) Alice recovers e by computing

$$e = (e' - r) \bmod (p - 1)$$

Similar protocols for the quadratic residuosity problem and for the primitive root problem are in [3,4]. (See also Section 4.8.)

23.7 Fair Coin Flips

The following protocols allow Alice and Bob to flip a fair coin over a data network (see Section 4.9) [194]. This is an example of flipping a coin into a well (see Section 4.10). At first, only Bob knows the result of the coin toss and tells it to Alice. Later, Alice may check to make sure that Bob told her the correct outcome of the toss.

Coin Flipping Using Square Roots

Coin-flip subprotocol:

(1) Alice chooses two large primes, p and q, and sends their product, n to Bob.

(2) Bob chooses a random positive integer, r, such that r is less than $n/2$. Bob computes

$$z = r^2 \bmod n$$

and sends z to Alice.

(3) Alice computes the four square roots of $z \pmod{n}$. She can do this because she knows the factorization of n. Let's call them $+x$, $-x$, $+y$, and $-y$. Call x' the smaller of these two numbers:

$$x \bmod n$$

$$-x \bmod n$$

Similarly, call y' the smaller of these two numbers:

$$y \bmod n$$

$$-y \bmod n$$

Note that r is equal either to x' or y'.

(4) Alice guesses whether $r = x'$ or $r = y'$, and sends her guess to Bob.

(5) If Alice's guess is correct, the result of the coin flip is heads. If Alice's guess is incorrect, the result of the coin flip is tails. Bob announces the result of the coin flip.

Verification subprotocol:

(6) Alice sends p and q to Bob.
(7) Bob computes x' and y' and sends them to Alice.
(8) Alice calculates r.

Alice has no way of knowing r, so her guess is real. She only tells Bob one bit of her guess in step (4) to prevent Bob from getting both x' and y'. If Bob has both of those numbers, he can change r after step (4).

Coin Flipping Using Exponentiation Modulo p

Exponentiation modulo a prime number, p, is used as a one-way function in this protocol [1306]:

Coin-flip subprotocol:

(1) Alice chooses a prime number, p, in such a way that the factorization of $p - 1$ is known and contains at least one large prime.
(2) Bob selects two primitive elements, h and t, in GF(p). He sends them to Alice.
(3) Alice checks that h and t are primitive and then chooses a random integer x, relatively prime to $p - 1$. She then computes one of the two values:

$$y = h^x \bmod p, \text{ or } y = t^x \bmod p$$

She sends y to Bob.
(4) Bob guesses whether Alice calculated y as a function of h or t, and sends his guess to Alice.
(5) If Bob's guess is correct, the result of the coin flip is heads. If Bob's guess is incorrect, the result of the coin flip is tails. Alice announces the result of the coin flip.

Verification subprotocol:

(6) Alice reveals x to Bob. Bob computes $h^x \bmod p$ and $t^x \bmod p$, to confirm that Alice has played fairly and to verify the result of the toss. He also checks that x and $p - 1$ are relatively prime.

For Alice to cheat, she has to know two integers, x and x', such that $h^x \equiv t^{x'} \pmod p$. If she knew those values, she would be able to calculate:

$$\log_t h = x'x^{-1} \bmod p - 1 \text{ and } \log_t h = x^{-1}x' \bmod p - 1$$

These are hard problems.

Alice would be able to do this if she knew $\log_t h$, but Bob chooses h and t in step (2). Alice has no other recourse except to try to compute the discrete logarithm. Alice could also attempt to cheat by choosing an x that is not relatively prime to $p - 1$, but Bob will detect that in step (6).

Bob can cheat if h and t are not primitive in GF(p), but Alice can easily check that after step (2) because she knows the prime factorization of $p - 1$.

One nice thing about this protocol is that if Alice and Bob want to flip multiple coins, they can use the same values for p, h, and t. Alice just generates a new x, and the protocol continues from step (3).

Coin Flipping Using Blum Integers

Blum integers can be used in a coin-flipping protocol.

(1) Alice generates a Blum integer, n, a random x relatively prime to n, $x_0 = x^2 \bmod n$, and $x_1 = x_0^2 \bmod n$. She sends n and x_1 to Bob.

(2) Bob guesses whether x_0 is even or odd.

(3) Alice sends x to Bob.

(4) Bob checks that n is a Blum integer (Alice would have to give Bob the factors of n and proofs of their primality, or execute some zero-knowledge protocol to convince him that n is a Blum integer), and he verifies that $x_0 = x^2 \bmod n$ and $x_1 = x_0^2 \bmod n$. If all this checks out, Bob wins the flip if he guessed correctly.

It is crucial that n be a Blum integer. Otherwise, Alice may be able to find an x'_0 such that $x'^2_0 \bmod n = x_0^2 \bmod n = x_1$, where x'_0 is also a quadratic residue. If x_0 were even and x'_0 were odd (or vice versa), Alice could freely cheat.

23.8 One-Way Accumulators

There is a simple one-way accumulator function [116] (see Section 4.12):

$$A(x_j, y) = x_{j-1}^{\,y} \bmod n$$

The numbers n (n is the product of two primes) and x_0 must be agreed upon in advance. Then, the accumulation of y_1, y_2, and y_3 would be

$$((x_0^{y_1} \bmod n)^{y_2} \bmod n)^{y_3} \bmod n$$

This computation is independent of the order of y_1, y_2, and y_3.

23.9 All-or-Nothing Disclosure of Secrets

This protocol allows multiple parties (at least two are required for the protocol to work) to buy individual secrets from a single seller (see Section 4.13) [1374,1175]. First, here's a definition. Take two bit strings, x and y. The **fixed bit index (FBI)** of x and y are the bits where the ith bit of x equals the ith bit of y.

For example:

$x = 110101001011$

$y = 101010000110$

FBI$(x, y) = \{1, 4, 5, 11\}$

(We're reading the bits from right to left, with the right-most bit as zero.)

Now, here's the protocol. Alice is the seller. Bob and Carol are buyers. Alice has k n-bit secrets: $S_1, S_2, \ldots, S_k$. Bob wants to buy secret S_b; Carol wants to buy secret S_c.

(1) Alice generates a public-key/private-key key pair and tells Bob (but not Carol) the public key. She generates another public-key/private-key key pair and tells Carol (but not Bob) the public key.

(2) Bob generates k n-bit random numbers, $B_1, B_2, \ldots, B_k$, and tells them to Carol. Carol generates k n-bit random numbers, $C_1, C_2, \ldots, C_k$, and tells them to Bob.

(3) Bob encrypts C_b (remember, S_b is the secret he wants to buy) with the public key from Alice. He computes the FBI of C_b and the result he just encrypted. He sends this FBI to Carol.

Carol encrypts B_c (remember, S_c is the secret she wants to buy) with the public key from Alice. She computes the FBI of B_c and the result she just encrypted. She sends this FBI to Bob.

(4) Bob takes each of the n-bit numbers $B_1, B_2, \ldots, B_k$, and replaces every bit whose index is not in the FBI he received from Carol with its complement. He sends this new list of n-bit numbers, $B'_1, B'_2, \ldots, B'_k$, to Alice.

Carol takes each of the n-bit numbers $C_1, C_2, \ldots, C_k$, and replaces every bit whose index is not in the FBI she received from Bob with its complement. She sends this new list of n-bit numbers, $C'_1, C'_2, \ldots, C'_k$, to Alice.

(5) Alice decrypts all C'_i with Bob's private key, giving her k n-bit numbers: $C''_1, C''_2, \ldots, C''_k$. She computes $S_i \oplus C''_i$, for $i = 1$ to k, and sends the results to Bob.

Alice decrypts all B'_i with Carol's private key, giving her k n-bit numbers: $B''_1, B''_2, \ldots, B''_k$. She computes $S_i \oplus B''_i$, for $i = 1$ to k, and sends the results to Carol.

(6) Bob computes S_b by XORing C_b and the bth number he received from Alice.

Carol computes S_c by XORing B_c and the cth number she received from Alice.

This is complicated. An example will go a long way to help.

Alice has the following eight 12-bit secrets for sale: $S_1 = 1990$, $S_2 = 471$, $S_3 = 3860$, $S_4 = 1487$, $S_5 = 2235$, $S_6 = 3751$, $S_7 = 2546$, and $S_8 = 4043$. Bob wants to buy S_7. Carol wants to buy S_2.

(1) Alice uses the RSA algorithm. The key pair she will use with Bob is: $n = 7387$, $e = 5145$, and $d = 777$. The key pair she will use with Carol is: $n = 2747$, $e = 1421$, and $d = 2261$. She tells Bob and Carol each their public key.

(2) Bob generates eight 12-bit random numbers, $B_1 = 743$, $B_2 = 1988$, $B_3 = 4001$, $B_4 = 2942$, $B_5 = 3421$, $B_6 = 2210$, $B_7 = 2306$, and $B_8 = 222$, and tells them to Carol. Carol generates eight 12-bit random numbers, $C_1 = 1708$, $C_2 = 711$, $C_3 = 1969$, $C_4 = 3112$, $C_5 = 4014$, $C_6 = 2308$, $C_7 = 2212$, and $C_8 = 222$, and tells them to Bob.

(3) Bob wants to buy S_7, so he encrypts C_7 with the public key that Alice gave him.

$$2212^{5145} \bmod 7387 = 5928$$

Now:

$$2212 = 0100010100100$$
$$5928 = 1011100101000$$

So, the FBI of those two numbers is {0, 1, 4, 5, 6}. He sends this to Carol.

Carol wants to buy S_2, so she encrypts B_2 with the public key that Alice gave her and computes the FBI of B_2 with the result of her encryption. She sends {0, 1, 2, 6, 9, 10} to Bob.

(4) Bob takes $B_1, B_2, \ldots, B_8$, and replaces every bit whose index is not in the set {0, 1, 2, 6, 9, 10} with its complement. For example:

$$B_2 = 111111000100 = 1988$$
$$B'_2 = 011001111100 = 1660$$

He sends $B'_1, B'_2, \ldots, B'_8$, to Alice.

Carol takes $C_1, C_2, \ldots, C_8$, and replaces every bit whose index is not in the set {0, 1, 4, 5, 6} with its complement. For example:

$$C_7 = 0100010100100 = 2212$$
$$C'_7 = 1011100101000 = 5928$$

She sends $C'_1, C'_2, \ldots, C'_8$, to Alice.

(5) Alice decrypts all C'_i with Bob's private key and XORs the results with S_i. For example, for $i = 7$:

$$5928^{777} \bmod 7387 = 2212; 2546 \oplus 2212 = 342$$

She sends the results to Bob.

Alice decrypts all B'_i with Carol's private key and XORs the results with S_i. For example, for $i = 2$:

$$1660^{2261} \pmod{2747} = 1988; 471 \oplus 1988 = 1555$$

She sends the results to Carol.

(6) Bob computes S_7 by XORing C_7 and the seventh number he received from Alice:

$$2212 \oplus 342 = 2546$$

Carol computes S_2 by XORing B_2 and the second number she received from Alice.

$$1988 \oplus 1555 = 471$$

The protocol works for any number of buyers. If Bob, Carol, and Dave want to buy secrets, Alice gives each buyer two public keys, one for each of the others. Each buyer gets a set of numbers from each other buyer. Then, they complete the protocol with Alice for each of their sets of numbers and XOR all of their final results from Alice to get their secret. More details are in [1374,1175].

Unfortunately, a pair of dishonest parties can cheat. Alice and Carol, working together, can easily find out what secret Bob is getting: If they know the FBI of C_b and Bob's encryption algorithm, they can find b such that C_b has the right FBI. And Bob and Carol, working together, can easily get all the secrets from Alice.

If you assume honest parties, there's an easier protocol [389].

(1) Alice encrypts all of the secrets with RSA and sends them to Bob:

$$C_i = S_i^e \bmod n$$

(2) Bob chooses his secret C_b, picks a random r, and sends C' to Alice.

$$C' = C_b r^e \bmod n$$

(3) Alice sends Bob P'.

$$P' = C'^d \bmod n$$

(4) Bob calculates S_b.

$$S_b = P' r^{-1} \bmod n$$

If the parties may be dishonest, Bob can prove in zero-knowledge that he knows some r such that $C' = C_b r^e \bmod n$ and keep b secret until Alice gives him P' in step (3) [246].

23.10 FAIR AND FAILSAFE CRYPTOSYSTEMS

Fair Diffie-Hellman

Fair cryptosystems are a way to do key escrowing in software (see Section 4.14). This example is from Silvio Micali [1084,1085]. It is patented [1086,1087].

In the basic Diffie-Hellman scheme, a group of users share a prime, p, and a generator, g. Alice's private key is s, and her public key is $t = g^s \bmod p$.

Here's how to make Diffie-Hellman fair (this example uses five trustees).

(1) Alice chooses five integers, s_1, s_2, s_3, s_4, and s_5, each less than $p - 1$. Alice's private key is

$$s = (s_1 + s_2 + s_3 + s_4 + s_5) \bmod p - 1$$

and her public key is

$$t = g^s \bmod p$$

Alice also computes

$$t_i = g^{s_i} \bmod p, \text{ for } i = 1 \text{ to } 5$$

Alice's public shares are t_i, and her private shares are s_i.

(2) Alice sends a private piece and corresponding public piece to each trustee. For example, she sends s_1 and t_1 to trustee 1. She sends t to the KDC.

(3) Each trustee verifies that

$$t_i = g^{s_i} \bmod p$$

If it does, the trustee signs t_i and sends it to the KDC. The trustee stores s_i in a secure place.

(4) After receiving all five public pieces, the KDC verifies that

$$t = (t_1 * t_2 * t_3 * t_4 * t_5) \bmod p$$

If it does, the KDC approves the public key.

At this point, the KDC knows that the trustees each have a valid piece, and that they can reconstruct the private key if required. However, neither the KDC nor any four of the trustees working together can reconstruct Alice's private key.

Micali's papers [1084,1085] also contain a procedure for making RSA fair and for combining a threshold scheme with the fair cryptosystem, so that m out of n trustees can reconstruct the private key.

Failsafe Diffie-Hellman

Like the previous protocol, a group of users share a prime, p, and a generator, g. Alice's private key is s, and her public key is $t = g^s \bmod p$.

(1) The KDC chooses a random number, B, between 0 and $p - 2$, and commits to B using a bit-commitment protocol (see Section 4.9).

(2) Alice chooses a random number, A, between 0 and $p - 2$. She sends $g^A \bmod p$ to the KDC.

(3) The user "shares" A with each trustee using a verifiable secret-sharing scheme (see Section 3.7).

(4) The KDC reveals B to Alice.

(5) Alice verifies the commitment from step (1). Then she sets her public key as

$$t = (g^A)g^B \bmod p$$

She sets her private key as

$$s = (A + B) \bmod (p - 1)$$

The trustees can reconstruct A. Since the KDC knows B, this is enough to reconstruct s. And Alice cannot make use of any subliminal channels to send unauthorized information. This protocol, discussed in [946,833] is being patented.

23.11 ZERO-KNOWLEDGE PROOFS OF KNOWLEDGE

Zero-Knowledge Proof of a Discrete Logarithm

Peggy wants to prove to Victor that she knows an x that satisfies

$$A^x \equiv B \pmod{p}$$

where p is a prime, and x is a random number relatively prime to $p - 1$. The numbers A, B, and p are public, and x is secret. Here's how Peggy can prove she knows x without revealing it (see Section 5.1) [338,337].

(1) Peggy generates t random numbers, $r_1, r_2, \ldots, r_t$, where all r_i are less than $p - 1$.

(2) Peggy computes $h_i = A^{r_i} \bmod p$, for all values of i, and sends them to Victor.

(3) Peggy and Victor engage in a coin-flipping protocol to generate t bits: $b_1, b_2, \ldots, b_t$.

(4) For all t bits, Peggy does one of the following:

   a) If $b_i = 0$, she sends Victor r_i

   b) If $b_i = 1$, she sends Victor $s_i = (r_i - r_j) \bmod (p - 1)$, where j is the lowest value for which $b_j = 1$

(5) For all t bits, Victor confirms one of the following:

   a) If $b_i = 0$, that $A^{r_i} \equiv h_i \pmod{p}$

   b) If $b_i = 1$, that $A^{s_i} \equiv h_i h_j^{-1} \pmod{p}$

(6) Peggy sends Victor Z, where

$$Z = (x - r_j) \bmod (p - 1)$$

(7) Victor confirms that

$$A^Z \equiv B h_j^{-1} \pmod{p}$$

Peggy's probability of successfully cheating is 2^{-t}.

Zero-Knowledge Proof of the Ability to Break RSA

Alice knows Carol's private key. Maybe she has broken RSA; maybe she has broken into Carol's house and stolen the key. Alice wants to convince Bob that she knows Carol's key. However, she doesn't want to tell Bob the key or even decrypt one of Carol's messages for Bob. Here's a zero-knowledge protocol by which Alice convinces Bob that she knows Carol's private key [888].

Carol's public key is e, her private key is d, and the RSA modulus is n.

(1) Alice and Bob agree on a random k and an m such that

$$km \equiv e \pmod{n}$$

They should choose the numbers randomly, using a coin-flip protocol to generate k and then computing m. If both k and m are greater than 3, the protocol continues. Otherwise, they choose again.

(2) Alice and Bob generate a random ciphertext, C. Again, they should use a coin-flip protocol.

(3) Alice, using Carol's private key, computes

$$M = C^d \bmod n$$

She then computes

$$X = M^k \bmod n$$

and sends X to Bob.

(4) Bob confirms that $X^m \bmod n = C$. If it does, he believes Alice.

A similar protocol can be used to demonstrate the ability to break a discrete logarithm problem [888].

Zero-Knowledge Proof that n Is a Blum Integer

There are no known truly practical zero-knowledge proofs that $n = pq$, where p and q are primes congruent to 3 modulo 4. However, if you allow n to be of the form $p^r q^s$, where r and s are odd, then the properties which make Blum integers useful in cryptography still hold. And there exists a zero-knowledge proof that n is of that form.

Assume Alice knows the factorization of the Blum integer n, where n is of the form previously discussed. Here's how she can prove to Bob that n is of that form [660].

(1) Alice sends Bob a number u which has a Jacobi symbol -1 modulo n.

(2) Alice and Bob jointly agree on random bits: $b_1, b_2, \ldots, b_k$.

(3) Alice and Bob jointly agree on random numbers: $x_1, x_2, \ldots, x_k$.

(4) For each $i = 1, 2, \ldots, k$, Alice sends Bob a square root modulo n, of one of the four numbers: $x_i, -x_i, ux_i, -ux_i$. The square root must have the Jacobi symbol b_i.

The odds of Alice successfully cheating are one in 2^k.

23.12 Blind Signatures

The notion of blind signatures (see Section 5.3) was invented by David Chaum [317,323], who also invented their first implementation [318]. It uses the RSA algorithm.

Bob has a public key, e, a private key, d, and a public modulus, n. Alice wants Bob to sign message m blindly.

(1) Alice chooses a random value, k, between 1 and n. Then she blinds m by computing

$$t = mk^e \bmod n$$

(2) Bob signs t

$$t^d = (mk^e)^d \bmod n$$

(3) Alice unblinds t^d by computing

$$s = t^d/k \bmod n$$

(4) And the result is

$$s = m^d \bmod n$$

This can easily be shown

$$t^d \equiv (mk^e)^d \equiv m^d k \pmod{n}, \text{ so } t^d/k = m^d k/k \equiv m^d \pmod{n}.$$

Chaum invented a family of more complicated blind signature algorithms in [320,324], called blind unanticipated signatures. These signatures are more complex in construction, but more flexible.

23.13 OBLIVIOUS TRANSFER

In this protocol by Michael Rabin [1286], Alice has a 50 percent chance of sending Bob two primes, p, and q. Alice will not know whether the transfer is successful. (See Section 5.5.) (This protocol can be used to send Bob any message with a 50 percent success rate if p and q reveal an RSA private key.)

(1) Alice sends Bob the product of the two primes: $n = pq$.

(2) Bob chooses a random x less than n, such that x is relatively prime to n. He sends Alice:

$$a = x^2 \bmod n$$

(3) Alice, knowing p and q, computes the four roots of a: x, $n - x$, y, and $n - y$. She chooses one of these roots at random and sends it to Bob.

(4) If Bob receives y or $n - y$, he can compute the greatest common divisor of $x + y$ and n, which is either p or q. Then, of course, $n/p = q$.
If Bob receives x or $n - x$, he can't compute anything.

This protocol may have a weakness: It might be the case that Bob can compute a number a such that given the square root of a you can calculate a factor of n all the time.

23.14 Secure Multiparty Computation

This protocol is from [1373]. Alice knows the integer i; Bob knows the integer j. Alice and Bob together wish to know whether $i \leq j$ or if $i > j$, but neither Alice nor Bob wish to reveal the integer each knows. This special case of secure multiparty computation (see Section 6.2) is sometimes known as **Yao's millionaire problem** [1627].

For this example, assume that i and j range from 1 to 100. Bob has a public key and a private key.

(1) Alice chooses a large random number, x, and encrypts it in Bob's public key.

$$c = E_B(x)$$

(2) Alice computes $c - i$ and sends the result to Bob.

(3) Bob computes the following 100 numbers:

$$y_u = D_B(c - i + u), \text{ for } 1 \leq u \leq 100$$

D_B is the decryption algorithm with Bob's private key.

He chooses a large random prime, p. (The size of p should be somewhat smaller than x. Bob doesn't know x, but Alice could easily tell him the size of x.) He then computes the following 100 numbers:

$$z_u = (y_u \bmod p), \text{ for } 1 \leq u \leq 100$$

He then verifies that, for all $u \neq v$

$$|z_u - z_v| \geq 2$$

and that for all u

$$0 < z_u < p - 1$$

If this is not true, Bob chooses another prime and tries again.

(4) Bob sends Alice this sequence of numbers in this exact order:

$$z_1, z_2, \ldots, z_j, z_{j+1} + 1, z_{j+2} + 1, \ldots, z_{100} + 1, p$$

(5) Alice checks whether the ith number in the sequence is congruent to $x \bmod p$. If it is, she concludes that $i \leq j$. If it is not, she concludes that $i > j$.

(6) Alice tells Bob the conclusion.

All the verification that Bob goes through in step (3) is to guarantee that no number appears twice in the sequence generated in step (4). Otherwise, if $z_a = z_b$, Alice knows that $a \leq j < b$.

The one drawback to this protocol is that Alice learns the result of the computation before Bob does. Nothing stops her from completing the protocol up to step (5) and then refusing to tell Bob the results in step (6). She could even lie to Bob in step (6).

Example of the Protocol

Assume they're using RSA. Bob's public key is 7 and his private key is 23; $n = 55$. Alice's secret value, i, is 4; Bob's secret value, j, is 2. (Assume that only the values 1, 2, 3, and 4 are possible for i and j.)

(1) Alice chooses $x = 39$, and $c = E_B(39) = 19$.

(2) Alice computes $c - i = 19 - 4 = 15$. She sends 15 to Bob.

(3) Bob computes the following 4 numbers:

$$y_1 = D_B(15 + 1) = 26$$
$$y_2 = D_B(15 + 2) = 18$$
$$y_3 = D_B(15 + 3) = 2$$
$$y_4 = D_B(15 + 4) = 39$$

He chooses $p = 31$ and calculates:

$$z_1 = (26 \bmod 31) = 26$$
$$z_2 = (18 \bmod 31) = 18$$
$$z_3 = (2 \bmod 31) = 2$$
$$z_4 = (39 \bmod 31) = 8$$

He does all the verifications and confirms that the sequence is fine.

(4) Bob sends Alice this sequence of numbers in this exact order:

$$26, 18, 2 + 1, 8 + 1, 31 = 26, 18, 3, 9, 31$$

(5) Alice checks whether the 4th number in the sequence is congruent to $x \bmod p$. Since $9 \not\equiv 39 \pmod{31}$, then $i > j$.

(6) Alice tells Bob.

This protocol can be used to create far more complicated protocols. A group of people can conduct a secret auction over a computer network. They arrange themselves in a logical circle, and through individual pairwise comparisons, determine which is offering the highest price. In order to prevent people from changing their bids in mid-auction, some sort of bit-commitment protocol could also be used. If the auction is a Dutch auction, then the highest bidder gets the item for his highest price. If it is an English auction, then he gets the item for the second-highest price. (This can be determined by a second round of pairwise comparisons.) Similar ideas have applications in bargaining, negotiations, and arbitration.

23.15 PROBABILISTIC ENCRYPTION

The notion of **probabilistic encryption** was invented by Shafi Goldwasser and Silvio Micali [624]. Although its theory makes it the most secure cryptosystem invented, its early implementation was impractical [625]. More recent implementations have changed that.

The point behind probabilistic encryption is to eliminate any information leaked with public-key cryptography. Because a cryptanalyst can always encrypt random messages with a public key, he can get some information. Assuming he has ciphertext $C = E_K(M)$ and is trying to recover plaintext message M, he can pick a random message M' and encrypt it: $C' = E_K(M')$. If $C' = C$, then he guessed the correct plaintext. If it's wrong, he just guesses again.

Also, no partial information is leaked about the original message. With public-key cryptography, sometimes a cryptanalyst can learn things about the bits: The XOR of bits 5, 17, and 39 is 1, and so on. With probabilistic encryption, even this type of information remains hidden.

Not a whole lot of information is to be gained here, but there are potential problems with allowing a cryptanalyst to encrypt random messages with your public key. Some information is being leaked to the cryptanalyst every time he encrypts a message. No one really knows how much.

Probabilistic encryption tries to eliminate that leakage. The goal is that no computation on the ciphertext, or on any other trial plaintexts, can give the cryptanalyst any information about the corresponding plaintext.

In probabilistic encryption, the encrypting algorithm is probabilistic rather than deterministic. In other words, a large number of ciphertexts will decrypt to a given plaintext, and the particular ciphertext used in any given encryption is randomly chosen.

$$C_1 = E_K(M),\ C_2 = E_K(M),\ C_3 = E_K(M),\ \ldots,\ C_i = E_K(M)$$
$$M = D_K(C_1) = D_K(C_2) = D_K(C_3) = \ldots = D_K(C_i)$$

With probabilistic encryption, a cryptanalyst can no longer encrypt random plaintexts looking for the correct ciphertext. To illustrate, assume the cryptanalyst has ciphertext $C_i = E_K(M)$. Even if he guesses M correctly, when he encrypts $E_K(M)$, the result will be a completely different C: C_j. He cannot compare C_i and C_j, and so cannot know that he has guessed the message correctly.

This is amazingly cool stuff. Even if a cryptanalyst has the public encryption key, the plaintext, and the ciphertext, he cannot prove that the ciphertext is the encryption of the plaintext without the private decryption key. Even if he tries exhaustive search, he can only prove that every conceivable plaintext is a possible plaintext.

Under this scheme, the ciphertext will always be larger than the plaintext. You can't get around this; it's a result of the fact that many ciphertexts decrypt to the same plaintexts. The first probabilistic encryption scheme [625] resulted in a ciphertext so much larger than the plaintext that it was unusable.

However, Manual Blum and Goldwasser have an efficient implementation of probabilistic encryption using the Blum Blum Shub (BBS) random-bit generator described in Section 17.9 [199].

The BBS generator is based on the theory of quadratic residues. In English, there are two primes, p and q, that are congruent to 3 modulo 4. That's the private key. Their product, $pq = n$, is the public key. (Mind your ps and qs; the security of this scheme rests in the difficulty of factoring n.)

To encrypt a message, M, first choose some random x, relatively prime to n. Then compute

$$x_0 = x^2 \bmod n$$

Use x_0 as the seed of the BBS pseudo-random-bit generator and use the output of the generator as a stream cipher. XOR M, one bit at a time, with the output of the generator. The generator spits out bits b_i (the least-significant bit of x_i, where $x_i = x_{i-1}^2 \bmod n$), so

$$M = M_1, M_2, M_3, \ldots, M_t$$
$$C = M_1 \oplus b_1, M_2 \oplus b_2, M_3 \oplus b_3, \ldots, M_t \oplus b_t$$

where t is the length of the plaintext

Append the last computed value, x_t, to the end of the message and you're done.

The only way to decrypt this message is to recover x_0 and then set up the same BBS generator to XOR with the ciphertext. Because the BBS generator is secure to the left, the value x_t is of no use to the cryptanalyst. Only someone who knows p and q can decrypt the message.

In C, the algorithm to recover x_0 from x_t is:

```
int x0 (int p, int q, int n, int t, int xt)
{
     int a, b, u, v, w, z;

     /* we already know that gcd(p, q) == 1 */
     (void)extended_euclidian(p, q, &a, &b);
     u = modexp ((p+1)/4, t, p-1);
     v = modexp ((q+1)/4, t, q-1);
     w = modexp (xt%p, u, p);
     z = modexp (xt%q, v, q);
     return (b*q*w + a*p*z) % n;
}
```

Once you have x_0, decryption is easy. Just set up the BBS generator and XOR the output with the ciphertext.

You can make this scheme go even faster by using all the known secure bits of x_i, not just the least significant bit. With this improvement, Blum-Goldwasser probabilistic encryption is faster than RSA while leaking no partial information about the plaintext. You can also prove that the difficulty of breaking this scheme is the same as the difficulty of factoring n.

On the other hand, this scheme is totally insecure against a chosen-ciphertext attack. From the least significant bits of the right quadratic residues, it is possible to calculate the square root of any quadratic residue. If you can do this, then you can factor. For details, consult [1570,1571,35,36].

23.16 QUANTUM CRYPTOGRAPHY

Quantum cryptography taps the natural uncertainty of the quantum world. With it, you can create a communications channel where it is impossible to eavesdrop without disturbing the transmission. The laws of physics secure this quantum channel: even if the eavesdropper can do whatever he wants, even if the eavesdropper has

unlimited computing power, even if **P** = **NP**. Charles Bennett, Gilles Brassard, Claude Crépeau, and others have expanded on this idea, describing quantum key distribution, quantum coin flipping, quantum bit commitment, quantum oblivious transfer, and quantum secure multiparty computation. Their work is described in [128,129,123,124,125,133,126,394,134,392,243,517,132,130,244,393,396]. The best overview of quantum cryptography can be found in [131]; [1651] is another good nontechnical overview. A complete bibliography of quantum cryptography is [237].

This would still be on the lunatic fringe of cryptography, but Bennett and Brassard actually went and built a working model of the thing [127,121,122]. Now we have *experimental* quantum cryptography.

So sit back, get yourself something to drink, and relax. I'm going to explain what this is all about.

According to quantum mechanics, particles don't actually exist in any single place. They exist in several places at once, with probabilities of being in different places if someone looks. However, it isn't until a scientist comes along and measures the particle that it "collapses" into a single location. But you can't measure every aspect (for example, position and velocity) of a particle at the same time. If you measure one of those two quantities, the very act of measuring it destroys any possibility of measuring the other quantity. The quantum world has a fundamental uncertainty and there's no way to avoid it.

That uncertainty can be used to generate a secret key. As they travel, photons vibrate in some direction; up and down, left to right, or more likely at some angle. Normal sunlight is unpolarized; the photons vibrate every which way. When a large group of photons vibrate in the same direction they are **polarized**. Polarization filters allow only photons that are polarized in a certain direction through; the rest are blocked. For example, a horizontal polarization filter only allows horizontally polarized photons through. Turn that filter 90 degrees, and only vertically polarized photons can come through.

Let's say you have a pulse of horizontally polarized photons. If they try to pass through a horizontally polarized filter, they all get through. Slowly turn that filter 90 degrees; the number of photons getting through gets smaller and smaller, until none get through. This is counterintuitive. You'd think that turning the filter just a little will block all the photons, since the photons are horizontally polarized. But in quantum mechanics, each particle has a probability of suddenly switching its polarization to match the filter. If the angle is a little bit off, it has a high probability. If the angle is 90 degrees off, it has zero probability. And if the angle is 45 degrees off, it has a 50 percent probability of passing through the filter.

Polarization can be measured in any **basis**: two directions at right angles. An example basis is rectilinear: horizontal and vertical. Another is diagonal: left-diagonal and right-diagonal. If a photon pulse is polarized in a given basis and you measure it in the same basis, you learn the polarization. If you measure it in the wrong basis, you get a random result. We're going to use this property to generate a secret key:

(1) Alice sends Bob a string of photon pulses. Each of the pulses is randomly polarized in one of four directions: horizontal, vertical, left-diagonal, and right-diagonal.

For example, Alice sends Bob:

| | / — — \ — | — /

(2) Bob has a polarization detector. He can set his detector to measure rectilinear polarization or he can set his detector to measure diagonal polarization. He can't do both; quantum mechanics won't let him. Measuring one destroys any possibility of measuring the other. So, he sets his detectors at random, for example:

× + + × × × + × + +

Now, when Bob sets his detector correctly, he will record the correct polarization. If he sets his detector to measure rectilinear polarization and the pulse is polarized rectilinearly, he will learn which way Alice polarized the photon. If he sets his detector to measure diagonal polarization and the pulse is polarized rectilinearly, he will get a random measurement. He won't know the difference. In this example, he might get the result:

/ | — \ / \ — / — |

(3) Bob tells Alice, over an insecure channel, what settings he used.

(4) Alice tells Bob which settings were correct. In our example, the detector was correctly set for pulses 2, 6, 7, and 9.

(5) Alice and Bob keep only those polarizations that were correctly measured. In our example, they keep:

* | * * * \ — * — *

Using a prearranged code, Alice and Bob each translate those polarization measurements into bits. For example, horizontal and left-diagonal might equal one, and vertical and right-diagonal might equal zero. In our example, they both have:

0 0 1 1

So, Alice and Bob have generated four bits. They can generate as many as they like using this system. On the average, Bob will guess the correct setting 50 percent of the time, so Alice has to send $2n$ photon pulses to generate n bits. They can use these bits as a secret key for a symmetric algorithm or they can guarantee perfect secrecy and generate enough bits for a one-time pad.

The really cool thing is that Eve cannot eavesdrop. Just like Bob, she has to guess which type of polarization to measure; and like Bob, half of her guesses will be wrong. Since wrong guesses change the polarization of the photons, she can't help introducing errors in the pulses as she eavesdrops. If she does, Alice and Bob will end up with different bit strings. So, Alice and Bob finish the protocol like this:

(6) Alice and Bob compare a few bits in their strings. If there are discrepancies, they know they are being bugged. If there are none, they discard the bits they used for comparison and use the rest.

Enhancements to this protocol allow Alice and Bob to use their bits even in the presence of Eve [133,134,192]. They could compare only the parity of subsets of the bits. Then, if no discrepancies are found, they only have to discard one bit of the subset. This detects eavesdropping with only a 50 percent probability, but if they do this with n different subsets Eve's probability of eavesdropping without detection is only 1 in 2^n.

There's no such thing as passive eavesdropping in the quantum world. If Eve tries to recover all the bits, she will necessarily disrupt the communications.

Bennett and Brassard built a working model of quantum key distribution and have exchanged secure bits on a laser table. The latest I heard, some folks at British Telecom were sending bits over a 10-kilometer fiber-optic link [276,1245,1533]. They figure 50 kilometers is feasible. The mind boggles.

PART IV

THE REAL WORLD

CHAPTER 24

Example Implementations

It's one thing to design protocols and algorithms, but another thing to field them in operational systems. In theory, theory and practice are the same; in practice they are different. Often ideas that look good on paper don't work in real life. Maybe the bandwidth requirements are too large; maybe the protocol is too slow. Chapter 10 discusses some of the issues related to using cryptography; this chapter gives examples of how it has been done in practice.

24.1 IBM SECRET-KEY MANAGEMENT PROTOCOL

In the late 1970s IBM developed a complete key management system for communications and file security on a computer network, using only symmetric cryptography [515,1027]. This protocol is less important in the actual mechanisms and more in its overall philosophy: By automating the generation, distribution, installation, storage, changing, and destruction of keys, the protocol went a long way to ensure the security of the underlying cryptographic algorithms.

This protocol provides three things: secure communications between a server and several terminals, secure file storage at the server, and secure communication among servers. The protocol doesn't really provide for direct terminal-to-terminal communication, although it can be modified to do that.

Each server on the network is attached to a cryptographic facility, which does all of the encrypting and decrypting. Each server has a **Master Key**, KM_0, and two variants, KM_1 and KM_2, both of which are simple variants of KM_0. These keys are used to encrypt other keys and to generate new keys. Each terminal has a **Master Terminal Key**, *KMT*, which is used to exchange keys with other terminals.

The servers store *KMT*, encrypted with KM_1. All other keys, such as those used to encrypt files of keys (called *KNF*), are stored in encrypted form under KM_2. The master key, KM_0, is stored in some nonvolatile security module. Today that could be

either a ROM key or a magnetic card, or it could be typed in by the user (probably as a text string and then key crunched). KM_1 and KM_2 are not stored anywhere in the system, but are computed from KM_0 whenever they are needed. Session keys, for communication among servers, are generated with a pseudo-random process in the server. Keys to encrypt files for storage (*KNF*) are generated in the same manner.

The heart of the protocol is a tamper-resistant module, called a **cryptographic facility**. At both the server and the terminal, all encryption and decryption takes place within this facility. The most important keys, those used to generate the actual encryption keys, are stored in this module. These keys can never be read once they are stored. And they are tagged by use: A key dedicated for one purpose cannot accidentally be used for another. This concept of **key control vectors** is probably the most significant contribution of this system. Donald Davies and William Price discuss this key management protocol in detail [435].

A Variation

A variation on this scheme of master and session keys can be found in [1478]. It's built around network nodes with key notarization facilities that serve local terminals. It is designed to:

- Secure two-way communication between any two terminal users.
- Secure communications using encrypted mail.
- Provide personal file protection.
- Provide a digital signature capability.

For communication and file transfer among users, the scheme uses keys generated in the key notarization facility and sent to the users encrypted under a master key. The identities of the users are incorporated with the key, to provide evidence that the session key has been used between a particular pair of users. This **key notarization** feature is central to the system. Although the system does not use public-key cryptography, it has a digital-signature-like capability: A key could have only come from a particular source and could only be read at a particular destination.

24.2 MITRENET

One of the earliest implementations of public-key cryptography was the experimental system MEMO (MITRE Encrypted Mail Office). MITRE is a DoD contractor, a government think tank, and an all-around bunch of smart guys. MEMO was a secure electronic mail system for users in the MITRENET network, using public-key cryptography for key exchange and DES for file encryption.

In the MEMO system, all public keys are stored in a Public Key Distribution Center, which is a separate node on the network. They are stored in an EPROM to prevent anyone from changing them. Private keys are generated by users or by the system.

For a user to send secure messages, the system first establishes a secure communications path with the Public Key Distribution Center. The user requests a file of all public keys from the Center. If the user passes an identification test using his private key, the Center sends this list to the user's workstation. The list is encrypted using DES to ensure file integrity.

The implementation uses DES to encrypt messages. The system generates a random DES key for file encryption; the user encrypts the file with the DES key and encrypts the DES key with the recipient's public key. Both the DES-encrypted file and the public-key-encrypted key are sent to the recipient.

MEMO makes no provision for lost keys. There is some provision for integrity checking of the messages, using checksums. No authentication is built into the system.

The particular public-key implementation used for this system—Diffie-Hellman key exchange over GF(2^{127})—was proven insecure before the system was implemented (see Section 11.6), although it is easy to modify the system to use larger numbers. MEMO was intended mainly for experimental purposes and was never made operational on the real MITRENET system.

24.3 ISDN

Bell-Northern Research developed a prototype secure Integrated Services Digital Network (ISDN) telephone terminal [499,1192,493,500]. As a telephone, it was never developed beyond prototype. The resulting product was the Packet Data Security Overlay. The terminal uses Diffie-Hellman key exchange, RSA digital signatures, and DES data encryption; it can transmit and receive voice and data at 64 kilobits per second.

Keys

A long-term public-key/private-key key pair is embedded in the phone. The private key is stored in a tamper-resistant area of the phone. The public key serves as the identification of the phone. These keys are part of the phone itself and cannot be altered in any way.

Additionally, two other public keys are stored in the phone. One of these keys is the owner's public key. This key is used to authenticate commands from the owner and can be changed via a command signed by the owner. In this way an owner can transfer ownership of the phone to someone else.

The public key of the network is also stored in the phone. This key is used to authenticate commands from the network's key management facility and to authenticate calls from other users on the network. This key can also be changed via a signed command from the owner. This permits the owner to move his phone from one network to another.

These keys are considered long-term keys: rarely, if ever, changed. A short-term public-key/private-key key pair is also stored on the phone. These are encapsulated

in a certificate signed by the key management facility. When two phones set up a call, they exchange certificates. The public key of the network authenticates these certificates.

This exchange and verification of certificates only sets up a secure call from phone to phone. To set up a secure call from person to person, the protocol has an additional piece. The owner's private key is stored on a hardware **ignition key**, which is inserted into the telephone by the owner. This ignition key contains the owner's private key, encrypted under a secret password known only by the owner (not by the phone, not by the network's key management facility, not by anybody). It also contains a certificate signed by the network's key management facility that contains the owner's public key and some identifying information (name, company, job title, security clearance, favorite pizza toppings, sexual preference, or whatever). This is also encrypted. To decrypt this information and enter it into the phone, the owner types his secret password on the phone's keypad. After the phone uses this information to set up calls, it is erased after the owner removes his ignition key.

The phone also stores a set of certificates from the network's key management facility. These certificates authorize particular users to use particular phones.

Calling

A call from Alice to Bob works as follows.

(1) Alice inserts her ignition key into the phone and enters her password.

(2) The phone interrogates the ignition key to determine Alice's identity and gives Alice a dial tone.

(3) The phone checks its set of certificates to ensure that Alice is authorized to use the particular phone.

(4) Alice dials the number; the phone places the call.

(5) The two telephones use a public-key cryptography key-exchange protocol to generate a unique and random session key. All subsequent protocol steps are encrypted using this key.

(6) Alice's phone transmits its certificate and user authentication.

(7) Bob's phone authenticates the signatures on both the certificate and the user authentication using the network's public key.

(8) Bob's phone initiates a challenge-and-reply sequence. It demands real-time signed responses to time-dependent challenges. (This prevents an adversary from using certificates copied from a previous exchange.) One response must be signed by Alice's phone's private key; another must be signed by Alice's private key.

(9) Bob's phone rings, unless he is already on the phone.

(10) If Bob is home, he inserts his ignition key into the phone. His phone interrogates the ignition key and checks Bob's certificate as in steps (2) and (3).

(11) Bob transmits his certificate and user authentication.

(12) Alice's phone authenticates Bob's signatures as in step (7), and initiates a challenge-and-reply sequence as in step (8).

(13) Both phones display the identity of the other user and phone on their displays.

(14) The secure conversation begins.

(15) When one party hangs up, the session key is deleted, as are the certificates Bob's phone received from Alice's phone and the certificates Alice's phone received from Bob's phone.

Each DES key is unique to each call. It exists only inside the two phones for the duration of the call and is destroyed immediately afterward. If an adversary captures one or both of the phones involved in the call, he will not be able to decrypt any previous call between the two phones.

24.4 STU-III

STU stands for "Secure Telephone Unit," an NSA-designed secure phone. The unit is about the size and shape of a conventional telephone, and can be used as such. The phones are also tamper-resistant, enough so that they are unclassified if unkeyed. They also have a data port and can be used to secure modem traffic as well as voice [1133].

Whitfield Diffie described the STU-III in [494]:

> To make a call with a STU-III, the caller first places an ordinary call to another STU-III, then inserts a key-shaped device containing a cryptographic variable and pushes a "go secure" button. After an approximately 15-second wait for cryptographic setup, each phone shows information about the identity and clearance of the other party on its display and the call can proceed.
>
> In an unprecedented move, Walter Deeley, NSA's deputy director for communications security, announced the STU-III or Future Secure Voice System in an exclusive interview given to *The New York Times* [282]. The objective of the new system was primarily to provide secure voice and low-speed data communications for the U.S. Defense Department and its contractors. The interview didn't say much about how it was going to work, but gradually the word began to leak out. The new system was using public key.
>
> The new approach to key management was reported early on [68] and one article spoke of phones being "reprogrammed once a year by secure telephone link," a turn of phrase strongly suggestive of a certificate passing protocol, similar to that described [in Section 24.3], that minimizes the need for phones to talk to the key management center. Recent reports have been more forthcoming, speaking of a key management system called FIREFLY that [1341] "evolved from public key

technology and is used to establish pair-wise traffic encryption keys." Both this description and testimony submitted to the U.S. Congress by Lee Neuwirth of Cylink [1164] suggest a combination of key exchange and certificates similar to that used in the ISDN secure phone and it is plausible that FIREFLY too is based on exponentiation.

STU-IIIs are manufactured by AT&T and GE. Somewhere between 300,000 and 400,000 have been fielded through 1994. A new version, the Secure Terminal Equipment (STE), will work on ISDN lines.

24.5 KERBEROS

Kerberos is a trusted third-party authentication protocol designed for TCP/IP networks. A Kerberos service, sitting on the network, acts as a trusted arbitrator. Kerberos provides secure network authentication, allowing a person to access different machines on the network. Kerberos is based on symmetric cryptography (DES as implemented, but other algorithms could be used instead). Kerberos shares a different secret key with every entity on the network and knowledge of that secret key equals proof of identity.

Kerberos was originally developed at MIT for Project Athena. The Kerberos model is based on Needham-Schroeder's trusted third-party protocol (see Section 3.3) [1159]. The original version of Kerberos, Version 4, is specified in [1094,1499]. (Versions 1 through 3 were internal development versions.) Version 5, modified from Version 4, is specified in [876,877,878]. The best overview of Kerberos is [1163]. Other survey articles are [1384,1493], and two good articles on using Kerberos in the real world are [781,782].

The Kerberos Model

The basic Kerberos protocol was outlined in Section 3.3. In the Kerberos model, there are entities—clients and servers—sitting on the network. Clients can be users, but can also be independent software programs that need to do things: download files, send messages, access databases, access printers, obtain administrative privileges, whatever.

Kerberos keeps a database of clients and their secret keys. For a human user, the secret key is an encrypted password. Network services requiring authentication, as well as clients who wish to use these services, register their secret key with Kerberos.

Because Kerberos knows everyone's secret key, it can create messages that convince one entity of another entity's identity. Kerberos also creates session keys which are given to a client and a server (or to two clients) and no one else. A session key is used to encrypt messages between the two parties, after which it is destroyed.

Kerberos uses DES for encryption. Kerberos Version 4 provided a nonstandard mode for authentication. This mode is weak: It fails to detect certain changes to the ciphertext (see Section 9.10). Kerberos Version 5 uses CBC mode.

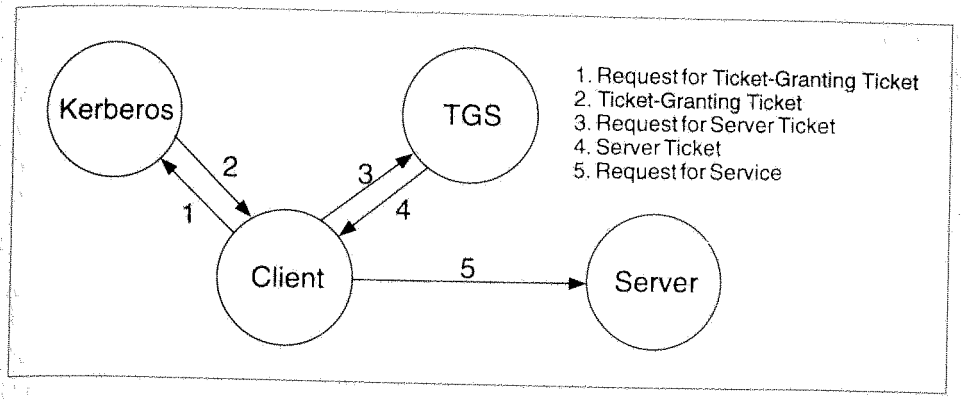


Figure 24.1 Kerberos authentication steps.

How Kerberos Works

This section discusses Kerberos Version 5. I will outline the differences between Version 4 and Version 5 further on. The Kerberos protocol is straightforward (see Figure 24.1). A client requests a ticket for a **Ticket-Granting Service** (**TGS**) from Kerberos. This ticket is sent to the client, encrypted in the client's secret key. To use a particular server, the client requests a ticket for that server from the TGS. Assuming everything is in order, the TGS sends the ticket back to the client. The client then presents this ticket to the server along with an authenticator. Again, if there's nothing wrong with the client's credentials, the server lets the client have access to the service.

Table 24.1
Kerberos Table of Abbreviations

c	= client
s	= server
a	= client's network address
v	= beginning and ending validity time for a ticket
t	= timestamp
K_x	= x's secret key
$K_{x,y}$	= session key for x and y
$\{m\}K_x$	= m encrypted in x's secret key
$T_{x,y}$	= x's ticket to use y
$A_{x,y}$	= authenticator from x to y

Credentials

Kerberos uses two types of credentials: **tickets** and **authenticators**. (The rest of this section uses the notation used in Kerberos documents—see Table 24.1.) A ticket is used to pass securely to the server the identity of the client for whom the ticket was issued. It also contains information that the server can use to ensure that the client using the ticket is the same client to whom the ticket was issued. An authenticator is an additional credential, presented with the ticket.

A Kerberos ticket takes this form:

$$T_{c,\,s} = s,\ \{c,\ a,\ v,\ K_{c,\,s}\}K_s$$

A ticket is good for a single server and a single client. It contains the client's name and network address, the server's name, a timestamp, and a session key. This information is encrypted with the server's secret key. Once the client gets this ticket, she can use it multiple times to access the server—until the ticket expires. The client cannot decrypt the ticket (she does not know the server's secret key), but she can present it to the server in its encrypted form. No one listening on the network can read or modify the ticket as it passes through the network.

A Kerberos authenticator takes this form:

$$A_{c,\,s} = \{c,\ t,\ key\}K_{c,\,s}$$

The client generates it every time she wishes to use a service on the server. The authenticator contains the client's name, a timestamp, and an optional additional session key, all encrypted with the session key shared between the client and the server. Unlike a ticket, it can only be used once. However, since the client can generate authenticators as needed (it knows the shared secret key), this is not a problem.

The authenticator serves two purposes. First, it contains some plaintext encrypted in the session key. This proves that it also knows the key. Just as important, the sealed plaintext includes the timestamp. An eavesdropper who records both the ticket and the authenticator can't replay them two days later.

Kerberos Version 5 Messages

Kerberos Version 5 has five messages (see Figure 24.1):

1. Client to Kerberos: $c,\ tgs$
2. Kerberos to client: $\{K_{c,\,tgs}\}K_c,\ \{T_{c,\,tgs}\}K_{tgs}$
3. Client to TGS: $\{A_{c,\,s}\}K_{c,\,tgs},\ \{T_{c,\,tgs}\}K_{tgs,\,s}$
4. TGS to client: $\{K_{c,\,s}\}K_{c,\,tgs},\ \{T_{c,\,s}\}K_s$
5. Client to server: $\{A_{c,\,s}\}K_{c,\,s},\ \{T_{c,\,s}\}K_s$

These will now be discussed in detail.

Getting an Initial Ticket

The client has one piece of information that proves her identity: her password. Obviously we don't want her to send this password over the network. The Kerberos protocol minimizes the chance that this password will be compromised, while at the same time not allowing a user to properly authenticate herself unless she knows the password.

The client sends a message containing her name and the name of her TGS server to the Kerberos authentication server. (There can be many TGS servers.) In reality, the user probably just enters her name into the system and the login program sends the request.

The Kerberos authentication server looks up the client in his database. If the client is in the database, Kerberos generates a session key to be used between her and the TGS. This is called a **Ticket Granting Ticket** (**TGT**). Kerberos encrypts that session key with the client's secret key. Then it creates a TGT for the client to authenticate herself to the TGS, and encrypts that in the TGS's secret key. The authentication server sends both of these encrypted messages back to the client.

The client now decrypts the first message and retrieves the session key. The secret key is a one-way hash of her password, so a legitimate user will have no trouble doing this. If the user were an imposter, he would not know the correct password and therefore could not decrypt the response from the Kerberos authentication server. Access would be denied and he wouldn't be able to get the ticket or the session key.

The client saves the TGT and session key and erases the password and the one-way hash. This information is erased to reduce the chance of compromise. If an adversary manages to copy the client's memory, he will only get the TGT and the session key. These are valuable pieces of information, but only during the lifetime of the TGT. After the TGT expires, they will be worthless.

The client can now prove her identity to the TGS for the lifetime of the TGT.

Getting Server Tickets

A client has to obtain a separate ticket for each service she wants to use. The TGS grants tickets for individual servers.

When a client needs a ticket that she does not already have, she sends a request to the TGS. (In reality, the program would do this automatically, and it would be invisible to the user.)

The TGS, upon receiving the request, decrypts the TGT with his secret key. Then he uses the session key included in the TGT to decrypt the authenticator. Finally, he compares the information in the authenticator with the information in the ticket, the client's network address with the address the request was sent from, and the timestamp with the current time. If everything matches, he allows the request to proceed.

Checking timestamps assumes that all machines have synchronized clocks, at least to within several minutes. If the time in the request is too far in the future or

the past, the TGS treats the request as an attempt to replay a previous request. The TGS should also keep track of all live authenticators, because past requests can have timestamps that are still valid. Another request with the same ticket and timestamp as one already received can be ignored.

The TGS responds to a valid request by returning a valid ticket for the client to present to the server. The TGS also creates a new session key for the client and the server, encrypted with the session key shared by the client and the TGS. Both of these messages are then sent back to the client. The client decrypts the message and extracts the session key.

Requesting a Service

Now the client is ready to authenticate herself to the server. She creates a message very similar to the one sent to the TGS (which makes sense, since the TGS is a service).

The client creates an authenticator, consisting of her name and network address, and a timestamp, encrypted with the session key for her and the server that the TGS generated. The request consists of the ticket received from Kerberos (already encrypted with the server's secret key) and the encrypted authenticator.

The server decrypts and checks the ticket and the authenticator, as discussed previously, and also checks the client's address and the timestamp. If everything checks out, the server knows that, according to Kerberos, the client is who she says she is.

For applications that require mutual authentication, the server sends the client back a message consisting of the timestamp, encrypted with the session key. This proves that the server knew his secret key and could decrypt the ticket and therefore the authenticator.

The client and the server can encrypt future messages with the shared key, if desired. Since only they share this key, they both can assume that a recent message encrypted in that key originated with the other party.

Kerberos Version 4

The previous sections discussed Kerberos Version 5. In the messages and the construction of the tickets and authenticators, Version 4 is slightly different.

In Kerberos Version 4, the five messages looked like:

1. Client to Kerberos: c, tgs
2. Kerberos to client: $\{K_{c,\,tgs}, \{T_{c,\,tgs}\}K_{tgs}\}K_c$
3. Client to TGS: $\{A_{c,\,s}\}K_{c,\,tgs}, \{T_{c,\,tgs}\}K_{tgs}, s$
4. TGS to client: $\{K_{c,\,s}, \{T_{c,\,s}\}K_s\}K_{c,\,tgs}$
5. Client to server: $\{A_{c,\,s}\}K_{c,\,s}, \{T_{c,\,s}\}K_s$

$$T_{c,\,s} = \{s, c, a, v, l, K_{c,\,s}\}K_s$$
$$A_{c,\,s} = \{c, a, t\}K_{c,\,s}$$

Messages 1, 3, and 5 are identical. The double encryption of the ticket in steps 2 and 4 has been removed in Version 5. The Version 5 ticket adds the possibility of multiple addresses, and it replaces a "lifetime" field, *l*, with a beginning and ending time. The Version 5 authenticator adds the option of including an additional key.

Security of Kerberos

Steve Bellovin and Michael Merritt discussed several potential security vulnerabilities of Kerberos [108]. Although this paper was written about the Version 4 protocols, many of their comments also apply to Version 5.

It may be possible to cache and replay old authenticators. Although timestamps are supposed to prevent this, replays can be done during the lifetime of the ticket. Servers are supposed to store all valid tickets to prevent replays, but this is not always possible. And ticket lifetimes can be long; eight hours is typical.

Authenticators rely on the fact that all the clocks in the network are more or less synchronized. If a host can be fooled about the correct time, then an old authenticator can be replayed without any problem. Most network time protocols are insecure, so this can be a serious problem.

Kerberos is also vulnerable to password-guessing attacks. An intruder can collect tickets and then try to decrypt them. Remember that the average user doesn't usually choose good passwords. If Mallory collects enough tickets, his chances of recovering a password are good.

Perhaps the most serious attack involves malicious software. The Kerberos protocols rely on the fact that the Kerberos software is trustworthy. There's nothing to stop Mallory from surreptitiously replacing all client Kerberos software with a version that, in addition to completing the Kerberos protocols, records passwords. This is a problem with any cryptographic software package on an insecure computer, but the widespread use of Kerberos in these environments makes it a particularly tempting target.

Enhancements to Kerberos are in the works, including an implementation of public-key cryptography and a smart-card interface for key management.

Licenses

Kerberos is not in the public domain, but MIT's code is freely available. Actually implementing it into a working UNIX environment is another story. Several companies sell versions of Kerberos, but you can get a good version free from Cygnus Support, 814 University Ave., Palo Alto, CA, 94301; (415) 322-3811; fax: (415) 322-3270.

24.6 KryptoKnight

KryptoKnight (Kryptonite—get it?) is an authentication and key distribution system designed by IBM. It is a secret-key protocol and uses either DES in CBC mode (see Section 9.3) or a modified version of MD5 (see Section 18.5).

KryptoKnight supports four security services:

- User authentication (called single sign-on)
- Two-party authentication
- Key distribution
- Authentication of data origin and content

From a user's perspective, KryptoKnight is similar to Kerberos. Some differences are:

- KryptoKnight uses a hash function for authentication and encrypting tickets.
- KryptoKnight does not rely on synchronized clocks; it uses nonces for challenges (see Section 3.3).
- If Alice wishes to communicate with Bob, KryptoKnight has the option of allowing Alice to send a message to Bob and then for Bob to initiate the key exchange protocol.

KryptoKnight has tickets and authenticators, just like Kerberos. It has TGSs, but KryptoKnight calls them authentication servers. KryptoKnight's designers spent considerable effort minimizing the number of messages, lengths of messages, and amount of encryption. For further information on KryptoKnight, read [1110,173,174,175].

24.7 SESAME

SESAME stands for Secure European System for Applications in a Multivendor Environment. It's a European Community security project, 50 percent funded by RACE (see Section 25.7), whose primary objective is producing technology for user authentication with distributed access control. Think of it as kind of a European version of Kerberos. It's a two-part project: Stage one is a basic prototype of the architecture, and stage two is a set of commercial projects. The three companies with the greatest hand in development are ICL in the United Kingdom, Siemens in Germany, and Bull in France.

SESAME is an authentication and key-exchange system [361,1248,797,1043]. It uses the Needham-Schroeder protocol, with public-key cryptography to communicate between different security domains. The system is seriously flawed in several respects. Instead of using a real encryption algorithm, they use XOR with a 64-bit key size. Even worse, they use XOR in CBC mode, which leaves half the plaintext unencrypted. In their defense, they planned on using DES until the French government complained; they validated the code with DES but then removed it, and expect people to add it back. I am unimpressed nonetheless.

Authentication in SESAME is a function on the first block of a message, not on the entire message. This has the effect of authenticating "Dear Sir" and not the body of a letter. Key generation consists of two calls to the UNIX rand function, which isn't very random. SESAME uses crc32 and MD5 as one-way hash functions. And of course, SESAME is vulnerable to Kerberos-like password-guessing.

24.8 IBM Common Cryptographic Architecture

The Common Cryptographic Architecture (CCA) was designed and developed by IBM to provide cryptographic primitives for confidentiality, integrity, key management, and personal identification number (PIN) processing [751,784,1025,1026, 940,752]. Keys are managed by control vectors (CVs) (see Section 8.5). Every key has a CV XORed with it and is never separated from the vector unless inside secure hardware. The CV is a data structure providing an intuitive understanding of the privileges associated with a particular key.

The individual bits of the CV are defined to have specific meanings for using and handling each key managed by CCA. The CV is carried with the encrypted key in data structures called key tokens. Internal key tokens are used locally and contain keys encrypted under the local master key (MK). External key tokens are used to export and import encrypted keys between systems. Keys in external key tokens are encrypted under key-encrypting keys (KEK). The KEKs are managed in internal key tokens. Keys are separated according to their permitted uses.

Key length is also specified and enforced using bits in the CV. Single length keys are 56 bits and are used for such functions as privacy and message authentication. Double length keys are 112 bits and are used for key management, PIN functions, and other special uses. Keys can be required to be DOUBLE-ONLY in which both the left and right halves of the key must be different, DOUBLE in which the halves are permitted to be equal by chance, SINGLE-REPLICATED in which the left and right halves are equal, or SINGLE which contains only 56 bits. The CCA functions specify hardware enforcement of certain key types to be used for some operations.

The CV is checked in a secure hardware processor: It must conform to the permitted CCA rules for each CCA function. If the CV successfully passes the test requirements, a variant of the KEK or MK is obtained by the XOR of the KEK or MK with the CV, and the plaintext target key is recovered for use internally with the CCA function. When new keys are generated, the CV specifies the uses of the generated key. Those combinations of key types that could be used in attacking the system are not generated or imported into a CCA-compliant system.

CCA uses a combination of public-key cryptography and secret-key cryptography for key distribution. The KDC shares a secret master key with each user and encrypts session keys using that master key. Master keys are distributed using public-key cryptography.

The system's designers chose this hybrid approach for two reasons. The first is performance. Public-key cryptography is computationally intensive; if session keys are distributed using public-key cryptography, the system might bog down. The second is backwards compatibility; this system can be overlaid on existing secret-key schemes with minimal disruption.

CCA systems are designed to be interoperable. For systems that are non-CCA compliant, a Control Vector Translate (CVXLT) function permits keys to be passed between the two implementations. Initialization of the CVXLT function requires dual control. Two individuals must set up the required translation tables indepen-

dently. Such dual control provides a high degree of assurance concerning the integrity and pedigree of any keys introduced into the system.

A key of type DATA is provided for compatibility with other systems. A DATA key is stored with a CV that identifies the key as a DATA key. DATA keys can have broad uses and as such must be regarded with suspicion and used with care. DATA keys may not be used for any key management functions.

The Commercial Data Masking Facility (CDMF) provides an exportable version of CCA. It has a special feature that reduces DES keys to an effective 40 bits for export (see Section 15.5) [785].

24.9 ISO AUTHENTICATION FRAMEWORK

Public-key cryptography has been recommended for use with the ISO authentication framework, also known as the X.509 protocols [304]. This framework provides for authentication across networks. Although no particular algorithms are specified for either security or authentication, the specification recommends RSA. There are

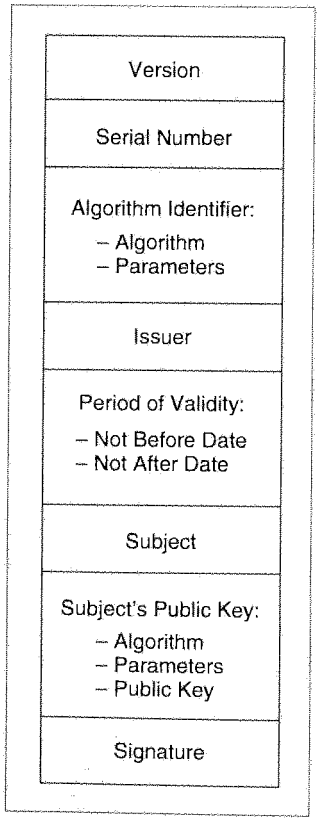


Figure 24.2 An X.509 certificate.

provisions, however, for multiple algorithms and hash functions. X.509 was initially issued in 1988. After public review and comment, it was revised in 1993 to correct some security problems [1100,750].

Certificates

The most important part of X.509 is its structure for public-key certificates. Each user has a distinct name. A trusted Certification Authority (CA) assigns a unique name to each user and issues a signed certificate containing the name and the user's public key. Figure 24.2 shows an X.509 certificate [304].

The version field identifies the certificate format. The serial number is unique within the CA. The next field identifies the algorithm used to sign the certificate, together with any necessary parameters. Issuer is the name of the CA. The period of validity is a pair of dates; the certificate is valid during the time period between the two. Subject is the name of the user. The subject's public key information includes the algorithm name, any necessary parameters, and the public key. The last field is the CA's signature.

If Alice wants to communicate with Bob, she first gets his certificate from a database. Then she verifies its authenticity. If both share the same CA, this is easy. Alice simply verifies the CA's signature on Bob's certificate.

If they use different CAs, it's more complicated. Think of a tree structure, with different CAs certifying other CAs and users. On the top is one master CA. Each CA has a certificate signed by the CA above it, and by the CAs below it. Alice uses these certificates to verify Bob's certificate.

Figure 24.3 illustrates this. Alice's certificate is certified by CA_A; Bob's is certified by CA_B. Alice knows CA_A's public key. CA_C has a certificate signed by CA_A, so Alice

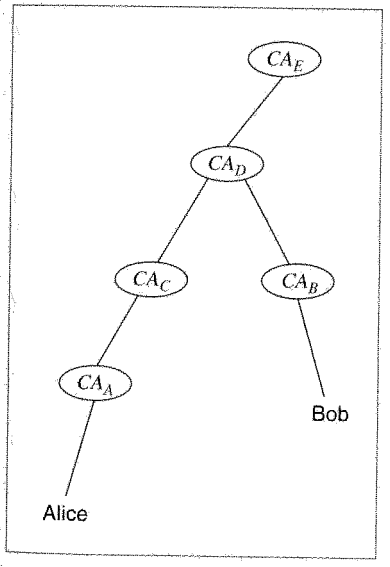


Figure 24.3 Sample certification hierarchy.

can verify that. CA_D has a certificate signed by CA_C. CA_B has a certificate signed by CA_D. And Bob's certificate is signed by CA_B. By moving up the certification tree to a common point, in this case CA_D, and then down to Bob, Alice can verify Bob's certificate.

Certificates can be stored on databases around the network. Users can send them to each other. When a certificate expires, it should be removed from any public directories. The issuing CA, however, should maintain a copy of the certificate. Should a dispute arise later, it will be required.

Certificates can also be revoked, either because the user's key has been compromised, the CA's key has been compromised, or because the CA no longer wants to certify the user. Each CA must maintain a list of all revoked but not expired certificates. When Alice receives a new certificate, she should check to see if it has been revoked. She can check a database of revoked keys on the network, but more likely she will check a locally cached list of revoked certificates. There are certainly possible abuses to this system; key revocation is probably its weakest part.

Authentication Protocols

Alice wants to communicate with Bob. First she goes to a database and obtains what is called a **certification path** from Alice to Bob, and Bob's public key. At this point Alice can initiate either a one-way, two-way, or three-way authentication protocol.

The one-way protocol is a single communication from Alice to Bob. It establishes the identities of both Alice and Bob and the integrity of any information communicated by Alice to Bob. It also prevents any replay attacks on the communication.

The two-way protocol adds a reply from Bob. It establishes that Bob, and not an imposter, sent the reply. It also establishes the secrecy of both communications and prevents replay attacks.

Both the one-way and two-way protocols use timestamps. A three-way protocol adds another message from Alice to Bob and obviates the need for timestamps (and therefore authenticated time).

The one-way protocol is:

(1) Alice generates a random number, R_A.

(2) Alice constructs a message, $M = (T_A, R_A, I_B, d)$, where T_A is Alice's timestamp, I_B is Bob's identity, and d is an arbitrary piece of data. The data may be encrypted with Bob's public key, E_B, for security.

(3) Alice sends $(C_A, D_A(M))$ to Bob. (C_A is Alice's certificate; D_A is the common point on the certification tree.)

(4) Bob verifies C_A and obtains E_A. He makes sure these keys have not expired. (E_A is Alice's public key.)

(5) Bob uses E_A to decrypt $D_A(M)$. This verifies both Alice's signature and the integrity of the signed information.

(6) Bob checks the I_B in M for accuracy.

(7) Bob checks the T_A in M and confirms that the message is current.

(8) As an option, Bob can check R_A in M against a database of old random numbers to ensure the message is not an old one being replayed.

The two-way protocol consists of the one-way protocol and then a similar one-way protocol from Bob to Alice. After executing steps (1) through (8) of the one-way protocol, the two-way protocol continues with:

(9) Bob generates another random number, R_B.

(10) Bob constructs a message $M' = (T_B, R_B, I_A, R_A, d)$, where T_B is Bob's timestamp, I_A is the identity of Alice and d is arbitrary data. The data may be encrypted with Alice's public key, E_A, for security. R_A is the random number Alice generated in step (1).

(11) Bob sends $D_B(M')$ to Alice.

(12) Alice uses E_B to decrypt $D_B(M')$. This verifies both Bob's signature and the integrity of the signed information.

(13) Alice checks the I_A in M' for accuracy.

(14) Alice checks the T_B in M' and confirms that the message is current.

(15) As an option, Alice can check the R_B in M' to ensure the message is not an old one being replayed.

The three-way protocol accomplishes the same thing as the two-way protocol, but without timestamps. Steps (1) through (15) are identical to the two-way protocol, with $T_A = T_B = 0$.

(16) Alice checks the received version of R_A against the R_A she sent to Bob in step (3).

(17) Alice sends $D_A(R_B)$ to Bob.

(18) Bob uses E_A to decrypt $D_A(R_B)$. This verifies both Alice's signature and the integrity of the signed information.

(19) Bob checks the received version of R_B against the R_B he sent to Alice in step (10).

24.10 Privacy-Enhanced Mail (PEM)

PEM is the Internet Privacy-Enhanced Mail standard, adopted by the Internet Architecture Board (IAB) to provide secure electronic mail over the Internet. It was initially designed by the Internet Resources Task Force (IRTF) Privacy and Security Research Group (PSRG), and then handed over to the Internet Engineering Task Force (IETF) PEM Working Group. The PEM protocols provide for encryption, authentication, message integrity, and key management.

The complete PEM protocols were initially detailed in a series of RFCs (Requests for Comment) in [977] and then revised in [978]. The third iteration of the protocols [979,827,980] is summarized in [177,178]. The protocols were modified and improved, and the final protocols are detailed in another series of RFCs [981,825,76,802]. Another paper by Matthew Bishop [179] details the changes. Reports of attempts to implement PEM include [602,1505,1522,74,351,1366, 1367]. See also [1394].

PEM is an inclusive standard. The PEM procedures and protocols are intended to be compatible with a wide range of key-management approaches, including both symmetric and public-key schemes to encrypt data-encrypting keys. Symmetric cryptography is used for message-text encryption. Cryptographic hash algorithms are used for message integrity. Other documents support key-management mechanisms using public-key certificates; algorithms, modes, and associated identifiers; and paper and electronic format details and procedures for the key-management infrastructure to support these services.

PEM supports only certain algorithms, but allows for different suites of algorithms to be specified later. Messages are encrypted with DES in CBC mode. Authentication, provided by something called a **Message Integrity Check** (**MIC**), uses either MD2 or MD5. Symmetric key management can use either DES in ECB mode or triple-DES using two keys (called EDE mode). PEM also supports public-key certificates for key management, using the RSA algorithm (key length up to 1024 bits) and the X.509 standard for certificate structure.

PEM provides three privacy-enhancement services: confidentiality, authentication, and message integrity. No special processing requirements are imposed on the electronic mail system. PEM can be incorporated selectively, by site or by user, without affecting the rest of the network.

PEM Documents

The specifications for PEM come from four documents:

- RFC 1421: Part I, Message Encryption and Authentication Procedures. This document defines message encryption and authentication procedures in order to provide privacy-enhanced mail services for electronic mail transfer on the Internet.
- RFC 1422: Part II, Certificate-Based Key Management. This document defines a supporting key management architecture and infrastructure, based on public-key certificate techniques to provide keying information to message originators and recipients.
- RFC 1423: Part III, Algorithms, Modes, and Identifiers. This document provides definitions, formats, references, and citations for cryptographic algorithms, usage modes, and associated identifiers and parameters.
- RFC 1424: Part IV, Key Certification and Related Services. This document describes three types of service in support of PEM: key certification, certificate revocation list (CRL) storage, and CRL retrieval.

Certificates

PEM is compatible with the authentication framework described in [304]; see also [826]. PEM is a superset of X.509; it establishes procedures and conventions for a key-management infrastructure for use with PEM and with other protocols (from both the TCP/IP and OSI suites) in the future.

The key-management infrastructure establishes a single root for all Internet certification. The Internet Policy Registration Authority (IPRA) establishes global policies that apply to all certification under this hierarchy. Beneath the IPRA root are Policy Certification Authorities (PCAs), each of which establishes and publishes its policies for registering users or organizations. Each PCA is certified by the IPRA. Below PCAs, CAs certify users and subordinate organizational entities (such as departments, offices, subsidiaries). Initially, the majority of users are expected to be registered with some organization.

Some PCAs are expected to provide certification for users who wish to register independent of any organization. For users who wish anonymity while taking advantage of PEM privacy facilities, one or more PCAs are expected to be established with policies that allow for registration of users who do not wish to disclose their identities.

PEM Messages

PEM's heart is its message format. Figure 24.4 shows an encrypted message using symmetric key management, Figure 24.5 shows an authenticated and encrypted message using public-key key management, and Figure 24.6 shows an authenticated (but unencrypted) message using public-key key management.

The first field is "Proc-Type," and identifies the type of processing performed on the message. There are three possible types of messages. The "ENCRYPTED" spec-

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Proc-Type: 4,ENCRYPTED
Content-Domain: RFC822
DEK-Info: DES-CBC,F8143EDE5960C597
Originator-ID-Symmetric: schneier@counterpane.com,,
Recipient-ID-Symmetric: schneier@chinet.com,ptf-kmc,3
Key-Info:
DES-ECB,RSA-MD2,9FD3AAD2F2691B9A,B70665BB9BF7CBCDA60195DB94F727D3
Recipient-ID-Symmetric: pem-dev@tis.com,ptf-kmc,4
Key-Info:
DES-ECB,RSA-MD2,161A3F75DC82EF26,E2EF532C65CBCFF79F83A2658132DB47
LLrHB0eJzyhP+/fSStdW8okeEnv47jxe7SJ/iN72ohNcUk2jHEUSoH1nvNSIWL9M
8tEjmF/zxB+bATMtPjCUWbz8Lr9w1oXIkjHU1BLpvXROUrUzYbkNpk0agV2IzUpk
J6UiRRGcDSvzrsoK+oNvqu6z7Xs5Xfz5rDqUcM1K1Z672OdcBWGGsDLpTpSCnpot
dXd/H5LMDWnonNvPCwQUHt==
-----END PRIVACY-ENHANCED MESSAGE-----
```

Figure 24.4 Example of an encapsulated message (symmetric case).

```
   -----BEGIN PRIVACY-ENHANCED MESSAGE-----
   Proc-Type: 4,ENCRYPTED
   Content-Domain: RFC822
   DEK-Info: DES-CBC,BFF968AA74691AC1
   Originator-Certificate:
MIIB1TCCASCgAWUwDQYJKoZIhvcNAQECBQAwUTELMAkGA1UEBhMCVVMxIDAeBgNV
BAoTF1JTQSBEYXRhIFNlY3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDzAN
BgNVBAsTBk5PVEFSWTAeFw05MTA5MDQxODM4MTdaFw05MzA5MDMxODM4MTZaMEUx
CzAJBgNVBAYTAlVTMSAwHgYDVQQKExdSUOEgRGFOYSBTZWNlcmlOeSwgSW5jLjEU
MBIGA1UEAxMLVGVzdCBVc2VyIDEwWTAKBgRVCAEBAgICAANLADBIAkEAwHZHl7i+
yJcqDtjJCowzTdBJrdAiLAnSC+CnnjOJELyuQiBgkGrgIh3j8/x0fM+YrsyFlu3F
LZPVtzIndhYFJQIDAQABMA0GCSqGSIb3DQEBAgUAA1kACKrOPqphJYw1j+YPtcIq
iW1FPuN5jJ79Khfg7ASFxskYkEMjRNZV/HZDZQEhtVaU7Jxfzs2wfX5byMp2X3U/
   5XUXGx7qusDgHQGs7Jk9W8CW1fuSWUgN4w==
   Key-Info: RSA,
I3rRIGXUGWAF8js5wCzRTkdhO34PTHdRZY9Tuvm03M+NM7fx6qc5udixps2Lng0+
   wGrtiUm/ovtKdinz6ZQ/aQ==
   Issuer-Certificate:
MIIB3DCCAUgCAQowDQYJKoZIhvcNAQECBQAwTzELMAkGA1UEBhMCVVMxIDAeBgNV
BAoTF1JTQSBEYXRhIFNlY3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDTAL
BgNVBAsTBFRMQOEwHhcNOTEwOTAxMDgwMDAwWhcNOTIwOTAxMDc1OTU5WjBRMQsw
CQYDVQQGEwJVUzEgMB4GA1UEChMXU1NBIERhdGEgU2VjdXJpdHksIEluYy4xDzAN
BgNVBAsTBkJldGEgMTEPMA0GA1UECxMGTk9UQVJZMHAwCgYEVQgBAQICArwDYgAw
XwJYCsnp61QCxYykNlODwutF/jMJ3kL+3PjYyHOwk+/9rLg6X65B/LD4bJHtO5XW
cqAz/7R7XhjYCmOPcqbdzoACZtIlETrKrcJiDYoP+DkZ8klgCk7hQHpbIwIDAQAB
MA0GCSqGSIb3DQEBAgUAA38AAICPv4f9Gx/tY4+p4DB7MV+tKZnvBoy8zgoMGOx
dD2jMZ/3HsyWKWgSFOeH/AJB3qr9zosG47pyMnTf3aSy2nBO7CMxpUWRBcXUpE+x
EREZd9++32ofGBIXaialnOgVUnOOzSYgugiQ077nJLDUj0hQehCizEs5wUJ35a5h
   MIC-Info: RSA-MD5,RSA,
UdFJR8u/TIGhfH65ieewe21OW4tooa3vZCvVNGBZirf/7nrgzWDABz8w9NsXSexv
   AjRFbHoNPzBuxwmOAFeAOHJszL4yBvhG
   Recipient-ID-Asymmetric:
MFExCzAJBgNVBAYTAlVTMSAwHgYDVQQKExdSUOEgRGFOYSBTZWNlcmlOeSwgSW5j
   LjEPMA0GA1UECxMGQmVOYSAxMQ8wDQYDVQQLEwZOT1RBU1k=,
   66
   Key-Info: RSA,
O6BS1ww9CTyHPtS3bMLD+LOhejdvX6Qv1HK2ds2sQPEaXhX8EhvVphHYTjwekdWv
   7xOZ3Jx2vTAhOYHMcqqCjA==
qeWlj/YJ2Uf5ng9yznPbtDOmYloSwIuV9FRYx+gzY+8iXd/NQrXHfi6/MhPfPF3d
   jIqCJAxvld2xgqQimUzoS1a4r7kQQ5c/Iua4LqKeq3ciFzEv/MbZhA==
   -----END PRIVACY-ENHANCED MESSAGE-----
```

Figure 24.5 Example of an encapsulated ENCRYPTED message (asymmetric case).

ifier says that the message is encrypted and signed. The "MIC-ONLY" and "MIC-CLEAR" specifiers would indicate that the message is signed, but not encrypted. MIC-CLEAR messages are not encoded and can be read using non-PEM software. MIC-ONLY messages need PEM software to transform them to a human-readable form. A PEM message is always signed; it is optionally encrypted.

The next field, "Content-Domain," specifies the type of mail message. It has nothing to do with security. The "DEK-Info" field gives information on the **Data Exchange Key** (**DEK**), the encryption algorithm used to encrypt the text, and any parameters associated with the encryption algorithm. Only DES in CBC mode is currently specified, or "DES-CBC." The second subfield specifies the IV. Other algorithms may be specified by PEM in the future; their use will be noted in DEK-Info and in other fields that identify algorithms.

For messages with symmetric key management (see Figure 24.4), the next field is "Originator-ID-Symmetric" with three subfields. The first subfield identifies the sender by a unique electronic mail address. The second subfield is optional and identifies the authority that issued the interchange key. The third is an optional Version/Expiration subfield.

Continuing with the symmetric key-management case, each recipient has two fields: "Recipient-ID-Symmetric" and "Key-Info." The "Recipient-ID-Symmetric" field has three subfields; these identify the receiver in the same way that "Originator-ID-Symmetric" identified the sender.

The "Key-Info" field specifies the key-management parameters. This field has four subfields. The first subfield gives the algorithm used to encrypt the DEK. Since the key management in this message is symmetric, the sender and receiver have to share a common key. This is called the **Interchange Key** (**IK**), which is used to encrypt the DEK. The DEK can be either encrypted using DES in ECB (denoted by "DES-ECB") or triple-DES (which would be denoted "DES-EDE"). The second subfield specifies the MIC algorithm. It can be either MD2 (denoted by "RSA-MD2") or MD5 (which would be denoted "RSA-MD5"). The third subfield, the DEK, and the fourth field, the MIC, are both encrypted with the IK.

Figures 24.5 and 24.6 show messages with public-key key management (called "asymmetric" in PEM nomenclature). The headers are different. In ENCRYPTED messages, after the "DEK-Info" field comes the "Originator-Certificate" field. The certificate follows the X.509 standard (see Section 24.9). The next field is "Key-Info" with two subfields. The first subfield specifies the public-key algorithm used to encrypt the DEK; currently only RSA is supported. The next subfield is the DEK, encrypted in the originator's public key. This is an optional field, intended to permit the originator to decrypt his own message in the event that it is returned by the mail system. The next field "Issuer-Certificate," is the certificate of whomever signed the Originator-Certificate.

Continuing with the asymmetric key-management case, the next field is "MIC-Info." The first subfield gives the algorithm under which the MIC was computed. The second subfield shows the algorithm under which the MIC was signed. The third subfield consists of the MIC, signed by the sender's private key.

```
   -----BEGIN PRIVACY-ENHANCED MESSAGE-----
   Proc-Type: 4,MIC-ONLY
   Content-Domain: RFC822
   Originator-Certificate:
MIIB1TCCASCgAWIwDQYJKoZIhvcNAQECBQAwUTELMAkGA1UEBhMCVVMxIDAeBgNV
BAoTF1JTQSBEYXRhIFN1Y3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDzAN
BgNVBAsTBk5PVEFSWTAeFw05MTA5MDQxODM4MTdaFw05MzA5MDMxODM4MTZaMEUx
CzAJBgNVBAYTA1VTMSAwHgYDVQQKExdSU0EgRGF0YSBTZWN1cml0eSwgSW5jLjEU
MBIGA1UEAxMLVGVzdCBVc2VyIDEwWTAKBgRVCAEBAgICAANLADBIAkEAwHZH17i+
yJcqDtjJCowzTdBJrdAiLAnSC+CnnjOJELyuQiBgkGrgIh3j8/x0fM+YrsyF1u3F
LZPVtzlndhYFJQIDAQABMA0GCSqGSIb3DQEBAgUAA1kACKrOPqphJYw1j+YPtcIq
iW1FPuN5jJ79Khfg7ASFxskYkEMjRNZV/HZDZQEhtVaU7Jxfzs2wfX5byMp2X3U/
   5XUXGx7qusDgHQGs7Jk9W8CW1fuSWUgN4w==
   Issuer-Certificate:
MIIB3DCCAUgCAQowDQYJKoZIhvcNAQECBQAwTzELMAkGA1UEBhMCVVMxIDAeBgNV
BAoTF1JTQSBEYXRhIFN1Y3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDTAL
BgNVBAsTBFRMQ0EwHhcNOTEwOTAxMDgwMDAwWhcNOTIwOTAxMDc1OTU5WjBRMQsw
CQYDVQQGEwJVUzEgMB4GA1UEChMXU1NBIERhdGEgU2VjdXJpdHksIEluYy4xDzAN
BgNVBAsTBkJldGEgMTEPMA0GA1UECxMGTk9UQVJZMHAwCgYEVQgBAQICArwDYgAw
XwJYCsnp6lQCxYykNlODwutF/jMJ3kL+3PjYyHOwk+/9rLg6X65B/LD4bJHtO5XW
cqAz/7R7XhjYCmOPcqbdzoACZtIlETrKrcJiDYoP+DkZ8klgCk7hQHpbIwIDAQAB
MA0GCSqGSIb3DQEBAgUAA38AAICPv4f9Gx/tY4+p+4DB7MV+tKZnvBoy8zgoMGOx
dD2jMZ/3HsyWKWgSFOeH/AJB3qr9zosG47pyMnTf3aSy2nBO7CMxpUWRBcXUpE+x
EREZd9++32ofGBIXaialnOgVUnOOzSYgugiQ077nJLDUjOhQehCizEs5wUJ35a5h
   MIC-Info: RSA-MD5,RSA,
jV2OfH+nnXHU8bnL8kPAad/mSQ1TDZ1bVuxvZAOVRZ5q5+Ej15bQvqNeqOUNQjr6
   EtE7K2QDeVMCyXsdJ1A8fA==
LSBBIG11c3NhZ2UgZm9yIHVzZSBpbiBOZXN0aW5nLgOKLSBGb2xsb3dpbmcgaXMg
   YSBibGFuayBsaW51OgOKDQpUaG1zIG1zIHRoZSB1bmQuDQo=
   -----END PRIVACY-ENHANCED MESSAGE-----
```

Figure 24.6 Example of an encapsulated MIC-ONLY message (asymmetric case).

Still continuing with asymmetric key management, the next fields deal with the recipients. There are two fields for each recipient: "Recipient-ID-Asymmetric" and "Key-Info." The "Recipient-ID-Asymmetric" field has two subfields. The first identifies the authority that issued the receiver's public key; the second is an optional Version/Expiration subfield. The "Key-Info" field specifies the key management parameters: The first subfield identifies the algorithm used to encrypt the message and the second subfield is the DEK encrypted with the receiver's public key.

Security of PEM

RSA keys in PEM can range from 508 bits to 1024 bits. This should be long enough for anyone's security needs. A more likely attack would be against the key-management protocols. Mallory could steal your private key—don't write it down

anywhere—or attempt to fool you into accepting a bogus public key. The key certification provisions of PEM make this unlikely if everyone follows proper procedures, but people have been known to be sloppy.

A more insidious attack would be for Mallory to modify the PEM implementation running on your system. This modified implementation could surreptitiously send Mallory all of your mail, encrypted with his public key. It could even send him a copy of your private key. If the modified implementation works well, you will never know what is happening.

There's no real way to prevent this kind of attack. You could use a one-way hash function and fingerprint the PEM code. Then, each time you run it, you could check the fingerprint for modification. But Mallory could modify the fingerprint code at the same time he modifies the PEM code. You could fingerprint the fingerprint code, but Mallory could modify that as well. If Mallory can get access to your machine, he can subvert the security of PEM.

The moral is that you can never really trust a piece of software if you cannot trust the hardware it is running on. For most people, this kind of paranoia is unwarranted. For some, it is very real.

TIS/PEM

Trusted Information Systems, partially supported by the U.S. government Advanced Research Projects Agency, has designed and implemented a reference implementation of PEM (TIS/PEM). Developed for UNIX-based platforms, it has also been ported to VMS, DOS, and Windows.

Although the PEM specifications indicate a single certification hierarchy for use by the Internet, TIS/PEM supports the existence of multiple certification hierarchies. Sites may specify a set of certificates that are to be considered valid, including all certificates issued by them. A site need not join the Internet hierarchy in order to use TIS/PEM.

TIS/PEM is currently available to all U.S. and Canadian organizations and citizens upon request. It will be distributed in source code form. Interested parties should contact: Privacy-Enhanced Mail, Trusted Information Systems, Inc., 3060 Washington Road (Rte. 97), Glenwood, MD 21738; (301) 854-6889; fax: (301) 854-5363; Internet: pem-info@tis.com.

RIPEM

RIPEM is a program, written by Mark Riordan, that implements the PEM protocols. Although technically not public domain, the program is publicly available and can be used royalty-free for personal, noncommercial applications. A license for its use is included with the documentation.

The code cannot be exported. Of course, U.S. government laws don't apply outside the United States, and some people have ignored the export rules. RIPEM code is available on bulletin boards worldwide. Something called RIPEM/SIG, which only does digital signatures, is exportable.

At this writing, RIPEM is not a complete implementation of the PEM protocols; it does not implement certificates for authenticating keys.

Before writing RIPEM, Riordan wrote a similar program called RPEM. This was intended to be a public-domain electronic-mail encryption program. To try to avoid patent issues, Riordan used Rabin's algorithm (see Section 19.5). Public Key Partners claimed that their patents were broad enough to cover all of public-key cryptography and threatened to sue; Riordan stopped distributing the program.

RPEM isn't really used anymore. It is not compatible with RIPEM. Since RIPEM can be used with the full blessing of Public Key Partners, there is no reason to use RPEM instead.

24.11 MESSAGE SECURITY PROTOCOL (MSP)

The Message Security Protocol (MSP) is the military equivalent of PEM. It was developed by the NSA in the late 1980s under the Secure Data Network System (SDNS) program. It is an X.400-compatible application-level protocol for securing electronic mail. MSP will be used for signing and encrypting messages in the Department of Defense's planned Defense Message System (DMS) network.

The Preliminary Message Security Protocol (PMSP), to be used for "unclassified but sensitive" messages, is a version of MSP adapted for use with both X.400 and TCP/IP. This protocol is also called Mosaic.

Like PEM, MSP and PMSP software applications are flexible and designed to accommodate a variety of algorithms for security functions including signing, hashing, and encryption. PSMP will work with the Capstone chip (see Section 24.17).

24.12 PRETTY GOOD PRIVACY (PGP)

Pretty Good Privacy (PGP) is a freeware electronic-mail security program, originally designed by Philip Zimmermann [1652]. It uses IDEA for data encryption, RSA (with keys up to 2047 bits) for key management and digital signatures, and MD5 as a one-way hash function.

PGP's random public keys use a probabilistic primality tester, and get their initial seeds from measuring the user's keyboard latency while typing. PGP generates random IDEA keys using the method delineated in ANSI X9.17, Appendix C (see Section 8.1) [55], with IDEA as the symmetric algorithm instead of DES. PGP also encrypts the user's private key using a hashed pass phrase instead of a password.

PGP-encrypted messages have layered security. The only thing a cryptanalyst can learn about an encrypted message is who the recipient is, assuming he knows the recipient's key ID. Only after the recipient decrypts the message does he learn who signed the message, if it is signed. Contrast this approach with PEM, which leaves quite a bit of information about the sender, recipient, and message in the unencrypted header.

The most interesting aspect of PGP is its distributed approach to key management (see Section 8.12). There are no key certification authorities; PGP instead supports a "web of trust." Every user generates and distributes his own public key. Users sign each other's public keys, creating an interconnected community of PGP users.

For example, Alice might physically give her public key to Bob. Bob knows Alice, so he signs her public key. He then gives the signed key back to her and keeps a copy for himself. When Alice wants to communicate with Carol, Alice sends Carol a copy of the key Bob signed. Carol, who already has Bob's public key (she got it at some other time) and trusts Bob to certify other people's keys, verifies his signature on Alice's key and accepts it as valid. Bob has introduced Alice to Carol.

PGP does not specify a policy for establishing trust; users are free to decide who they trust and who they do not. PGP provides mechanisms for associating trust with public keys and for using trust. Each user keeps a collection of signed public keys in a file called a **public-key ring**. Each key in the ring has a key legitimacy field that indicates the degree to which the particular user trusts the validity of the key. The higher the trust level, the more the user believes the key is legitimate. A signature trust field measures how far the user trusts the signer to certify the public keys of other users. And finally, an owner trust field indicates the degree to which the particular user trusts the key's owner to sign other public keys; this field is set manually by the user. PGP continuously updates these fields as users supply new information.

Figure 24.7 shows how this model might look for a particular user, Alice. Alice's key is at the top, and the owner trust value is ultimate trust. Alice has signed Bob's, Carol's, Dave's, Ellen's, and Frank's keys. She trusts Bob and Carol to sign other people's public keys, and she partially trusts Dave and Ellen to sign other people's public keys. And she trusts Gail to sign other people's public keys, even though she has not signed Gail's key herself.

Two partially trusted signatures may be sufficient to certify a key. Alice believes that Kurt's key is legitimate because both Dave and Ellen have signed it. This is not automatic in PGP; Alice can set her own paranoia level.

Just because Alice believes a key to be valid, she does not have to trust it to sign other people's keys. She does not trust Frank to sign other people's public keys, even though she signed his key herself. And she does not trust Ivan's signature on Martin's key, or Kurt's signature on Nancy's key.

Owen's key doesn't fit into the web anywhere; perhaps Alice got it from a key server. PGP does not assume that the key is valid; Alice must either declare the key valid or decide to trust one of the key's signers.

Of course, nothing prevents Alice from using keys she does not trust. PGP's job is to alert Alice that the key is not trusted, not to prevent communications.

The weakest link of this whole system is key revocation: It is impossible to guarantee that no one will use a compromised key. If Alice's private key is stolen she can send out something called a **key revocation certificate**, but since key distribution is *ad hoc* and largely word of mouth there is no guarantee that it will reach

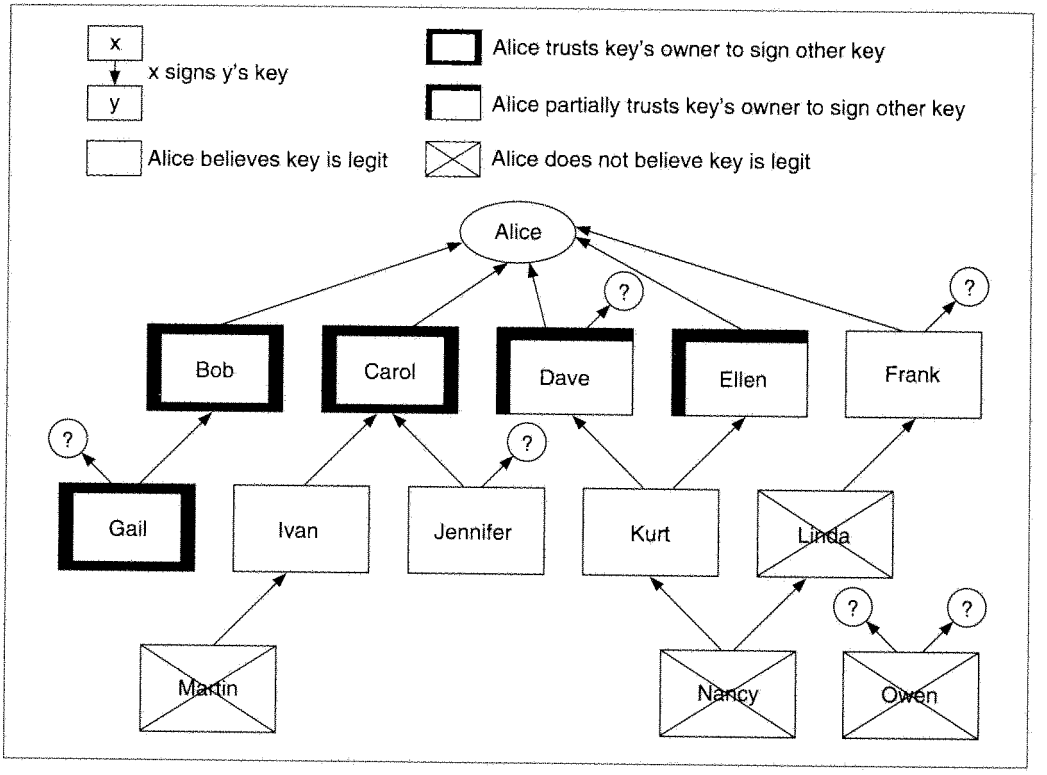


Figure 24.7 PGP trust model.

everyone who has her public key on his key ring. And as Alice has to sign the key revocation certificate with her private key; if she loses the key altogether she cannot revoke it.

The current version of PGP is 2.6.2. A new version of PGP, PGP 3.0, is scheduled for release by the end of 1995. Changes in 3.0 include options for triple-DES, SHA, and other public-key algorithms, a split of the encryption and signature public-key/private-key key pairs, enhanced procedures for key revocation, improved key-ring management functions, an API for integrating PGP in other programs, and a completely rewritten code base.

PGP is available for MS-DOS, UNIX, Macintosh, Amiga, and Atari. It is free for personal, noncommercial use, and is available from many ftp sites on the Internet. To ftp PGP from MIT, telnet to net-dist.mit.edu, log in as getpgp, answer the questions, then ftp to net-dist.mit.edu and change to the directory named in the telnet session. It is also available from ftp.ox.ac.uk, ftp.dsi.unimi.it, ftp.funet.fi, ftp.demon.co.uk, Compuserve, AOL, and elsewhere. For U.S. commercial users, PGP can be bought—com-

plete with licenses—for about $100 from a company called ViaCrypt, 9033 N 24th Ave., Phoenix, AZ, 85021; (602) 944-0773; viacrypt@acm.org. Several shareware front-ends are available to help integrate PGP into MS-DOS, Microsoft Windows, Macintosh, and UNIX.

There are several books about PGP [601,1394,1495]. The source code has even been published in book form [1653] in an attempt to frustrate the U.S. Department of State, which continues to maintain that source code is exportable on paper but not electronically. Assuming you trust IDEA, PGP is the closest you're likely to get to military-grade encryption.

24.13 Smart Cards

A smart card is a plastic card, the size and shape of a credit card, with an embedded computer chip. It's an old idea—the first patents were filed 20 years ago—but practical limitations made them feasible only five or so years ago. Since then they have taken off, mostly in Europe. Many countries use smart cards for pay telephones. There are also smart credit cards, smart cash cards, smart everything cards. The U.S. credit-card companies are looking at the technology, and within a few years even backwards Americans will have smart cards in their wallets.

A smart card contains a small computer (usually an 8-bit microprocessor), RAM (about a quarter kilobyte), ROM (about 6 or 8 kilobytes), and either EPROM or EEPROM (a few kilobytes). Future-generation smart cards will undoubtedly have more capacity, but some physical limitations on smart cards make expansion difficult. The card has its own operating system, programs, and data. (What it doesn't have is power; that comes when the card is plugged in to a reader.) And it is secure. In a world where you might not trust someone else's computer or telephone or whatever, you can still trust a card that you keep with you in your wallet.

Smart cards can have different cryptographic protocols and algorithms programmed into them. They might be configured as an electronic purse, and be able to spend and receive digital cash. They may be able to perform zero-knowledge authentication protocols; they may have their own encryption keys. They might be able to sign documents, or unlock applications on a computer.

Some smart cards are assumed to be tamperproof; this often protects the institution that issues the cards. A bank wouldn't want you to be able to hack their smart card to give yourself more money.

There is a lot of interest in smart cards, and a lot of information about them is available. A good survey article on the cryptography in smart cards is [672]. CARTES is a conference held in Paris every October; and CardTech is held in Washington, D.C. every April. The proceedings of two other smart-card conferences are [342, 382]. There are hundreds of smart-card patents, mostly owned by European companies. An interesting paper on possible future applications—integrity checking, audit trails, copy protection, digital cash, secure postage meters—is [1628].

24.14 PUBLIC-KEY CRYPTOGRAPHY STANDARDS (PKCS)

The Public-Key Cryptography Standards (PKCS) are RSA Data Security, Inc.'s attempt to provide an industry standard interface for public-key cryptography. Traditionally, this sort of thing would be handled by ANSI, but, considering the current situation in cryptography politics, RSADSI figured that they had better do it on their own. Working with a variety of companies, they developed a series of standards. Some are compatible with other standards and some are not.

These are not standards in the traditional sense of the word; no standards body convened and voted on PKCS. According to its own materials, RSADSI will "retain sole decision-making authority on what each standard is" and will "publish revised standards when appropriate" [803].

Even so, there is a lot of good stuff here. If you're not sure what kind of syntax and data structures to use when programming public-key cryptography, these standards are probably as good as anything else you can come up with. And, since they're not really standards, you can tailor them to suit your needs.

Following is a short description of each PKCS (PKCS #2 and PKCS #4 have been incorporated into PKCS #1).

PKCS #1 [1345] describes a method for RSA encryption and decryption, primarily for constructing the digital signatures and digital envelopes described in PKCS #7. For digital signatures, the message is hashed and then the hash is encrypted with the private key of the signer. Both message and hash are represented together as detailed in PKCS #7. For digital envelopes (encrypted messages), the message is first encrypted with a symmetric algorithm, and then the message key is encrypted with the public key of the recipient. The encrypted message and encrypted key are represented together according to the syntax of PKCS #7. Both of these methods are compatible with PEM standards. PKCS #1 also describes a syntax, identical to the syntax in X.509 and PEM, for RSA public and private keys and three signature algorithms—MD2 and RSA, MD4 and RSA, and MD5 and RSA—for signing certificates and the like.

PKCS #3 [1346] describes a method for implementing Diffie-Hellman key exchange.

PKCS #5 [1347] describes a method for encrypting messages with a secret key derived from a password. It uses either MD2 or MD5 to derive the key from the password, and encrypts with DES in CBC mode. The method is intended primarily to encrypt private keys when transferring them from one computer system to another, but can be used to encrypt messages.

PKCS #6 [1348] describes a standard syntax for public key certificates. The syntax is a superset of an X.509 certificate, so that X.509 certificates can be extracted if necessary. Over and above the X.509 set, additional attributes extend the certification process beyond just the public key. These include other information, such as electronic mail address.

PKCS # 7 [1349] is a general syntax for data that may be encrypted or signed, such as digital envelopes or digital signatures. The syntax is recursive, so that envelopes can be nested, or someone can sign some previously encrypted data. The syntax also

allows other attributes, such as timestamps, to be authenticated along with the message content. PKCS #7 is compatible with PEM so that signed and encrypted messages can be converted to PEM messages without any cryptographic operations, and vice versa. PKCS #7 can support a variety of architectures—PEM is one—for certificate-based key management.

PKCS #8 [1350] describes a syntax for private key information—including a private key and a set of attributes—and a syntax for encrypted private keys. PKCS #5 can be used to encrypt the private key information.

PKCS #9 [1351] defines selected attribute types for PKCS #6 extended certificates, PKCS #7 digitally signed messages, and PKCS #8 private-key information.

PKCS #10 [1352] describes a standard syntax for certification requests. A certification comprises a distinguished name, a public key, and (optionally) a set of attributes, collectively signed by the person requesting certification. Certification requests are sent to a certification authority, who either transforms the request into an X.509 public-key certificate or a PKCS #6 certificate.

PKCS #11 [1353], the Cryptographic Token API Standard, specifies a programming interface called "Cryptoki" for portable cryptographic devices of all kinds. Cryptoki presents a common logical model, enabling applications to perform cryptographic operations on portable devices without knowing details of the underlying technology. The standard also defines application profiles: sets of algorithms that a device may support.

PKCS #12 [1354] describes syntax for storing in software a user's public keys, protected private keys, certificates, and other related cryptographic information. The goal is to standardize on a single key file for use among a variety of applications.

These standards are comprehensive, but not exhaustive. Many things are outside their scope: the problem of naming, noncryptographic issues surrounding certification, key lengths, and conditions on various parameters. What the PKCS provide are a format for transferring data based on public-key cryptography and an infrastructure to support that transfer.

24.15 Universal Electronic Payment System (UEPS)

The UEPS is a smart-card banking application initially developed for rural South Africa, but later adopted by all of that country's major banking groups. About 2 million cards were issued in that country by early 1995. It has also been adopted in Namibia, and is also being deployed by at least one bank in Russia.

The system provides a secure debit card suitable for regions where poor telephone service make on-line verification impossible. Both customers and merchants have cards; customers can use their cards to transfer money to merchants. Merchants can then take their cards to a telephone and deposit the money in their bank account; customers can take their cards to a telephone and have money moved onto their card. There is no intention to provide anonymity, only to prevent fraud.

Here is the communications protocol between customer Alice and merchant Bob. (Actually, Alice and Bob just plug their cards into a machine and wait for it to complete the transaction.) When Alice first gets her card, she is given a key pair, K_1 and K_2; the bank calculates them from her name and some secret function. Only the merchant cards have the secrets necessary to work out these customer keys.

(1) Alice sends Bob her name, A, his name, B, and a random number, R_A, encrypted using DES: first with K_2 and then with K_1. She also sends her name in the clear.

$$A, E_{K_1}[E_{K_2}[A, B, R_A]]$$

(2) Bob calculates K_1 and K_2 from Alice's name. He decrypts the message, confirms that A and B are correct, then encrypts Alice's unencrypted second message with K_2.

$$E_{K_2}[A, B, R_A]$$

Bob does not send this message to Alice; 56 bits of the ciphertext become K_3. Bob then sends Alice his name, her name, and another random number, R_B, encrypted using DES: first with K_3 and then with K_1.

$$E_{K_1}[E_{K_3}[B, A, R_B]]$$

(3) Alice computes K_3 in the same manner Bob did. She decrypts Bob's message, confirms that B and A are correct, then encrypts Bob's unencrypted message with K_3.

$$E_{K_3}[B, A, R_B]$$

Alice does not send this message to Bob; 56 bits of the ciphertext become K_4. Alice then sends Bob her name, his name, and the digital check, C. This check contains the names of the sender and recipient, a date, a check number, an amount, and two MACs, all encrypted using DES: first with K_4 and then with K_1. One of the MACs can be verified by Alice's bank, and the other can only be verified by the clearing center. Alice debits her account by the correct amount.

$$E_{K_1}[E_{K_4}[A, B, C]]$$

(4) Bob computes K_4 in the same manner Alice did. Assuming all the names match and the check is correctly formed, he accepts it for payment.

A really clever thing about this protocol is that the encryption key for each message depends on the previous message. Each message doubles as an authenticator for *all* previous messages. This means that someone can't replay an old message; the receiver could never decrypt it. I am impressed with this idea and expect that it will see wider use once it becomes widely known.

Another clever thing about this protocol is that it enforces correct implementation. If the application developer doesn't implement this protocol correctly, it just won't work.

Both cards store records of every transaction. When the cards eventually go online to communicate with the bank—the merchant to deposit his money and the customer to get more money—the bank uploads these records for auditing purposes.

Tamperproof hardware prevents either participant from messing with the data; Alice cannot change the value of her card. Extensive audit trails provide data to identify and prosecute fraudulent transactions. There are universal secrets in the cards—MAC keys in the customer cards, functions to convert customer names to K_1 and K_2 in the merchant cards—but these are assumed to be difficult to reverse-engineer.

This scheme is not meant to be perfect, only more secure than either paper checks or traditional debit cards. The threat of fraud is not from rival militaries, but from opportunistic customers and merchants. UEPS protects against that kind of abuse.

The message exchange is an excellent example of a robust protocol: Every message names both parties, includes unique information to ensure freshness, and depends explicitly on all the messages that came before it.

24.16 CLIPPER

The Clipper chip (also known as the MYK-78T) is an NSA-designed, tamper-resistant VLSI chip designed for encrypting voice conversations; it is one of the two chips that implements the U.S. government's Escrowed Encryption Standard (EES) [1153]. VLSI Technologies, Inc. manufactures the chip, and Mykotronx, Inc. programs it. Initially, the Clipper chip will be available in the AT&T Model 3600 Telephone Security Device (see Section 24.18). The chip implements the Skipjack encryption algorithm (see Section 13.12), an NSA-designed classified secret-key encryption algorithm, in OFB only.

The most controversial aspect of the Clipper chip, and the entire EES, is the key-escrow protocol (see Section 4.14). Each chip has a special key, not needed for messages. This key is used to encrypt a copy of each user's message key. As part of the synchronization process, the sending Clipper chip generates and sends a **Law Enforcement Access Field** (**LEAF**) to the receiving Clipper chip. The LEAF contains a copy of the current session key, encrypted with a special key (called the **unit key**). This allows a government eavesdropper to recover the session key, and then recover the plaintext of the conversation.

According to the director of NIST [812]:

> A "key-escrow" system is envisioned that would ensure that the "Clipper Chip" is used to protect the privacy of law-abiding Americans. Each device containing the chip will have two unique "keys," numbers that will be needed by authorized government agencies to decode messages encoded by the device. When the device is manufactured, the two keys would be deposited separately in two "key-escrow" databases established by the attorney general. Access to these keys would be limited to government officials with legal authorization to conduct a wiretap.

The government also wants to encourage the sale of telephones with these devices abroad; no one knows what might happen to those key-escrow databases.

Politics aside, the internal structure of the LEAF is worth discussing [812,1154, 1594,459,107,462]. The LEAF is a 128-bit string containing enough information to allow law enforcement to recover the session key, K_S, assuming the two **escrow agencies** in charge of those key-escrow databases cooperate. The LEAF contains a 32-bit unit identifier, U, unique to the Clipper chip. It also contains the current 80-bit session key encrypted with the chip's unique unit key, K_U, and a 16-bit checksum, C, called an escrow identifier. This checksum is a function of the session key, the IV, and possibly other information. These three fields are encrypted with a fixed family key, K_F, shared by all interoperable Clipper chips. The family key, the encryption modes used, the details of the checksum, and the exact structure of the LEAF are all secret. It probably looks something like this:

$$E_{K_F}[E_{K_U}[K_S, U, C]]$$

K_U is programmed into Clipper chips at the factory. This key is then split (see Section 3.5) and stored in two different key-escrow databases, guarded by two different escrow agencies.

For Eve to recover K_S from the LEAF, she first has to decrypt the LEAF with K_F and recover U. Then she has to take a court order to each escrow agency, who each return half of K_U for the given U. Eve XORs the two halves together to recover K_U, then she uses K_U to recover K_S, and K_S to eavesdrop on the conversation.

The checksum is designed to prevent someone from circumventing this scheme; the receiving Clipper chip won't decrypt if the checksum doesn't check. However, there are only 2^{16} possible checksum values, and a bogus LEAF with the right checksum but the wrong key can be found in about 42 minutes [187]. This isn't much help for Clipper voice conversations. Because the key exchange protocol is not part of the Clipper chip, the 42-minute brute-force attack must occur after key exchange; it cannot be done before making the telephone call. This attack may work for facsimile transmission or with the Fortezza card (see Section 24.17).

Supposedly, the Clipper chip will resist reverse-engineering by "a very sophisticated, well-funded adversary" [1154], but rumors are that Sandia National Laboratories successfully reverse-engineered one. Even if those rumors aren't true, I suspect that the largest chip manufacturers in the world can reverse-engineer Clipper; it's just a matter of time before someone with the right combination of resources and ethics comes along.

Enormous privacy issues are associated with this scheme. Numerous civil liberty advocacy groups are actively campaigning against any key-escrow mechanism that gives the government the right to eavesdrop on citizens. But the sneaky thing is that this idea never went through Congress; NIST published the Escrowed Encryption Standard as a FIPS [1153], bypassing that irritating legislative process. Right now it looks like the EES is dying a slow and quiet death, but standards have a way of creeping up on you.

Anyway, Table 24.2 lists the different agencies participating in this program. Anyone want to do a threat analysis on having both escrow agents in the executive

branch? Or on having escrow agents who really don't know anything about the wiretap requests, and can do no more than blindly approve them? Or on having the government impose a secret algorithm as a commercial standard?

In any case, implementing Clipper raises enough problems to question its value in court. Remember, Clipper only works in OFB mode. Despite what you may have been told to the contrary, this does not provide integrity or authentication. Imagine that Alice is on trial, and a Clipper-encrypted telephone call is part of the evidence. Alice claims that she never made the call; the voice is not hers. The phone's compression algorithm is so bad that it is hard to recognize Alice's voice, but the prosecution argues that since only Alice's escrowed key will decipher the call it must have been made from her telephone.

Alice argues that the call was forged like so [984,1339]: Given the ciphertext and the plaintext, it is possible to XOR them to get the keystream. This keystream can then be XORed with an entirely different plaintext to form a forged ciphertext, which can then be converted to forged plaintext when fed into the Clipper decryptor. True or not, this argument could easily put enough doubt in a jury's mind to disregard the telephone call as evidence.

Another attack, called the Squeeze attack, allows Alice to frame Bob. Here's how [575]: Alice calls Bob using Clipper. She saves a copy of his LEAF as well as the session key. Then, she calls Carol (who she knows is being wiretapped). During the key setup, Alice forces the session key to be identical to the one she used with Bob; this requires hacking the phone, but it is not hard. Then, instead of sending her LEAF she sends Bob's. It's a valid LEAF, so Carol's phone will not notice. Now she can say whatever she wants to Carol; when the police decrypt the LEAF, they will find that it is Bob's. Even if Bob wasn't framed by Alice, the mere fact that he can claim this in court undermines the purpose of the scheme.

The law enforcement authorities of the United States should not be in the business of collecting information in criminal investigations that is useless in court. Even if key escrow were a good idea, Clipper is a bad way of implementing it.

24.17 CAPSTONE

Capstone (also known as the MYK-80) is the other NSA-developed VLSI cryptographic chip that implements the U.S. government's Escrowed Encryption Standard [1153]. Capstone includes the following functions [1155,462]:

Table 24.2
EES Participating Agencies

Justice—System Sponsor and Family Key Agent
NIST—Program Manager and Escrow Agent
FBI—Decrypt User and Family Key Agent
Treasury—Escrow Agent
NSA—Program Developer

- The Skipjack algorithm in any of the four basic modes: ECB, CBC, CFB, and OFB.
- A public-key Key Exchange Algorithm (KEA), probably Diffie-Hellman.
- The Digital Signature Algorithm (DSA).
- The Secure Hash Algorithm (SHA).
- A general purpose exponentiation algorithm.
- A general purpose, random-number generator that uses a pure noise source.

Capstone provides the cryptographic functionality needed for secure electronic commerce and other computer-based applications. The first application of Capstone is in a PCMCIA card called Fortezza. (It was originally called Tessera until a company called Tessera, Inc. complained.)

NSA had considered lengthening Capstone's LEAF checksum in production versions for use in Fortezza cards, in order to foil the brute-force attack against the LEAF previously discussed. Instead, they added a feature that reset the card after 10 incorrect LEAFs. This only increases the time required to find a fake but valid LEAF by 10 percent, to 46 minutes. I am not impressed.

24.18 AT&T MODEL 3600 TELEPHONE SECURITY DEVICE (TSD)

The AT&T Telephone Security Device (TSD) is the Clipper phone. Actually, there are four models of the TSD. One contains the Clipper chip, another contains an exportable proprietary AT&T encryption algorithm, the third contains a proprietary algorithm for domestic use plus the exportable algorithm, and the fourth contains the Clipper, domestic, and exportable algorithms.

TSDs use a different session key for each telephone call. A pair of TSDs generate a session key using Diffie-Hellman key exchange, independent of the Clipper chip. Since Diffie-Hellman incorporates no authentication, the TSD has two methods to thwart a man-in-the-middle attack.

The first is a screen. The TSD hashes the session key and displays that hash on a small screen as four Hex digits. The conversants should confirm that their screens show the same digits. The voice quality is good enough that they can recognize each other's voice.

Eve still has a possible attack. Imagine her in the middle of Alice and Bob's conversation. She uses one TSD on the line with Alice and a modified TSD on the line with Bob; in the middle she bridges the two phone calls. Alice tries to go secure. She generates a key as normal, except that Eve is acting as Bob. Eve recovers the key, and using the modified TSD, forces the key she generates with Bob to have the same

hash value. This attack may not sound very likely, but the TSD uses a variant of the interlock protocol to prevent it.

The TSD generates random numbers using a noise source and a chaotic amplifier with digital feedback. This generates a bit stream, which is fed through a post-whitening filter using the digital signal processor.

Despite all of this, the TSD manual does not mention security at all. In fact, it says [70]:

> AT&T makes no warranty that the TSD will prevent cryptanalytic attack on any encrypted transmission by any government agency, its agents, or any third party. Furthermore, AT&T makes no warranty that the TSD will prevent any attack on any communication by methods which bypass encryption.

CHAPTER 25

Politics

25.1 NATIONAL SECURITY AGENCY (NSA)

The NSA is the National Security Agency (once called "No Such Agency" or "Never Say Anything," but they've been more open recently), the official security body of the U.S. government. President Harry Truman created the agency in 1952 under the Department of Defense, and for many years its very existence was kept secret. The NSA is concerned with signals intelligence; its mandate is to listen in on and decode all foreign communications of interest to the security of the United States.

The following paragraphs are excerpted from NSA's original charter, signed by President Truman in 1952, and classified for many years thereafter [1535]:

> The COMINT mission of the National Security Agency (NSA) shall be to provide an effective, unified organization and control of the communications intelligence activities of the United States conducted against foreign governments, to provide for integrated operational policies and procedures pertaining thereto. As used in this directive, the terms "communications intelligence" or "COMINT" shall be construed to mean all procedures and methods used in the interception of communications other than foreign press and propaganda broadcasts and the obtaining of information from such communications by other than intended recipients, but shall exclude censorship and the production and dissemination of finished intelligence.
>
> The special nature of COMINT actives requires that they be treated in all respects as being outside the framework of other or general intelligence activities. Order, directives, policies, or recommendations of any authority of the Executive Branch relating to the collection, production, security, handling, dissemination, or utilization of intelligence, and/or classified material, shall not be applicable to COMINT actives, unless specifically so stated and issued by competent departmental of agency authority represented on the Board. Other National Security

> Council Intelligence Directive to the Director of Central Intelligence and related implementing directives issued by the Director of Central Intelligence shall be construed as non-applicable to COMINT activities, unless the National Security Council has made its directive specifically applicable to COMINT.

NSA conducts research in cryptology, both in designing secure algorithms to protect U.S. communications and in designing cryptanalytic techniques to listen in on non-U.S. communications. The NSA is known to be the largest employer of mathematicians in the world; it is also the largest purchaser of computer hardware in the world. The NSA probably possesses cryptographic expertise many years ahead of the public state of the art (in algorithms, but probably not in protocols) and can undoubtedly break many of the systems used in practice. But, for reasons of national security, almost all information about the NSA—even its budget—is classified. (Its budget is rumored to be $13 billion per year—including military funding of NSA projects and personnel—and it is rumored to employ 16,000 people.)

The NSA uses its power to restrict the public availability of cryptography, so as to prevent national enemies from employing encryption methods too strong for the NSA to break. James Massey discusses this struggle between academic and military research in cryptography [1007]:

> If one regards cryptology as the prerogative of government, one accepts that most cryptologic research will be conducted behind closed doors. Without doubt, the number of workers engaged today in such secret research in cryptology far exceeds that of those engaged in open research in cryptology. For only about 10 years has there in fact been widespread open research in cryptology. There have been, and will continue to be, conflicts between these two research communities. Open research is a common quest for knowledge that depends for its vitality on the open exchange of ideas via conference presentations and publications in scholarly journals. But can a government agency, charged with responsibilities of breaking the ciphers of other nations, countenance the publication of a cipher that it cannot break? Can a researcher in good conscience publish such a cipher that might undermine the effectiveness of his own government's code-breakers? One might argue that publication of a provably secure cipher would force all governments to behave like Stimson's "gentlemen," but one must be aware that open research in cryptography is fraught with political and ethical considerations of a severity more than in most scientific fields. The wonder is not that some conflicts have occurred between government agencies and open researchers in cryptology, but rather that these conflicts (at least those of which we are aware) have been so few and so mild.

James Bamford wrote a fascinating book about the NSA: *The Puzzle Palace* [79], recently updated by Bamford and Wayne Madsen [80].

The Commercial COMSEC Endorsement Program (CCEP)

The Commercial COMSEC Endorsement Program (CCEP), codenamed Overtake, is a 1984 NSA initiative to facilitate the development of computer and communications products with embedded cryptography [85,1165]. The military had always paid

for this kind of thing for themselves, and it was very expensive. The NSA figured that if companies could sell equipment to both the military and to corporate users, even overseas, costs would go down and everyone would benefit. They would no longer endorse equipment as complying with Federal Standard 1027, and then CCEP would provide government-endorsed cryptographic equipment [419].

NSA developed a series of cryptographic modules for different purposes. Different algorithms would be used in the modules for different applications, and manufacturers would be able to pull one module out and plug in another depending on the customer. There were modules for military use (Type I), modules for "unclassified but sensitive" government use (Type II), modules for corporate use (Type III), and modules for export (Type IV). Table 25.1 summarizes the different modules, applications, and names.

This program is still around, but never became popular outside the government. All the modules were tamperproof, all the algorithms were classified, and you had to get your keys from NSA. Corporations never really bought into the idea of using classified algorithms dictated by the government. You'd think the NSA would have learned from this lesson and not even bothered with Clipper, Skipjack, and escrowed encryption chips.

25.2 National Computer Security Center (NCSC)

The National Computer Security Center, a branch of the NSA, is responsible for the government's trusted computer program. Currently, the center evaluates commercial security products (both hardware and software), sponsors and publishes research, develops technical guidelines, and generally provides advice, support, and training.

The NCSC publishes the infamous "Orange Book" [465]. Its actual title is the *Department of Defense Trusted Computer System Evaluation Criteria*, but that's a mouthful to say and the book has an orange cover. The Orange Book attempts to define security requirements, gives computer manufacturers an objective way to measure the security of their systems, and guides them as to what to build into their secure products. It focuses on computer security and doesn't really say a lot about cryptography.

The Orange Book defines four broad divisions of security protection. It also defines classes of protection within some of those divisions. They are summarized in Table 25.2.

Table 25.1
CCEP Modules

Application	Type I	Type II
Voice/low-speed data	Winster	Edgeshot
Computer	Tepache	Bulletproof
High-speed data	Foresee	Brushstroke
Next Generation	Countersign I	Countersign II

Sometimes manufacturers say things like "we have C2 security." This is what they're talking about. For more information on this, read [1365]. The computer security model used in these criteria is called the Bell-LaPadula model [100,101,102,103].

The NCSC has published a whole series of books on computer security, sometimes called the Rainbow Books (all the covers have different colors). For example, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria* [1146], sometimes called the "Red Book," interprets the Orange Book for networks and network equipment. The *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria* [1147]—I can't even begin to describe the color of that cover—does the same for databases. There are now over 30 of these books, some with hideously colored covers.

For a complete set of the Rainbow Books, write Director, National Security Agency, INFOSEC Awareness, Attention: C81, 9800 Savage Road, Fort George G. Meade, MD 20755-6000; (301) 766-8729. Don't tell them I sent you.

25.3 NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST)

The NIST is the National Institute of Standards and Technology, a division of the U.S. Department of Commerce. Formerly the NBS (National Bureau of Standards), it changed its name in 1988. Through its Computer Systems Laboratory (CSL), NIST promotes open standards and interoperability that it hopes will spur the economic development of computer-based industries. To this end, NIST issues standards and guidelines that it hopes will be adopted by all computer systems in the United States. Official standards are published as FIPS (Federal Information Processing Standards) publications.

If you want copies of any FIPS (or any other NIST publication), contact National Technical Information Service (NTIS), U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161; (703) 487-4650; or visit gopher://csrc.ncsl.nist.gov.

When Congress passed the Computer Security Act of 1987, NIST was mandated to define standards for ensuring the security of sensitive but unclassified informa-

Table 25.2
Orange Book Classifications

D: Minimal Security
C: Discretionary Protection
C1: Discretionary Security Protection
C2: Controlled Access Protection
B: Mandatory Protection
B1: Labeled Security Protection
B2: Structured Protection
B3: Security Domains
A: Verified Protection
A1: Verified Design

tion in government computer systems. (Classified information and Warner Amendment data are under the jurisdiction of the NSA.) The Act authorizes NIST to work with other government agencies and private industry in evaluating proposed technology standards.

NIST issues standards for cryptographic functions. U.S. government agencies are required to use them for sensitive but unclassified information. Often the private sector adopts these standards as well. NIST issued DES, DSS, SHS, and EES.

All these algorithms were developed with some help from the NSA, ranging from analyzing DES to designing DSS, SHS, and the Skipjack algorithm in EES. Some people have criticized NIST for allowing the NSA to have too much control over these standards, since the NSA's interests may not coincide with those of NIST. It is unclear how much actual influence NSA has on the design and development of the algorithms. Given NIST's limited staff, budget, and resources, NSA's involvement is probably considerable. NSA has significant resources to contribute, including a computer facility second-to-none.

The official "Memorandum of Understanding" (MOU) between the two agencies reads:

> MEMORANDUM OF UNDERSTANDING BETWEEN THE DIRECTOR OF THE NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY AND THE DIRECTOR OF THE NATIONAL SECURITY AGENCY CONCERNING THE IMPLEMENTATION OF PUBLIC LAW 100-235
>
> Recognizing that:
>
> A. Under Section 2 of the Computer Security Act of 1987 (Public Law 100-235), (the Act), the National Institute of Standards and Technology (NIST) has the responsibility within the Federal Government for:
>
> 1. Developing technical, management, physical, and administrative standards and guidelines for the cost-effective security and privacy of sensitive information in Federal computer systems as defined in the Act; and,
>
> 2. Drawing on the computer system technical security guidelines of the National Security Agency (NSA) in this regard where appropriate.
>
> B. Under Section 3 of the Act, the NIST is to coordinate closely with other agencies and offices, including the NSA, to assure:
>
> 1. Maximum use of all existing and planned programs, materials, studies, and reports relating to computer systems security and privacy, in order to avoid unnecessary and costly duplication of effort; and,
>
> 2. To the maximum extent feasible, that standards developed by the NIST under the Act are consistent and compatible with standards and procedures developed for the protection of classified information in Federal computer systems.
>
> C. Under the Act, the Secretary of Commerce has the responsibility, which he has delegated to the Director of NIST, for appointing the members of the Computer System Security and Privacy Advisory Board, at least one of whom shall be from the NSA.
>
> Therefore, in furtherance of the purposes of this MOU, the Director of the NIST and the Director of the NSA hereby agree as follows:

I. The NIST will:

1. Appoint to the Computer Security and Privacy Advisory Board at least one representative nominated by the Director of the NSA.

2. Draw upon computer system technical security guidelines developed by the NSA to the extent that the NIST determines that such guidelines are consistent with the requirements for protecting sensitive information in Federal computer systems.

3. Recognize the NSA-certified rating of evaluated trusted systems under the Trusted Computer Security Evaluation Criteria Program without requiring additional evaluation.

4. Develop telecommunications security standards for protecting sensitive unclassified computer data, drawing upon the expertise and products of the National Security Agency, to the greatest extent possible, in meeting these responsibilities in a timely and cost-effective manner.

5. Avoid duplication where possible in entering into mutually agreeable arrangements with the NSA for the NSA support.

6. Request the NSA's assistance on all matters related to cryptographic algorithms and cryptographic techniques including but not limited to research, development evaluation, or endorsement.

II. The NSA will:

1. Provide the NIST with technical guidelines in trusted technology, telecommunications security, and personal identification that may be used in cost-effective systems for protecting sensitive computer data.

2. Conduct or initiate research and development programs in trusted technology, telecommunications security, cryptographic techniques and personal identification methods.

3. Be responsive to the NIST's requests for assistance in respect to all matters related to cryptographic algorithms and cryptographic techniques including but not limited to research, development, evaluation, or endorsement.

4. Establish the standards and endorse products for application to secure systems covered in 10 USC Section 2315 (the Warner Amendment).

5. Upon request by Federal agencies, their contractors and other government-sponsored entities, conduct assessments of the hostile intelligence threat to federal information systems, and provide technical assistance and recommend endorsed products for application to secure systems against that threat.

III. The NIST and the NSA shall:

1. Jointly review agency plans for the security and privacy of computer systems submitted to NIST and NSA pursuant to section 6(b) of the Act.

2. Exchange technical standards and guidelines as necessary to achieve the purposes of the Act.

3. Work together to achieve the purposes of this memorandum with the greatest efficiency possible, avoiding unnecessary duplication of effort.

4. Maintain an on-going open dialogue to ensure that each organization remains abreast of emerging technologies and issues affecting automated information system security in computer-based systems.

5. Establish a Technical Working Group to review and analyze issues of mutual interest pertinent to protection of systems that process sensitive or other unclassified information. The Group shall be composed of six federal employees, three each selected by NIST and NSA and to be augmented as necessary by representatives of other agencies. Issues may be referred to the group by either the NSA Deputy Director for Information Security or the NIST Deputy Director or may be generated and addressed by the group upon approval by the NSA DDI or NIST Deputy Director. Within days of the referral of an issue to the Group by either the NSA Deputy Director for Information Security or the NIST Deputy Director, the Group will respond with a progress report and plan for further analysis, if any.

6. Exchange work plans on an annual basis on all research and development projects pertinent to protection of systems that process sensitive or other unclassified information, including trusted technology, for protecting the integrity and availability of data, telecommunications security and personal identification methods. Project updates will be exchanged quarterly, and project reviews will be provided by either party upon request of the other party.

7. Ensure the Technical Working Group reviews prior to public disclosure all matters regarding technical systems security techniques to be developed for use in protecting sensitive information in federal computer systems to ensure they are consistent with the national security of the United States. If NIST and NSA are unable to resolve such an issue within 60 days, either agency may elect to raise the issue to the Secretary of Defense and the Secretary of Commerce. It is recognized that such an issue may be referred to the President through the NSC for resolution. No action shall be taken on such an issue until it is resolved.

8. Specify additional operational agreements in annexes to this MOU as they are agreed to by NSA and NIST.

IV. Either party may elect to terminate this MOU upon six months' written notice. This MOU is effective upon approval of both signatories.

/signed/

RAYMOND G. KAMMER

Acting Director, National Institute of Standards and Technology, 24 March 1989

W. O. STUDEMAN

Vice Admiral, U.S. Navy; Director, National Security Agency, 23 March 1989

25.4 RSA Data Security, Inc.

RSA Data Security, Inc. (RSADSI) was founded in 1982 to develop, license, and market the RSA patent. It has some commercial products, including a standalone e-mail security package, and various cryptographic libraries (available in either source or object form). RSADSI also markets the RC2 and RC4 symmetric algorithms (see Section 11.8). RSA Laboratories, a research lab associated with RSADSI, performs basic cryptographic research and provides consulting services.

Anyone interested in either their patents or products should contact Director of Sales, RSA Data Security, Inc., 100 Marine Parkway, Redwood City, CA 94065; (415) 595-8782; fax: (415) 595-1873.

25.5 PUBLIC KEY PARTNERS

The five patents in Table 25.3 are held by Public Key Partners (PKP) of Sunnyvale, California, a partnership between RSADSI and Caro-Kahn, Inc.—the parent company of Cylink. (RSADSI gets 65 percent of the profits and Caro-Kahn gets 35 percent.) PKP claims that these patents, and 4,218,582 in particular, apply to *all uses* of public-key cryptography.

In [574], PKP wrote:

> These patents [4,200,770, 4,218,582, 4,405,829, and 4,424,414] cover all known methods of practicing the art of Public Key, including the variations collectively known as ElGamal.
>
> Due to the broad acceptance of RSA digital signatures throughout the international community, Public Key Partners strongly endorses its incorporation in a digital signature standard. We assure all interested parties that Public Key Partners will comply with all of the policies of ANSI and the IEEE concerning the availability of licenses to practice this art. Specifically, in support of any RSA signature standard which may be adopted, Public Key Partners hereby gives its assurance that licenses to practice RSA signatures will be available under reasonable terms and conditions on a nondiscriminatory basis.

Whether this is true depends on who you talk to. PKP's licenses have mostly been secret, so there is no way to check if the licenses are standard. Although they claim to have never denied a license to anyone, at least two companies claim to have been denied a license. PKP guards its patents closely, threatening anyone who tries to use public-key cryptography without a license. In part, this is a reaction to U.S. patent law. If you hold a patent and fail to prosecute an infringement, you can lose your patent. There has been much talk about whether the patents are legal, but so far it has all been talk. All legal challenges to PKP's patents have been settled before judgment.

Table 25.3
Public Key Partners' Patents

Patent #	Date	Inventors	Patent Covers
4,200,770	3/29/80	Hellman, Diffie, Merkle	Diffie-Hellman Key Exchange
4,218,582	8/19/80	Hellman, Merkle	Merkle-Hellman Knapsacks
4,405,829	9/20/83	Rivest, Shamir, Adleman	RSA
4,424,414	3/3/84	Hellman, Pohlig	Pohlig-Hellman
4,995,082	2/19/91	Schnorr	Schnorr Signatures

I am not going to dispense legal advice in this book. Maybe the RSA patent will not hold up in court. Maybe the patents do not apply to the entirety of public-key cryptography. (Honestly, I can't see how they cover ElGamal or elliptic curve cryptosystems.) Perhaps someone will eventually win a suit against PKP or RSADSI. But keep in mind that corporations with large legal departments like IBM, Microsoft, Lotus, Apple, Novell, Digital, National Semiconductor, AT&T, and Sun have all licensed RSA for use in their products rather than fight them in court. And Boeing, Shell Oil, DuPont, Raytheon, and Citicorp have all licensed RSA for their own internal use.

In one case, PKP brought suit against TRW Corporation for using the ElGamal algorithm without a license. TRW claimed they did not need a license. PKP and TRW reached a settlement in June 1992. The details of the settlement are unknown, but they included an agreement by TRW to license the patents. This does not bode well. TRW can afford good lawyers; I can only assume that if they thought they could win the suit without spending an unreasonable amount of money, they would have fought.

Meanwhile, PKP is having its own internal problems. In June 1994 Caro-Kahn sued RSADSI alleging, among other things, that the RSA patent is invalid and unenforceable [401]. Both partners are trying to have the partnership dissolved. Are the patents valid or not? Will users have to get a license from Caro-Kahn to use the RSA algorithm? Who will own the Schnorr patent? The matter will probably be sorted out by the time this book sees publication.

Patents are good for only 17 years, and cannot be renewed. On March 29, 1997, Diffie-Hellman key exchange (and the ElGamal algorithm) will enter the public domain. On September 20, 2000, RSA will enter the public domain. Mark your calendars.

25.6 International Association for Cryptologic Research (IACR)

The International Association for Cryptologic Research is the worldwide cryptographic research organization. Its stated purpose is to advance the theory and practice of cryptology and related fields. Membership is open to any person. The association sponsors two annual conferences, Crypto (held in Santa Barbara in August) and Eurocrypt (held in Europe in May), and publishes quarterly *The Journal of Cryptology* and the *IACR Newsletter.*

The address of the IACR Business Office changes whenever the president does. The current address is: IACR Business Office, Aarhus Science Park, Gustav Wieds Vej 10, DK-8000 Aarhus C, Denmark.

25.7 RACE Integrity Primitives Evaluation (RIPE)

The Research and Development in Advanced Communication Technologies in Europe (RACE) program was launched by the European Community to support pre-

competitive and pre-normative work in communications standards and technologies to support Integrated Broadband Communication (IBC). As part of that effort, RACE established the RACE Integrity Primitives Evaluation (RIPE) to put together a portfolio of techniques to meet the anticipated security requirements of IBC.

Six leading European cryptography research groups made up the RIPE consortium: Center for Mathematics and Computer Science, Amsterdam; Siemens AG; Philips Crypto BV; Royal PTT Nederland NV, PTT Research; Katholieke Universiteit Leuven; and Aarhus Universitet. After calls for algorithms in 1989 and 1991 [1564], 32 submissions from around the world, and a 350 man-month evaluation project, the consortium published *RIPE Integrity Primitives* [1305,1332]. The report included an introduction and some basic integrity concepts, and these primitives: MDC-4 (see Section 14.11), RIPE-MD (see Section 14.8), RIPE-MAC (see Section 14.14), IBC-HASH, SKID (see Section 3.2), RSA, COMSET (see Section 16.1), and RSA key generation.

25.8 CONDITIONAL ACCESS FOR EUROPE (CAFE)

Conditional Access for Europe (CAFE) is a project in the European Community's ESPRIT program [204,205]. Work began in December 1992 and is scheduled to be finished by the end of 1995. The consortium involved consists of groups for social and market studies (Cardware, Institut für Sozialforschung), software and hardware manufacturers (DigiCash, Gemplus, Ingenico, Siemens), and cryptographers (CWI Amsterdam, PTT Research Netherlands, SPET, Sintef Delab Trondheim, Universities of Århus, Hildesheim and Leuven).

The goal is to develop systems for conditional access, particularly digital payment systems. Payment systems must give legal certainty to everybody at all times and require as little trust as possible—this certainty should not depend on the tamper-resistance of any devices.

The basic device for CAFE is an electronic wallet: a small computer that looks something like a pocket calculator. It has a battery, keyboard, screen, and an infrared channel for communicating with other wallets. Every user owns and uses his own wallet, which administers his rights and guarantees his security.

A device with a keyboard and screen has an advantage over a smart card; it can operate independent of a terminal. A user can directly enter his password and the amount of the payment. The user does not have to give his wallet up to complete a transaction, unlike the current situation with credit cards.

Additional features are:

- Offline transactions. The purpose of the system is to replace small cash transactions; an online system would be too cumbersome.
- Loss tolerance. If a user loses his wallet, or if it breaks or is stolen, he can recover his money.
- Support for different currencies.

— An open architecture and open system. A user should be able to pay for arbitrary services, such as shopping, telephone, and public transport, by a range of service providers. The system should be interoperable between any number of electronic money issuers, and between different wallet types and manufacturers.

— Low cost.

At this writing there is a software version of the system, and the consortium is hard at work on a hardware prototype.

25.9 ISO/IEC 9979

In the mid-80s, the ISO tried to standardize DES, which by then was already a FIPS and an ANSI standard. After some political wrangling, the ISO decided not to standardize cryptographic algorithms, but instead to register algorithms. Only encryption algorithms can be registered; hash functions and signature schemes cannot. Any national body can submit an algorithm for registration.

Currently only three algorithms have been submitted (see Table 25.4). A submission includes information about applications, parameters, implementations, modes, and test vectors. A detailed description is optional; it is possible to submit secret algorithms for registration.

The fact that an algorithm is registered does not imply anything about its quality, nor is registration an approval of the algorithm by the ISO/IEC. Registration merely indicates that a single national body wants to register the algorithm, based on whatever criteria that body uses.

I am not impressed with this idea. Registration obstructs the standardization process. Rather than agreeing on a few algorithms, the ISO is allowing any algorithm to be registered. With so little control over what is registered, stating that an algorithm is "ISO/IEC 9979 Registered" sounds a whole lot better than it is. In any case, the registry is maintained by the National Computer Centre Ltd., Oxford Road, Manchester, M1 7ED, United Kingdom.

Table 25.4
ISO/IEC 9979
Registered Algorithms

Registration Number	Name
0001	B-CRYPT
0002	IDEA
0003	LUC

25.10 PROFESSIONAL, CIVIL LIBERTIES, AND INDUSTRY GROUPS

Electronic Privacy Information Center (EPIC)

EPIC was established in 1994 to focus public attention on emerging privacy issues relating to the National Information Infrastructure, such as the Clipper chip, the Digital Telephony proposal, national identity numbers and systems, medical records privacy, and the sale of consumer data. EPIC conducts litigation, sponsors conferences, produces reports, publishes the *EPIC Alert*, and leads campaigns on privacy issues. Anyone interested in joining should contact Electronic Privacy Information Center, 666 Pennsylvania Avenue SE, Suite 301, Washington, D.C. 20003; (202) 544-9240; fax: (202) 547-5482; Internet: info@epic.org.

Electronic Frontier Foundation (EFF)

The EFF is dedicated to protecting civil rights in cyberspace. With respect to cryptographic policy in the United States, they believe that information and access to cryptography are fundamental rights, and therefore should be free of government restriction. They organized the Digital Privacy and Security Working Group, a coalition of 50 organizations. The group opposed the Digital Telephony bill and the Clipper initiative. The EFF is also helping in a lawsuit against cryptography export controls [143]. Anyone interested in joining the EFF should contact Electronic Frontier Foundation, 1001 G Street NW, Suite 950E, Washington, D.C. 20001; (202) 347-5400; fax: (202) 393-5509; Internet: eff@eff.org.

Association for Computing Machinery (ACM)

The ACM is an international computer industry organization. In 1994 the U.S. ACM Public Policy Committee produced an excellent report on U.S. cryptography policy [935]. This should be required reading for anyone interested in the politics of cryptography. It is available via anonymous ftp from info.acm.org in /reports/acm_crypt_study/acm_crypto_study.ps.

Institute of Electrical and Electronics Engineers (IEEE)

The IEEE is another professional organization. The U.S. office investigates and makes recommendations on privacy-related issues including encryption policy, identity numbers, and privacy protections on the Internet.

Software Publishers Association (SPA)

The SPA is a trade association of over 1000 personal computer software companies. They have lobbied for relaxation of export controls on cryptography, and maintain a list of commercially available foreign cryptography products.

25.11 SCI.CRYPT

Sci.crypt is the Usenet newsgroup for cryptology. It is read by an estimated 100,000 people worldwide. Most of the posts are nonsense, bickering, or both; some are

political, and most of the rest are requests for information or basic questions. Occasionally nuggets of new and useful information are posted to this newsgroup. If you follow sci.crypt regularly, you will learn how to use something called a kill file.

Another Usenet newsgroup is sci.crypt.research, a moderated newsgroup devoted to discussions about cryptology research. There are fewer posts and they are more interesting.

25.12 Cypherpunks

The Cypherpunks are an informal group of people interested in teaching and learning about cryptography. They also experiment with cryptography and try to put it into use. In their opinion, all the cryptographic research in the world doesn't do society any good unless it gets used.

In "A Cypherpunk's Manifesto," Eric Hughes writes [744]:

> We the Cypherpunks are dedicated to building anonymous systems. We are defending our privacy with cryptography, with anonymous mail forwarding systems, with digital signatures, and with electronic money.
>
> Cypherpunks write code. We know that someone has to write software to defend privacy, and since we can't get privacy unless we all do, we're going to write it. We publish our code so that our fellow Cypherpunks may practice and play with it. Our code is free for all to use, worldwide. We don't care much if you don't approve of the software we write. We know that software can't be destroyed and that widely dispersed systems can't be shut down.

People interested in joining the cypherpunks mailing list on the Internet should send mail to majordomo@toad.com. The mailing list is archived at ftp.csua.berkeley.edu in /pub/cypherpunks.

25.13 Patents

Software patents are an issue much larger than the scope of this book. Whether they're good or bad, they exist. Algorithms, cryptographic algorithms included, can be patented in the United States. IBM owned the DES patents [514]. IDEA is patented. Almost every public-key algorithm is patented. NIST even has a patent for the DSA. Some cryptography patents have been blocked by intervention from the NSA, under the authority of the Invention Secrecy Act of 1940 and the National Security Act of 1947. This means that instead of a patent, the inventor gets a secrecy order and is prohibited from discussing his invention with anybody.

The NSA has special dispensation when it comes to patents. They can apply for a patent and then block its issuance. It's a secrecy order again, but here the NSA is both the inventor and the issuer of the order. When, at some later date, the secrecy order is removed, the Census Office issues the patent good for the standard 17 years. This rather clearly protects the invention while keeping it secret. If someone else

invents the same thing, the NSA has already filed for the patent. If no one else invents it, then it remains secret.

Not only does this fly directly in the face of the patent process, which is supposed to disclose as well as protect inventions, it allows the NSA to keep a patent for more than 17 years. The 17-year clock starts ticking after the patent is issued, not when it is filled. How this will change, now that the United States has ratified the GATT treaty, is unclear.

25.14 U.S. EXPORT RULES

According to the U.S. government, cryptography can be a munition. This means it is covered under the same rules as a TOW missile or an M1 Abrams tank. If you sell cryptography overseas without the proper export license, then you are an international arms smuggler. Unless you think time in a federal penitentiary would look good on your résumé, pay attention to the rules.

With the advent of the Cold War in 1949, all of the NATO countries (except Iceland), and later Australia, Japan, and Spain, formed CoCom, the Coordinating Committee for Multilateral Export Controls. This is an unofficial nontreaty organization, chartered to coordinate national restrictions on the export of sensitive military technologies to the Soviet Union, other Warsaw Pact countries, and the People's Republic of China. Examples of controlled technologies are computers, milling machinery, and cryptography. The goal here was to slow technology transfer into those countries, and thereby keep their militaries inferior.

Since the end of the Cold War, the CoCom countries realized that many of their controls were obsolete. They are supposedly in the process of defining something called the "New Forum," another multinational organization designed to stop the flow of military technologies to countries the members don't particularly like.

In any case, U.S. export policy on strategic goods is defined by the Export Administration Act, the Arms Export Control Act, the Atomic Energy Act, and the Nuclear Non-Proliferation Act. The controls established by all this legislation are implemented through a number of statutes, none of them coordinated with each other. Over a dozen agencies including the military services administer controls; often their regulatory programs overlap and contradict.

Controlled technologies appear on several lists. Cryptography has traditionally been classified as a munition and appears on the U.S. Munitions List (USML), the International Munitions List (IML), the Commerce Control List (CCL), and the International Industrial List (IIL). The Department of State is responsible for the USML; it is published as part of the International Traffic in Arms Regulations (ITAR) [466,467].

Two U.S. government agencies control export of cryptography. One is the Bureau of Export Administration (BXA) in the Department of Commerce, authorized by the Export Administration Regulations (EAR). The other is the Office of Defense Trade Controls (DTC) in the State Department, authorized by the ITAR. As a rule of thumb, the Commerce Department's BXA has far less stringent requirements, but

State Department's DTC (which takes technical and national security advice from the NSA, and always seems to follow that advice) sees all cryptography exports first and can refuse to transfer jurisdiction to BXA.

The ITAR regulates this stuff. (Before 1990 the Office of Defense Trade Controls was called the Office of Munitions Controls; presumably this public relations effort is designed to help us forget that we're dealing with guns and bombs.) Historically, the DTC has been reluctant to grant export licenses for encryption products stronger than a certain level—not that they have ever been public about exactly what that level is.

The following sections are excerpted from the ITAR [466,467]:

§ 120.10 Technical data.

Technical data means, for purposes of this subchapter:

(1) Information, other than software as defined in 120.10(d), which is required for the design, development, production, processing, manufacture, assembly, operation, repair, maintenance or modification of defense articles. This includes, for example, information in the form of blueprints, drawings, photographs, plans, instructions and documentation;

(2) Classified information relating to defense articles and defense services;

(3) Information covered by an invention secrecy order;

(4) Software as defined in Sec. 121.8(f) directly related to defense articles;

(5) This definition does not include information concerning general scientific, mathematical or engineering principles commonly taught in schools, colleges and universities in the public domain as defined in § 120.11. It also does not include basic marketing information on function or purpose or general system descriptions of defense articles.

§ 120.11 Public domain.

Public domain means information which is published and which is generally accessible or available to the public:

(1) Through sales at newsstands and bookstores;

(2) Through subscriptions which are available without restriction to any individual who desires to obtain or purchase the published information;

(3) Through second class mailing privileges granted by the U.S. Government;

(4) At libraries open to the public or from which the public can obtain documents;

(5) Through patents available at any patent office;

(6) Through unlimited distribution at a conference, meeting, seminar, trade show or exhibition, generally accessible to the public, in the United States;

(7) Through public release (i.e., unlimited distribution) in any form (e.g., not necessarily in published form) after approval by the cognizant U.S. government department or agency (see also § 125.4(b)(13)).

(8) Through fundamental research in science and engineering at accredited institutions of higher learning in the U.S., where the resulting information is ordinarily published and shared broadly in the scientific community. Fundamental research is defined to mean basic and applied research in science and engineering where the

resulting information is ordinarily published and shared broadly within the scientific community, as distinguished from research the results of which are restricted for proprietary reasons or specific U.S. Government access and dissemination controls. University research will not be considered fundamental research if:

(i) The University or its researchers accept other restrictions on publication of scientific and technical information resulting from the project or activity, or

(ii) The research is funded by the U.S. Government and specific access and dissemination controls protecting information resulting from the research are applicable.

§ 120.17 Export.

Export means:

(1) Sending or taking defense articles out of the United States in any manner, except by mere travel outside of the United States by a person whose personal knowledge includes technical data; or

(2) Transferring registration, control or ownership to a foreign person of any aircraft, vessel, or satellite covered by the U.S. Munitions List, whether in the United States or abroad; or

(3) Disclosing (including oral or visual disclosure) or transferring in the United States any defense articles to an embassy, any agency or subdivision of a foreign government (e.g., diplomatic missions); or

(4) Disclosing (including oral or visual disclosure) or transferring technical data to a foreign person, whether in the United States or abroad; or

(5) Performing a defense service on behalf of, or for the benefit of, a foreign person, whether in the United States or abroad.

(6) A launch vehicle or payload shall not, by the launching of such vehicle, be considered export for the purposes of this subchapter. However, for certain limited purposes (see § 126.1 of this subchapter), the controls of this subchapter apply to sales and other transfers of defense articles or defense services.

Part 121—The United States Munitions List

§ 121.1 General. The United States Munitions List

Category XIII—Auxiliary Military Equipment

(1) Cryptographic (including key management) systems, equipment, assemblies, modules, integrated circuits, components or software with the capability of maintaining secrecy or confidentiality of information or information systems, except cryptographic equipment and software as follows:

(i) Restricted to decryption functions specifically designed to allow the execution of copy protected software, provided the decryption functions are not user-accessible.

(ii) Specifically designed, developed or modified for use in machines for banking or money transactions, and restricted to use only in such transactions. Machines for banking or money transactions include automatic teller machines, self-service statement printers, point of sale terminals or equipment for the encryption of interbanking transactions.

(iii) Employing only analog techniques to provide the cryptographic processing that ensures information security in the following applications. . . .

(iv) Personalized smart cards using cryptography restricted for use only in equipment or systems exempted from the controls of the USML.

(v) Limited to access control, such as automatic teller machines, self-service statement printers or point of sale terminals, which protects passwords or personal identification numbers (PIN) or similar data to prevent unauthorized access to facilities but does not allow for encryption or files or text, except as directly related to the password of PIN protection.

(vi) Limited to data authentication which calculates a Message Authentication Code (MAC) or similar result to ensure no alteration of text has taken place, or authenticate users, but does not allow for encryption of data, text or other media other than that needed for the authentication.

(vii) Restricted for fixed data compression or coding techniques.

(viii) Limited to receiving for radio broadcast, pay television or similar restricted audience television of the consumer type, without digital encryption and where digital decryption is limited to video, audio or management functions.

(ix) Software designed or modified to protect against malicious computer damage, (e.g., viruses).

(2) Cryptographic (including key management) systems, equipment, assemblies, modules, integrated circuits, components or software which have the capability of generating spreading or hopping codes for spread spectrum systems or equipment.

(3) Cryptographic systems, equipment, assemblies, modules, integrated circuits, components or software.

§ 125.2 Exports of unclassified technical data.

(a) General. A license (DSP-5) is required for the export of unclassified technical data unless the export is exempt from the licensing requirements of this subchapter. In the case of a plant visit, details of the proposed discussions must be transmitted to the Office of Defense Trade Controls for an appraisal of the technical data. Seven copies of the technical data or the details of the discussions must be provided.

(b) Patents. A license issued by the Office of Defense Trade Controls is required for the export of technical data whenever the data exceeds that which is used to support a domestic filing of a patent application or to support a foreign filing of a patent application whenever no domestic application has been filed. Requests for the filing of patent applications in a foreign country, and requests for the filing of amendments, modifications or supplements to such patents, should follow the regulations of the U.S. Patent and Trademark Office in accordance with 37 CFR part 5. The export of technical data to support the filing and processing of patent applications in foreign countries is subject to regulations issued by the U.S. Patent and Trademark Office pursuant to 35 U.S.C. 184.

(c) Disclosures. Unless otherwise expressly exempted in this subchapter, a license is required for the oral, visual or documentary disclosure of technical data

> by U.S. persons to foreign persons. A license is required regardless of the manner in which the technical data is transmitted (e.g., in person, by telephone, correspondence, electronic means, etc.). A license is required for such disclosures by U.S. persons in connection with visits to foreign diplomatic missions and consular offices.

And so on. There's a lot more information in this document. If you're going to try to export cryptography, I suggest you get a copy of the entire thing and a lawyer who speaks the language.

In reality, the NSA has control over the export of cryptographic products. If you want a Commodity Jurisdiction (CJ), you must submit your product to the NSA for approval and submit the CJ application to the State Department. After State Department approval, the matter moves under the jurisdiction of the Commerce Department, which has never cared much about the export of cryptography. However, the State Department will never grant a CJ without NSA approval.

In 1977 an NSA employee named Joseph A. Meyer wrote a letter—unauthorized, according to the official story of the incident—to the IEEE, warning them that the scheduled presentation of the original RSA paper would violate the ITAR. From *The Puzzle Palace:*

> He had a point. The ITAR did cover any "unclassified information that can be used, or adapted for use, in the design, production, manufacture, repair, overhaul, processing, engineering, development, operation, maintenance, or reconstruction" of the listed materials, as well as "any technology which advances the state-of-the-art or establishes a new art in an area of significant military applicability in the United States." And export did include transferring the information both by writing and by either oral or visual means, including briefings and symposia in which foreign nationals are present.
>
> But followed literally, the vague, overly broad regulations would seem to require that anyone planning to write or speak out publicly on a topic touching the Munitions List must first get approval from the State Department—a chilling prospect clearly at odds with the First Amendment and one as yet untested by the Supreme Court.

In the end NSA disavowed Meyer's actions and the RSA paper was presented as planned. No actions were taken against any of the inventors, although their work arguably enhanced foreign cryptography capabilities more than anything released since.

The following statement by NSA discusses the export of cryptography [363]:

> Cryptographic technology is deemed vital to national security interests. This includes economic, military, and foreign policy interests.
>
> We do not agree with the implications from the House Judiciary Committee hearing of 7 May 1992 and recent news articles that allege that U.S. export laws prevent U.S. firms' manufacture and use of top encryption equipment. We are unaware of any case where a U.S. firm has been prevented from manufacturing and using encryption equipment within this country or for use by the U.S. firm or

its subsidiaries in locations outside the U.S. because of U.S. export restrictions. In fact, NSA has always supported the use of encryption by U.S. businesses operating domestically and overseas to protect sensitive information.

For export to foreign countries, NSA as a component of the Department of Defense (along with the Department of State and the Department of Commerce) reviews export licenses for information security technologies controlled by the Export Administration Regulations or the International Traffic in Arms Regulations. Similar export control systems are in effect in all the Coordinating Committee for Multilateral Export Controls (CoCom) countries as well as many non-CoCom countries as these technologies are universally considered as sensitive. Such technologies are not banned from export and are reviewed on a case-by-case basis. As part of the export review process, licenses may be required for these systems and are reviewed to determine the effect such export could have on national security interests—including economic, military, and political security interests. Export licenses are approved or denied based upon the type of equipment involved, the proposed end use and the end user.

Our analysis indicates that the U.S. leads the world in the manufacture and export of information security technologies. Of those cryptologic products referred to NSA by the Department of State for export licenses, we consistently approve over 90%. Export licenses for information security products under the jurisdiction of the Department of Commerce are processed and approved without referral to NSA or DoD. This includes products using such techniques as the DSS and RSA which provide authentication and access control to computers or networks. In fact, in the past NSA has played a major role in successfully advocating the relaxation of export controls on RSA and related technologies for authentication purposes. Such techniques are extremely valuable against the hacker problem and unauthorized use of resources.

It is the stated policy of the NSA not to restrict the export of authentication products, only encryption products. If you want to export an authentication-only product, approval may merely be a matter of showing that your product cannot easily be used for encryption. Furthermore, the bureaucratic procedures are much simpler for authentication products than for encryption products. An authentication product needs State Department approval only once for a CJ; an encryption product may require approval for every product revision or even every sale.

Without a CJ, you must request export approval every time you wish to export the product. The State Department does not approve the export of products with strong encryption, even those using DES. Isolated exceptions include export to U.S. subsidiaries for the purposes of communicating to the U.S., exports for some banking applications, and export to appropriate U.S. military users. The Software Publishers Association (SPA) has been negotiating with the government to ease export license restrictions. A 1992 agreement between them and the State Department eased the export license rules for two algorithms, RC2 and RC4, as long as the key size is 40 bits or less. Refer to Section 7.1 for more information.

In 1993, Rep. Maria Cantwell (D-WA) introduced a bill at the behest of the software industry to relax export controls on encryption software. Sen. Patty Murray

(D-WA) introduced a companion bill in the Senate. The Cantwell Bill was appended to the general export control legislation going through Congress, but was deleted by the House Intelligence Committee after a massive lobbying effort by the NSA. Whatever the NSA did, it was impressive; the committee voted unanimously to delete the wording. I can't remember the last time a bunch of legislators voted unanimously to do anything.

In 1995 Dan Bernstein, with the help of the EFF, sued the U.S. government, seeking to bar the government from restricting publication of cryptographic documents and software [143]. The suit claimed that the export control laws are unconstitutional, an "impermissible prior restraint on speech, in violation of the First Amendment." Specifically, the lawsuit charges that the current export control process:

- Allows bureaucrats to restrict publication without ever going to court.
- Provides too few procedural safeguards for First Amendment rights.
- Requires publishers to register with the government, creating in effect a "licensed press."
- Disallows general publication by requiring recipients to be individually identified.
- Is sufficiently vague that ordinary people cannot know what conduct is allowed and what conduct is prohibited.
- Is overbroad because it prohibits conduct that is clearly protected (such as speaking to foreigners within the United States).
- Is applied too broadly, by prohibiting export of software that contains no cryptography, on the theory that cryptography could be added to it later.
- Egregiously violates the First Amendment by prohibiting private speech on cryptography because the government wishes its own opinions on cryptography to guide the public instead.
- Exceeds the authority granted by Congress in the export control laws in many ways, as well as exceeding the authority granted by the Constitution.

Everyone anticipates that the case will take several years to settle, and no one has any idea how it will come out.

Meanwhile, the Computer Security and Privacy Advisory Board, an official advisory board to NIST, voted in March 1992 to recommend a national policy review of cryptographic issues, including export policy. They said that export policy is decided solely by agencies concerned with national security, without input from agencies concerned with encouraging commerce. Those agencies concerned with national security are doing everything possible to make sure this doesn't change, but eventually it has to.

25.15 Foreign Import and Export of Cryptography

Other countries have their own import and export rules [311]. This summary is incomplete and probably out of date. Countries could have rules and ignore them, or could have no rules but restrict import, export, and use anyway.

- Australia requires an import certificate for cryptography only upon request from the exporting country.
- Canada has no import controls, and export controls are similar to those of the United States. The exportation of items from Canada may be subject to restriction if they are included on the Export Control List pursuant to the Export and Import Permits Act. Canada follows the CoCom regulations in the regulation of cryptographic technology. Encryption devices are outlined in category five, part two of Canada's export regulations. These provisions are similar to U.S. category five in the Export Administration Regulations.
- China has a licensing scheme for importing commodities; exporters must file an application with the Ministry of Foreign Trade. Based on China's List of Prohibited and Restricted Imports and Exports enacted in 1987, China restricts the import and export of voice-encoding devices.
- France has no special rules for the import of cryptography, but they have rules regarding the sale and use of cryptography in their country. All products must be certified: Either they must meet a published specification, or the company proprietary specification must be provided to the government. The government may also ask for two units for their own use. Companies must have a license to sell cryptography within France; the license specifies the target market. Users must have a license to buy and use cryptography; the license includes a statement to the effect that users must be prepared to give up their keys to the government up to four months after use. This restriction may be waived in some cases: for banks, large companies, and so on. And there is no use license requirement for cryptography exportable from the U.S.
- Germany follows the CoCom guidelines, requiring a license to export cryptography. They specifically maintain control of public-domain and mass-market cryptography software.
- Israel has import restrictions, but no one seems to know what they are.
- Belgium, Italy, Japan, Netherlands, and the United Kingdom follow the CoCom guidelines on cryptography, requiring a license for export.
- Brazil, India, Mexico, Russia, Saudi Arabia, Spain, South Africa, Sweden, and Switzerland have no import or export controls on cryptography.

25.16 LEGAL ISSUES

Are digital signatures real signatures? Will they stand up in court? Some preliminary legal research has resulted in the opinion that digital signatures would meet the requirements of legally binding signatures for most purposes, including commercial use as defined in the Uniform Commercial Code (UCC). A GAO (General Accounting Office) decision, made at the request of NIST, opines that digital signatures will meet the legal standards of handwritten signatures [362].

The Utah Digital Signature Act went into effect on May 1, 1995, providing a legal framework for the use of digital signatures in the judicial system. California has a bill pending, while Oregon and Washington are still writing theirs. Texas and Florida are right behind. By this book's publication, more states will have followed suit.

The American Bar Association (EDI and Information Technology Division of the Science and Technology Section) produced a model act for states to use for their own legislation. The act attempts to incorporate digital signatures into the existing legal infrastructure for signatures: the Uniform Commercial Code, the United States Federal Reserve regulations, common law of contracts and signatures, the United Nations Convention on Contracts for the International Sale of Goods, and the United Nations Convention on International Bills of Exchange and International Promissory Committees. Included in the act are responsibilities and obligations of certification authorities, issues of liability, and limits and policies.

In the United States, laws about signatures, contracts, and commercial transactions are state laws, so this model act is designed for states. The eventual goal is a federal act, but if this all begins at the state level there is less chance of the NSA mucking up the works.

Even so, the validity of digital signatures has not been challenged in court; their legal status is still undefined. In order for digital signatures to carry the same authority as handwritten signatures, they must first be used to sign a legally binding document, and then be challenged in court by one party. The court would then consider the security of the signature scheme and issue a ruling. Over time, as this happened repeatedly, a body of precedent rulings would emerge regarding which digital signature methods and what key sizes are required for a digital signature to be legally binding. This is likely to take years.

Until then, if two people wish to use digital signatures for contracts (or purchase requests, or work orders, or whatever), it is recommended that they sign a paper contract in which they agree in the future to be bound by any documents digitally signed by them [1099]. This document would specify algorithm, key size, and any other parameters; it should also delineate how disputes would be resolved.

Afterword by Matt Blaze

One of the most dangerous aspects of cryptology (and, by extension, of this book), is that you can almost measure it. Knowledge of key lengths, factoring methods, and cryptanalytic techniques makes it possible to estimate (in the absence of a real theory of cipher design) the "work factor" required to break a particular cipher. It's all too tempting to misuse these estimates as if they were overall security metrics for the systems in which they are used. The real world offers the attacker a richer menu of options than mere cryptanalysis. Often more worrisome are protocol attacks, Trojan horses, viruses, electromagnetic monitoring, physical compromise, blackmail and intimidation of key holders, operating system bugs, application program bugs, hardware bugs, user errors, physical eavesdropping, social engineering, and dumpster diving, to name just a few.

High-quality ciphers and protocols are important tools, but by themselves make poor substitutes for realistic, critical thinking about what is actually being protected and how various defenses might fail (attackers, after all, rarely restrict themselves to the clean, well-defined threat models of the academic world). Ross Anderson gives examples of cryptographically strong systems (in the banking industry) that fail when exposed to the threats of the real world [43,44]. Even when the attacker has access only to ciphertext, seemingly minor breaches in other parts of the system can leak enough information to render good cryptosystems useless. The Allies in World War II broke the German Enigma traffic largely by carefully exploiting operator errors [1587].

An NSA-employed acquaintance, when asked whether the government can crack DES traffic, quipped that real systems are so insecure that they never need to bother. Unfortunately, there are no easy recipes for making a system secure, no substitute for careful design and critical, ongoing scrutiny. Good cryptosystems have the nice property of making life much harder for the attacker than for the legitimate user; this is not the case for almost every other aspect of computer and communication

security. Consider the following (quite incomplete) "Top Ten Threats to Security in Real Systems" list; all are easier to exploit than to prevent.

1. The sorry state of software. Everyone knows that nobody knows how to write software. Modern systems are complex, with hundreds of thousands of lines of code; any one of them has the chance to compromise security. Fatal bugs may even be far-removed from the security portion of the software.
2. Ineffective protection against denial-of-service attacks. Some cryptographic protocols allow anonymity. It may be especially dangerous to deploy anonymous protocols if they increase the opportunities for unidentified vandals to disrupt service; anonymous systems therefore need to be especially resistant to denial-of-service attacks. Robust networks can more easily support anonymity; consider that hardly anyone worries very much about the millions of anonymous entry points to more robust networks like the telephone system or the postal service, where it's relatively difficult (or expensive) for an individual to cause large-scale failures.
3. No place to store secrets. Cryptosystems protect large secrets with smaller ones (keys). Unfortunately, modern computers aren't especially good at protecting even the smallest secrets. Multi-user networked workstations can be broken into and their memories compromised. Standalone, single-user machines can be stolen or compromised through viruses that leak secrets asynchronously. Remote servers, where there may be no user available to enter a passphrase (but see threat #5), are an especially hard problem.
4. Poor random-number generation. Keys and session variables need good sources of unpredictable bits. A running computer has a lot of entropy in it but rarely provides applications with a convenient or reliable way to exploit it. A number of techniques have been proposed for getting true random numbers in software (taking advantage of unpredictability in things like I/O interarrival timing, clock and timer skew, and even air turbulence inside disk enclosures), but all these are very sensitive to slight changes in the environments in which they are used.
5. Weak passphrases. Most cryptographic software addresses the key storage and key generation problems by relying on user-generated passphrase strings, which are presumed to be unpredictable enough to produce good key material and are also easy enough to remember that they do not require secure storage. While dictionary attacks are a well-known problem with short passwords, much less is known about lines of attack against user-selected passphrase-based keys. Shannon tells us that English text has only just over 1 bit of entropy per character, which would seem to leave most passphrases well within reach of brute-force search. Less is known, however, about good techniques for enumerating passphrases in order to exploit this. Until we have a better understanding of how to attack passphrases, we really have no idea how weak or strong they are.

6. Mismatched trust. Almost all currently available cryptographic software assumes that the user is in direct control over the systems on which it runs and has a secure path to it. For example, the interfaces to programs like PGP assume that their passphrase input always comes from the user over a secure path like the local console. This is not always the case, of course; consider the problem of reading your encrypted mail when logged in over a network connection. What the system designer assumes is trusted may not match the needs or expectations of the real users, especially when software can be controlled remotely over insecure networks.
7. Poorly understood protocol and service interactions. As systems get bigger and more complex, benign features frequently come back to haunt us, and it's hard to know even where to look when things fail. The Internet worm was propagated via an obscure and innocent-looking feature in the sendmail program; how many more features in how many more programs have unexpected consequences just waiting to be discovered?
8. Unrealistic threat and risks assessment. Security experts tend to focus on the threats they know how to model and prevent. Unfortunately, attackers focus on what they know how to exploit, and the two are rarely exactly the same. Too many "secure" systems are designed without considering what the attacker is actually likely to do.
9. Interfaces that make security expensive and special. If security features are to be used, they must be convenient and transparent enough that people actually turn them on. It's easy to design encryption mechanisms that come only at the expense of performance or ease of use, and even easier to design mechanisms that invite mistakes. Security should be harder to turn off than on; unfortunately, few systems actually work this way.
10. Little broad-based demand for security. This is a well-known problem among almost everyone who has tied his or her fortune to selling security products and services. Until there is widespread demand for transparent security, the tools and infrastructure needed to support it will be expensive and inaccessible to many applications. This is partly a problem of understanding and exposing the threats and risks in real applications and partly a problem of not designing systems that include security as a basic feature rather than as a later add-on.

A more complete list and discussion of these kinds of threats could easily fill a book of this size and barely scratch the surface. What makes them especially difficult and dangerous is that there are no magic techniques, beyond good engineering and ongoing scrutiny, for avoiding them. The lesson for the aspiring cryptographer is to respect the limits of the art.

Matt Blaze
New York, NY

PART V

SOURCE CODE

1. DES
2. LOKI91
3. IDEA
4. GOST
5. BLOWFISH
6. 3-Way
7. RC5
8. A5
9. SEAL

DES

```
#define ENO   0       /* MODE == encrypt */
#define DE1   1       /* MODE == decrypt */

typedef struct {
        unsigned long ek[32];
        unsigned long dk[32];
} des_ctx;

extern void deskey(unsigned char *, short);
/*                  hexkey[8]     MODE
 * Sets the internal key register according to the hexadecimal
 * key contained in the 8 bytes of hexkey, according to the DES,
 * for encryption or decryption according to MODE.
 */

extern void usekey(unsigned long *);
/*                cookedkey[32]
```

```
 * Loads the internal key register with the data in cookedkey.
 */

extern void cpkey(unsigned long *);
/*                cookedkey[32]
 * Copies the contents of the internal key register into the storage
 * located at &cookedkey[0].
 */

extern void des(unsigned char *, unsigned char *);
/*                 from[8]          to[8]
 * Encrypts/Decrypts (according to the key currently loaded in the
 * internal key register) one block of eight bytes at address 'from'
 * into the block at address 'to'.  They can be the same.
 */

static void scrunch(unsigned char *, unsigned long *);
static void unscrun(unsigned long *, unsigned char *);
static void desfunc(unsigned long *, unsigned long *);
static void cookey(unsigned long *);

static unsigned long KnL[32] = { 0L };
static unsigned long KnR[32] = { 0L };
static unsigned long Kn3[32] = { 0L };
static unsigned char Df_Key[24] = {
       0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,
       0xfe,0xdc,0xba,0x98,0x76,0x54,0x32,0x10,
       0x89,0xab,0xcd,0xef,0x01,0x23,0x45,0x67 };

static unsigned short bytebit[8]    = {
       0200, 0100, 040, 020, 010, 04, 02, 01 };

static unsigned long bigbyte[24] = {
       0x800000L,     0x400000L,     0x200000L,     0x100000L,
       0x80000L,      0x40000L,      0x20000L,      0x10000L,
       0x8000L,       0x4000L,       0x2000L,       0x1000L,
       0x800L,              0x400L,              0x200L,              0x100L,
       0x80L,               0x40L,               0x20L,               0x10L,
       0x8L,          0x4L,          0x2L,          0x1L    };

/* Use the key schedule specified in the Standard (ANSI X3.92-1981). */

static unsigned char pc1[56] = {
       56, 48, 40, 32, 24, 16,  8,   0, 57, 49, 41, 33, 25, 17,
        9,  1, 58, 50, 42, 34, 26,  18, 10,  2, 59, 51, 43, 35,
       62, 54, 46, 38, 30, 22, 14,   6, 61, 53, 45, 37, 29, 21,
       13,  5, 60, 52, 44, 36, 28,  20, 12,  4, 27, 19, 11,  3 };

static unsigned char totrot[16] = {
       1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28 };

static unsigned char pc2[48] = {
       13, 16, 10, 23,  0,  4,       2, 27, 14,  5, 20,  9,
       22, 18, 11,  3, 25,  7,      15,  6, 26, 19, 12,  1,
       40, 51, 30, 36, 46, 54,      29, 39, 50, 44, 32, 47,
       43, 48, 38, 55, 33, 52,      45, 41, 49, 35, 28, 31 };
```

```
void deskey(key, edf)       /* Thanks to James Gillogly & Phil Karn! */
unsigned char *key;
short edf;
{
        register int i, j, l, m, n;
        unsigned char pc1m[56], pcr[56];
        unsigned long kn[32];

        for ( j = 0; j < 56; j++ ) {
                l = pc1[j];
                m = l & 07;
                pc1m[j] = (key[l >> 3] & bytebit[m]) ? 1 : 0;
                }
        for( i = 0; i < 16; i++ ) {
                if( edf == DE1 ) m = (15 - i) << 1;
                else m = i << 1;
                n = m + 1;
                kn[m] = kn[n] = 0L;
                for( j = 0; j < 28; j++ ) {
                        l = j + totrot[i];
                        if( l < 28 ) pcr[j] = pc1m[l];
                        else pcr[j] = pc1m[l - 28];
                        }
                for( j = 28; j < 56; j++ ) {
                    l = j + totrot[i];
                    if( l < 56 ) pcr[j] = pc1m[l];
                    else pcr[j] = pc1m[l - 28];
                    }
                for( j = 0; j < 24; j++ ) {
                        if( pcr[pc2[j]] ) kn[m] |= bigbyte[j];
                        if( pcr[pc2[j+24]] ) kn[n] |= bigbyte[j];
                        }
                }
        cookey(kn);
        return;
}

static void cookey(raw1)
register unsigned long *raw1;
{
        register unsigned long *cook, *raw0;
        unsigned long dough[32];
        register int i;

        cook = dough;
        for( i = 0; i < 16; i++, raw1++ ) {
                raw0 = raw1++;
                *cook    = (*raw0 & 0x00fc0000L) << 6;
                *cook   |= (*raw0 & 0x00000fc0L) << 10;
                *cook   |= (*raw1 & 0x00fc0000L) >> 10;
                *cook++         |= (*raw1 & 0x00000fc0L) >> 6;
                *cook    = (*raw0 & 0x0003f000L) << 12;
                *cook   |= (*raw0 & 0x0000003fL) << 16;
                *cook   |= (*raw1 & 0x0003f000L) >> 4;
                *cook++         |= (*raw1 & 0x0000003fL);
```

```
            }
        usekey(dough);
        return;
}

void cpkey(into)
register unsigned long *into;
{
        register unsigned long *from, *endp;

        from = KnL, endp = &KnL[32];
        while( from < endp ) *into++ = *from++;
        return;
}

void usekey(from)
register unsigned long *from;
{
        register unsigned long *to, *endp;

        to = KnL, endp = &KnL[32];
        while( to < endp ) *to++ = *from++;
        return;
}

void des(inblock, outblock)
unsigned char *inblock, *outblock;
{
        unsigned long work[2];

        scrunch(inblock, work);
        desfunc(work, KnL);
        unscrun(work, outblock);
        return;
}

static void scrunch(outof, into)
register unsigned char *outof;
register unsigned long *into;
{
        *into   = (*outof++ & 0xffL) << 24;
        *into  |= (*outof++ & 0xffL) << 16;
        *into  |= (*outof++ & 0xffL) << 8;
        *into++ |= (*outof++ & 0xffL);
        *into   = (*outof++ & 0xffL) << 24;
        *into  |= (*outof++ & 0xffL) << 16;
        *into  |= (*outof++ & 0xffL) << 8;
        *into  |= (*outof   & 0xffL);
        return;
}

static void unscrun(outof, into)
register unsigned long *outof;
register unsigned char *into;
{
```

```
        *into++ = (*outof >> 24) & 0xffL;
        *into++ = (*outof >> 16) & 0xffL;
        *into++ = (*outof >>  8) & 0xffL;
        *into++ =  *outof++          & 0xffL;
        *into++ = (*outof >> 24) & 0xffL;
        *into++ = (*outof >> 16) & 0xffL;
        *into++ = (*outof >>  8) & 0xffL;
        *into   =  *outof     & 0xffL;
        return;
}

static unsigned long SP1[64] = {
        0x01010400L, 0x00000000L, 0x00010000L, 0x01010404L,
        0x01010004L, 0x00010404L, 0x00000004L, 0x00010000L,
        0x00000400L, 0x01010400L, 0x01010404L, 0x00000400L,
        0x01000404L, 0x01010004L, 0x01000000L, 0x00000004L,
        0x00000404L, 0x01000400L, 0x01000400L, 0x00010400L,
        0x00010400L, 0x01010000L, 0x01010000L, 0x01000404L,
        0x00010004L, 0x01000004L, 0x01000004L, 0x00010004L,
        0x00000000L, 0x00000404L, 0x00010404L, 0x01000000L,
        0x00010000L, 0x01010404L, 0x00000004L, 0x01010000L,
        0x01010400L, 0x01000000L, 0x01000000L, 0x00000400L,
        0x01010004L, 0x00010000L, 0x00010400L, 0x01000004L,
        0x00000400L, 0x00000004L, 0x01000404L, 0x00010404L,
        0x01010404L, 0x00010004L, 0x01010000L, 0x01000404L,
        0x01000004L, 0x00000404L, 0x00010404L, 0x01010400L,
        0x00000404L, 0x01000400L, 0x01000400L, 0x00000000L,
        0x00010004L, 0x00010400L, 0x00000000L, 0x01010004L };

static unsigned long SP2[64] = {
        0x80108020L, 0x80008000L, 0x00008000L, 0x00108020L,
        0x00100000L, 0x00000020L, 0x80100020L, 0x80008020L,
        0x80000020L, 0x80108020L, 0x80108000L, 0x80000000L,
        0x80008000L, 0x00100000L, 0x00000020L, 0x80100020L,
        0x00108000L, 0x00100020L, 0x80008020L, 0x00000000L,
        0x80000000L, 0x00008000L, 0x00108020L, 0x80100000L,
        0x00100020L, 0x80000020L, 0x00000000L, 0x00108000L,
        0x00008020L, 0x80108000L, 0x80100000L, 0x00008020L,
        0x00000000L, 0x00108020L, 0x80100020L, 0x00100000L,
        0x80008020L, 0x80100000L, 0x80108000L, 0x00008000L,
        0x80100000L, 0x80008000L, 0x00000020L, 0x80108020L,
        0x00108020L, 0x00000020L, 0x00008000L, 0x80000000L,
        0x00008020L, 0x80108000L, 0x00100000L, 0x80000020L,
        0x00100020L, 0x80008020L, 0x80000020L, 0x00100020L,
        0x00108000L, 0x00000000L, 0x80008000L, 0x00008020L,
        0x80000000L, 0x80100020L, 0x80108020L, 0x00108000L };

static unsigned long SP3[64] = {
        0x00000208L, 0x08020200L, 0x00000000L, 0x08020008L,
        0x08000200L, 0x00000000L, 0x00020208L, 0x08000200L,
        0x00020008L, 0x08000008L, 0x08000008L, 0x00020000L,
        0x08020208L, 0x00020008L, 0x08020000L, 0x00000208L,
        0x08000000L, 0x00000008L, 0x08020200L, 0x00000200L,
        0x00020200L, 0x08020000L, 0x08020008L, 0x00020208L,
```

```
        0x08000208L, 0x00020200L, 0x00020000L, 0x08000208L,
        0x00000008L, 0x08020208L, 0x00000200L, 0x08000000L,
        0x08020200L, 0x08000000L, 0x00020008L, 0x00000208L,
        0x00020000L, 0x08020200L, 0x08000200L, 0x00000000L,
        0x00000200L, 0x00020008L, 0x08020208L, 0x08000200L,
        0x08000008L, 0x00000200L, 0x00000000L, 0x08020008L,
        0x08000208L, 0x00020000L, 0x08000000L, 0x08020208L,
        0x00000008L, 0x00020208L, 0x00020200L, 0x08000008L,
        0x08020000L, 0x08000208L, 0x00000208L, 0x08020000L,
        0x00020208L, 0x00000008L, 0x08020008L, 0x00020200L };

static unsigned long SP4[64] = {
        0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
        0x00802080L, 0x00800081L, 0x00800001L, 0x00002001L,
        0x00000000L, 0x00802000L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00800080L, 0x00800001L,
        0x00000001L, 0x00002000L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002001L, 0x00002080L,
        0x00800081L, 0x00000001L, 0x00002080L, 0x00800080L,
        0x00002000L, 0x00802080L, 0x00802081L, 0x00000081L,
        0x00800080L, 0x00800001L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00000000L, 0x00802000L,
        0x00002080L, 0x00800080L, 0x00800081L, 0x00000001L,
        0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
        0x00802081L, 0x00000081L, 0x00000001L, 0x00002000L,
        0x00800001L, 0x00002001L, 0x00802080L, 0x00800081L,
        0x00002001L, 0x00002080L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002000L, 0x00802080L };

static unsigned long SP5[64] = {
        0x00000100L, 0x02080100L, 0x02080000L, 0x42000100L,
        0x00080000L, 0x00000100L, 0x40000000L, 0x02080000L,
        0x40080100L, 0x00080000L, 0x02000100L, 0x40080100L,
        0x42000100L, 0x42080000L, 0x00080100L, 0x40000000L,
        0x02000000L, 0x40080000L, 0x40080000L, 0x00000000L,
        0x40000100L, 0x42080100L, 0x42080100L, 0x02000100L,
        0x42080000L, 0x40000100L, 0x00000000L, 0x42000000L,
        0x02080100L, 0x02000000L, 0x42000000L, 0x00080100L,
        0x00080000L, 0x42000100L, 0x00000100L, 0x02000000L,
        0x40000000L, 0x02080000L, 0x42000100L, 0x40080100L,
        0x02000100L, 0x40000000L, 0x42080000L, 0x02080100L,
        0x40080100L, 0x00000100L, 0x02000000L, 0x42080000L,
        0x42080100L, 0x00080100L, 0x42000000L, 0x42080100L,
        0x02080000L, 0x00000000L, 0x40080000L, 0x42000000L,
        0x00080100L, 0x02000100L, 0x40000100L, 0x00080000L,
        0x00000000L, 0x40080000L, 0x02080100L, 0x40000100L };

static unsigned long SP6[64] = {
        0x20000010L, 0x20400000L, 0x00004000L, 0x20404010L,
        0x20400000L, 0x00000010L, 0x20404010L, 0x00400000L,
        0x20004000L, 0x00404010L, 0x00400000L, 0x20000010L,
        0x00400010L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x00000000L, 0x00400010L, 0x20004010L, 0x00004000L,
        0x00404000L, 0x20004010L, 0x00000010L, 0x20400010L,
```

```
        0x20400010L, 0x00000000L, 0x00404010L, 0x20404000L,
        0x00004010L, 0x00404000L, 0x20404000L, 0x20000000L,
        0x20004000L, 0x00000010L, 0x20400010L, 0x00404000L,
        0x20404010L, 0x00400000L, 0x00004010L, 0x20000010L,
        0x00400000L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x20000010L, 0x20404010L, 0x00404000L, 0x20400000L,
        0x00404010L, 0x20404000L, 0x00000000L, 0x20400010L,
        0x00000010L, 0x00004000L, 0x20400000L, 0x00404010L,
        0x00004000L, 0x00400010L, 0x20004010L, 0x00000000L,
        0x20404000L, 0x20000000L, 0x00400010L, 0x20004010L };

static unsigned long SP7[64] = {
        0x00200000L, 0x04200002L, 0x04000802L, 0x00000000L,
        0x00000800L, 0x04000802L, 0x00200802L, 0x04200800L,
        0x04200802L, 0x00200000L, 0x00000000L, 0x04000002L,
        0x00000002L, 0x04000000L, 0x04200002L, 0x00000802L,
        0x04000800L, 0x00200802L, 0x00200002L, 0x04000800L,
        0x04000002L, 0x04200000L, 0x04200800L, 0x00200002L,
        0x04200000L, 0x00000800L, 0x00000802L, 0x04200802L,
        0x00200800L, 0x00000002L, 0x04000000L, 0x00200800L,
        0x04000000L, 0x00200800L, 0x00200000L, 0x04000802L,
        0x04000802L, 0x04200002L, 0x04200002L, 0x00000002L,
        0x00200002L, 0x04000000L, 0x04000800L, 0x00200000L,
        0x04200800L, 0x00000802L, 0x00200802L, 0x04200800L,
        0x00000802L, 0x04000002L, 0x04200802L, 0x04200000L,
        0x00200800L, 0x00000000L, 0x00000002L, 0x04200802L,
        0x00000000L, 0x00200802L, 0x04200000L, 0x00000800L,
        0x04000002L, 0x04000800L, 0x00000800L, 0x00200002L };

static unsigned long SP8[64] = {
        0x10001040L, 0x00001000L, 0x00040000L, 0x10041040L,
        0x10000000L, 0x10001040L, 0x00000040L, 0x10000000L,
        0x00040040L, 0x10040000L, 0x10041040L, 0x00041000L,
        0x10041000L, 0x00041040L, 0x00001000L, 0x00000040L,
        0x10040000L, 0x10000040L, 0x10001000L, 0x00001040L,
        0x00041000L, 0x00040040L, 0x10040040L, 0x10041000L,
        0x00001040L, 0x00000000L, 0x00000000L, 0x10040040L,
        0x10000040L, 0x10001000L, 0x00041040L, 0x00040000L,
        0x00041040L, 0x00040000L, 0x10041000L, 0x00001000L,
        0x00000040L, 0x10040040L, 0x00001000L, 0x00041040L,
        0x10001000L, 0x00000040L, 0x10000040L, 0x10040000L,
        0x10040040L, 0x10000000L, 0x00040000L, 0x10001040L,
        0x00000000L, 0x10041040L, 0x00040040L, 0x10000040L,
        0x10040000L, 0x10001000L, 0x10001040L, 0x00000000L,
        0x10041040L, 0x00041000L, 0x00041000L, 0x00001040L,
        0x00001040L, 0x00040040L, 0x10000000L, 0x10041000L };

static void desfunc(block, keys)
register unsigned long *block, *keys;
{
        register unsigned long fval, work, right, leftt;
        register int round;

        leftt = block[0];
```

```
right = block[1];
work = ((leftt >> 4) ^ right) & 0x0f0f0f0fL;
right ^= work;
leftt ^= (work << 4);
work = ((leftt >> 16) ^ right) & 0x0000ffffL;
right ^= work;
leftt ^= (work << 16);
work = ((right >> 2) ^ leftt) & 0x33333333L;
leftt ^= work;
right ^= (work << 2);
work = ((right >> 8) ^ leftt) & 0x00ff00ffL;
leftt ^= work;
right ^= (work << 8);
right = ((right << 1) | ((right >> 31) & 1L)) & 0xffffffffL;
work = (leftt ^ right) & 0xaaaaaaaaL;
leftt ^= work;
right ^= work;
leftt = ((leftt << 1) | ((leftt >> 31) & 1L)) & 0xffffffffL;

for( round = 0; round < 8; round++ ) {
      work  = (right << 28) | (right >> 4);
      work ^= *keys++;
      fval  = SP7[ work             & 0x3fL];
      fval |= SP5[(work >>  8) & 0x3fL];
      fval |= SP3[(work >> 16) & 0x3fL];
      fval |= SP1[(work >> 24) & 0x3fL];
      work  = right ^ *keys++;
      fval |= SP8[ work             & 0x3fL];
      fval |= SP6[(work >>  8) & 0x3fL];
      fval |= SP4[(work >> 16) & 0x3fL];
      fval |= SP2[(work >> 24) & 0x3fL];
      leftt ^= fval;
      work  = (leftt << 28) | (leftt >> 4);
      work ^= *keys++;
      fval  = SP7[ work             & 0x3fL];
      fval |= SP5[(work >>  8) & 0x3fL];
      fval |= SP3[(work >> 16) & 0x3fL];
      fval |= SP1[(work >> 24) & 0x3fL];
      work  = leftt ^ *keys++;
      fval |= SP8[ work             & 0x3fL];
      fval |= SP6[(work >>  8) & 0x3fL];
      fval |= SP4[(work >> 16) & 0x3fL];
      fval |= SP2[(work >> 24) & 0x3fL];
      right ^= fval;
      }

right = (right << 31) | (right >> 1);
work = (leftt ^ right) & 0xaaaaaaaaL;
leftt ^= work;
right ^= work;
leftt = (leftt << 31) | (leftt >> 1);
work = ((leftt >> 8) ^ right) & 0x00ff00ffL;
right ^= work;
leftt ^= (work << 8);
```

```
        work = ((leftt >> 2) ^ right) & 0x33333333L;
        right ^= work;
        leftt ^= (work << 2);
        work = ((right >> 16) ^ leftt) & 0x0000ffffL;
        leftt ^= work;
        right ^= (work << 16);
        work = ((right >> 4) ^ leftt) & 0x0f0f0f0fL;
        leftt ^= work;
        right ^= (work << 4);
        *block++ = right;
        *block = leftt;
        return;
}

/* Validation sets:
 *
 * Single-length key, single-length plaintext -
 * Key     : 0123 4567 89ab cdef
 * Plain   : 0123 4567 89ab cde7
 * Cipher : c957 4425 6a5e d31d
 *
 ********************************************************************/

void des_key(des_ctx *dc, unsigned char *key){
        deskey(key,ENO);
        cpkey(dc->ek);
        deskey(key,DE1);
        cpkey(dc->dk);
}

/* Encrypt several blocks in ECB mode.  Caller is responsible for
   short blocks. */
void des_enc(des_ctx *dc, unsigned char *data, int blocks){
        unsigned long work[2];
        int i;
        unsigned char *cp;

        cp = data;
        for(i=0;i<blocks;i++){
                scrunch(cp,work);
                desfunc(work,dc->ek);
                unscrun(work,cp);
                cp+=8;
        }
}

void des_dec(des_ctx *dc, unsigned char *data, int blocks){
        unsigned long work[2];
        int i;
        unsigned char *cp;

        cp = data;
        for(i=0;i<blocks;i++){
                scrunch(cp,work);
                desfunc(work,dc->dk);
```

```
                unscrun(work,cp);
                cp+=8;
        }
}

void main(void){
        des_ctx dc;
        int i;
        unsigned long data[10];
        char *cp,key[8] = {0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef};
        char x[8] = {0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xe7};

        cp = x;

        des_key(&dc,key);
        des_enc(&dc,cp,1);
        printf("Enc(0..7,0..7) = ");
        for(i=0;i<8;i++) printf("%02x ", ((unsigned int) cp[i])&0x00ff);
        printf("\n");

        des_dec(&dc,cp,1);

        printf("Dec(above,0..7) = ");
        for(i=0;i<8;i++) printf("%02x ",((unsigned int)cp[i])&0x00ff);
        printf("\n");

        cp = (char *) data;
        for(i=0;i<10;i++)data[i]=i;

        des_enc(&dc,cp,5); /* Enc 5 blocks. */
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                i/2,data[i],data[i+1]);

        des_dec(&dc,cp,1);
        des_dec(&dc,cp+8,4);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                i/2,data[i],data[i+1]);

}
```

LOKI91

```
#include <stdio.h>

#define LOKIBLK         8               /* No of bytes in a LOKI data-block          */
#define ROUNDS          16              /* No of LOKI rounds                         */

typedef unsigned long           Long;   /* type specification for aligned LOKI blocks
*/

extern Long     lokikey[2];     /* 64-bit key used by LOKI routines          */
extern char     *loki_lib_ver;          /* String with version no. & copyright       */

#ifdef __STDC__                         /* declare prototypes for library functions  */
extern void enloki(char *b);
```

```
extern void deloki(char *b);
extern void setlokikey(char key[LOKIBLK]);
#else                           /* else just declare library functions extern */
extern void enloki(), deloki(), setlokikey();
#endif __STDC__

char P[32] = {
        31, 23, 15, 7, 30, 22, 14, 6,
        29, 21, 13, 5, 28, 20, 12, 4,
        27, 19, 11, 3, 26, 18, 10, 2,
        25, 17, 9, 1, 24, 16, 8, 0
        };

typedef         struct {
        short  gen;             /* irreducible polynomial used in this field */
        short  exp;             /* exponent used to generate this s function */
        } sfn_desc;

sfn_desc sfn[] = {
        { /* 101110111 */ 375, 31}, { /* 101111011 */ 379, 31},
        { /* 110000111 */ 391, 31}, { /* 110001011 */ 395, 31},
        { /* 110001101 */ 397, 31}, { /* 110011111 */ 415, 31},
        { /* 110100011 */ 419, 31}, { /* 110101001 */ 425, 31},
        { /* 110110001 */ 433, 31}, { /* 110111101 */ 445, 31},
        { /* 111000011 */ 451, 31}, { /* 111001111 */ 463, 31},
        { /* 111010111 */ 471, 31}, { /* 111011101 */ 477, 31},
        { /* 111100111 */ 487, 31}, { /* 111110011 */ 499, 31},
        { 00, 00}       };

typedef struct {
        Long loki_subkeys[ROUNDS];
} loki_ctx;

static Long     f();                    /* declare LOKI function f */
static short    s();                    /* declare LOKI S-box fn s */

#define ROL12(b) b = ((b << 12) | (b >> 20));
#define ROL13(b) b = ((b << 13) | (b >> 19));

#ifdef  LITTLE_ENDIAN
#define bswap(cb) {                                 \
        register char   c;                          \
        c = cb[0]; cb[0] = cb[3]; cb[3] = c;        \
        c = cb[1]; cb[1] = cb[2]; cb[2] = c;        \
        c = cb[4]; cb[4] = cb[7]; cb[7] = c;        \
        c = cb[5]; cb[5] = cb[6]; cb[6] = c;        \
}
#endif

void
setlokikey(loki_ctx *c, char *key)
{
        register        i;
        register Long   KL, KR;
```

```
#ifdef LITTLE_ENDIAN
        bswap(key);                     /* swap bytes round if little-endian */
#endif
        KL = ((Long *)key)[0];
        KR = ((Long *)key)[1];

        for (i=0; i<ROUNDS; i+=4) {             /* Generate the 16 subkeys */
            c->loki_subkeys[i] = KL;
            ROL12 (KL);
            c->loki_subkeys[i+1] = KL;
            ROL13 (KL);
            c->loki_subkeys[i+2] = KR;
            ROL12 (KR);
            c->loki_subkeys[i+3] = KR;
            ROL13 (KR);
        }

#ifdef LITTLE_ENDIAN
        bswap(key);                     /* swap bytes back if little-endian */
#endif
}

void
enloki (loki_ctx *c, char *b)
{
        register        i;
        register Long   L, R;           /* left & right data halves  */

#ifdef LITTLE_ENDIAN
        bswap(b);                       /* swap bytes round if little-endian */
#endif

        L = ((Long *)b)[0];
        R = ((Long *)b)[1];

        for (i=0; i<ROUNDS; i+=2) {             /* Encrypt with the 16 subkeys */
            L ^= f (R, c->loki_subkeys[i]);
            R ^= f (L, c->loki_subkeys[i+1]);
        }

        ((Long *)b)[0] = R;             /* Y = swap(LR) */
        ((Long *)b)[1] = L;

#ifdef LITTLE_ENDIAN
        bswap(b);                       /* swap bytes round if little-endian */
#endif
}

void
deloki(loki_ctx *c, char *b)
{
        register        i;
        register Long   L, R;                   /* left & right data halves  */

#ifdef LITTLE_ENDIAN
```

```
        bswap(b);                       /* swap bytes round if little-endian */
#endif

        L = ((Long *)b)[0];                     /* LR = X XOR K */
        R = ((Long *)b)[1];

        for (i=ROUNDS; i>0; i-=2) {                     /* subkeys in reverse order */
            L ^= f(R, c->loki_subkeys[i-1]);
            R ^= f(L, c->loki_subkeys[i-2]);
        }

        ((Long *)b)[0] = R;                     /* Y = LR XOR K */
        ((Long *)b)[1] = L;
}

#define MASK12          0x0fff                  /* 12 bit mask for expansion E */

static Long
f(r, k)
register Long  r;       /* Data value R(i-1) */
Long           k;       /* Key     K(i)   */
{
        Long   a, b, c;                 /* 32 bit S-box output, & P output */

        a = r ^ k;                      /* A = R(i-1) XOR K(i) */

        /* want to use slow speed/small size version */
        b = ((Long)s((a          & MASK12))       ) | /* B = S(E(R(i-1))^K(i)) */
            ((Long)s(((a >>  8) & MASK12)) <<  8) |
            ((Long)s(((a >> 16) & MASK12)) << 16) |
            ((Long)s((((a >> 24) | (a << 8)) & MASK12)) << 24);

        perm32(&c, &b, P);              /* C = P(S( E(R(i-1)) XOR K(i))) */

        return(c);                      /* f returns the result C */
}

static short s(i)
register Long i;        /* return S-box value for input i */
{
        register short r, c, v, t;
        short  exp8();                  /* exponentiation routine for GF(2^8) */

        r = ((i>>8) & 0xc) | (i & 0x3);                 /* row value-top 2 & bottom 2 */
        c = (i>>2) & 0xff;                              /* column value-middle 8 bits */
        t = (c + ((r * 17) ^ 0xff)) & 0xff;             /* base value for Sfn */
        v = exp8(t, sfn[r].exp, sfn[r].gen);            /* Sfn[r] = t ^ exp mod gen */
        return(v);
}

#define         MSB     0x80000000L             /* MSB of 32-bit word */

perm32(out, in , perm)
Long    *out;           /* Output 32-bit block to be permuted                   */
```

```
Long    *in;            /* Input  32-bit block after permutation            */
char    perm[32];       /* Permutation array                                */
{
        Long   mask = MSB;                      /* mask used to set bit in output    */
        register int   i, o, b;         /* input bit no, output bit no, value */
        register char  *p = perm;       /* ptr to permutation array  */

        *out = 0;                       /* clear output block */
        for (o=0; o<32; o++) {                  /* For each output bit position o */
              i =(int)*p++;                     /* get input bit permuted to output o */
              b = (*in >> i) & 01;              /* value of input bit i */
              if (b)                    /* If the input bit i is set */
                      *out |= mask;                     /*  OR in mask to output i */
              mask >>= 1;                       /* Shift mask to next bit    */
        }
}

#define SIZE 256                /* 256 elements in GF(2^8) */

short mult8(a, b, gen)
short   a, b;           /* operands for multiply */
short   gen;            /* irreducible polynomial generating Galois Field */
{
        short  product = 0;             /* result of multiplication */

        while(b != 0) {                         /* while multiplier is non-zero */
              if (b & 01)
                      product ^= a;             /*   add multiplicand if LSB of b set */
              a <<= 1;                  /*   shift multiplicand one place */
              if (a >= SIZE)
                      a ^= gen;         /*   and modulo reduce if needed */
              b >>= 1;                  /*   shift multiplier one place  */
        }
        return(product);
}

short exp8(base, exponent, gen)
short   base;           /* base of exponentiation       */
short   exponent;       /* exponent                     */
short   gen;            /* irreducible polynomial generating Galois Field */
{
        short  accum = base;            /* superincreasing sequence of base */
        short  result = 1;      /* result of exponentiation       */

        if (base == 0)                  /* if zero base specified then       */
              return(0);        /* the result is "0" if base = 0    */

        while (exponent != 0) {         /* repeat while exponent non-zero */
              if (( exponent & 0x0001) == 0x0001)           /* multiply if exp 1 */
                      result = mult8(result, accum, gen);
              exponent >>= 1;                   /* shift exponent to next digit */
              accum = mult8(accum, accum, gen);             /* & square  */
        }
        return(result);
}
```

```
void loki_key(loki_ctx *c, unsigned char *key){
        setlokikey(c,key);
}

void loki_enc(loki_ctx *c, unsigned char *data, int blocks){
        unsigned char *cp;
        int i;

        cp = data;
        for(i=0;i<blocks;i++){
                enloki(c,cp);
                cp+=8;
        }
}

void loki_dec(loki_ctx *c, unsigned char *data, int blocks){
        unsigned char *cp;
        int i;

        cp = data;
        for(i=0;i<blocks;i++){
                deloki(c,cp);
                cp+=8;
        }
}

void main(void){
        loki_ctx lc;
        unsigned long data[10];
        unsigned char *cp;
        unsigned char key[] = {0,1,2,3,4,5,6,7};
        int i;

        for(i=0;i<10;i++) data[i]=i;

        loki_key(&lc,key);

        cp = (char *)data;
        loki_enc(&lc,cp,5);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx\n",
                        i/2,data[i],data[i+1]);
        loki_dec(&lc,cp,1);
        loki_dec(&lc,cp+8,4);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx\n",
                        i/2,data[i],data[i+1]);
}
```

IDEA

```
typedef unsigned char boolean;      /* values are TRUE or FALSE */
typedef unsigned char byte; /* values are 0-255 */
typedef byte *byteptr;      /* pointer to byte */
```

```
typedef char *string;/* pointer to ASCII character string */
typedef unsigned short word16;      /* values are 0-65535 */
typedef unsigned long word32;       /* values are 0-4294967295 */

#ifndef TRUE
#define FALSE 0
#define TRUE (!FALSE)
#endif /* if TRUE not already defined */

#ifndef min   /* if min macro not already defined */
#define min(a,b) ( (a)<(b) ? (a) : (b) )
#define max(a,b) ( (a)>(b) ? (a) : (b) )
#endif /* if min macro not already defined */

#define IDEAKEYSIZE 16
#define IDEABLOCKSIZE 8

#define IDEAROUNDS 8
#define IDEAKEYLEN (6*IDEAROUNDS+4)

typedef struct{
      word16 ek[IDEAKEYLEN],dk[IDEAKEYLEN];
}idea_ctx;

/* End includes for IDEA.C */
#ifdef IDEA32        /* Use >16-bit temporaries */
#define low16(x) ((x) & 0xFFFF)
typedef unsigned int uint16;/* at LEAST 16 bits, maybe more */
#else
#define low16(x) (x) /* this is only ever applied to uint16's */
typedef word16 uint16;
#endif

#ifdef SMALL_CACHE
static uint16
mul(register uint16 a, register uint16 b)
{
      register word32 p;

      p = (word32)a * b;
      if (p) {
            b = low16(p);
            a = p>>16;
            return (b - a) + (b < a);
      } else if (a) {
            return 1-b;
      } else {
            return 1-a;
      }
} /* mul */
#endif /* SMALL_CACHE */

static uint16
mulInv(uint16 x)
{
```

```
        uint16 t0, t1;
        uint16 q, y;

        if (x <= 1)
                return x;       /* 0 and 1 are self-inverse */
        t1 = 0x10001L / x;      /* Since x >= 2, this fits into 16 bits */
        y = 0x10001L % x;
        if (y == 1)
                return low16(1-t1);
        t0 = 1;
        do {
                q = x / y;
                x = x % y;
                t0 += q * t1;
                if (x == 1)
                        return t0;
                q = y / x;
                y = y % x;
                t1 += q * t0;
        } while (y != 1);
        return low16(1-t1);
} /* mulInv */

static void
ideaExpandKey(byte const *userkey, word16 *EK)
{
        int i,j;

        for (j=0; j<8; j++) {
                EK[j] = (userkey[0]<<8) + userkey[1];
                userkey += 2;
        }
        for (i=0; j < IDEAKEYLEN; j++) {
                i++;
                EK[i+7] = EK[i & 7] << 9 | EK[i+1 & 7] >> 7;
                EK += i & 8;
                i &= 7;
        }
} /* ideaExpandKey */

static void
ideaInvertKey(word16 const *EK, word16 DK[IDEAKEYLEN])
{
        int i;
        uint16 t1, t2, t3;
        word16 temp[IDEAKEYLEN];
        word16 *p = temp + IDEAKEYLEN;

        t1 = mulInv(*EK++);
        t2 = -*EK++;
        t3 = -*EK++;
        *--p = mulInv(*EK++);
        *--p = t3;
        *--p = t2;
```

```
        *--p = t1;

        for (i = 0; i < IDEAROUNDS-1; i++) {
                t1 = *EK++;
                *--p = *EK++;
                *--p = t1;

                t1 = mulInv(*EK++);
                t2 = -*EK++;
                t3 = -*EK++;
                *--p = mulInv(*EK++);
                *--p = t2;
                *--p = t3;
                *--p = t1;
        }
        t1 = *EK++;
        *--p = *EK++;
        *--p = t1;

        t1 = mulInv(*EK++);
        t2 = -*EK++;
        t3 = -*EK++;
        *--p = mulInv(*EK++);
        *--p = t3;
        *--p = t2;
        *--p = t1;
/* Copy and destroy temp copy */
        memcpy(DK, temp, sizeof(temp));
        for(i=0;i<IDEAKEYLEN;i++)temp[i]=0;
} /* ideaInvertKey */

#ifdef SMALL_CACHE
#define MUL(x,y) (x = mul(low16(x),y))
#else /* !SMALL_CACHE */
#ifdef AVOID_JUMPS
#define MUL(x,y) (x = low16(x-1), t16 = low16((y)-1), \
                t32 = (word32)x*t16 + x + t16 + 1, x = low16(t32), \
                t16 = t32>>16, x = (x-t16) + (x<t16) )
#else /* !AVOID_JUMPS (default) */
#define MUL(x,y) \
        ((t16 = (y)) ? \
                (x=low16(x)) ? \
                        t32 = (word32)x*t16, \
                        x = low16(t32), \
                        t16 = t32>>16, \
                        x = (x-t16)+(x<t16) \
                : \
                        (x = 1-t16) \
        : \
                (x = 1-x))
#endif
#endif

static void
```

```
ideaCipher(byte *inbuf, byte *outbuf, word16 *key)
{
        register uint16 x1, x2, x3, x4, s2, s3;
        word16 *in, *out;
#ifndef SMALL_CACHE
        register uint16 t16; /* Temporaries needed by MUL macro */
        register word32 t32;
#endif
        int r = IDEAROUNDS;

        in = (word16 *)inbuf;
        x1 = *in++;  x2 = *in++;
        x3 = *in++;  x4 = *in;
#ifndef HIGHFIRST
        x1 = (x1 >>8) | (x1<<8);
        x2 = (x2 >>8) | (x2<<8);
        x3 = (x3 >>8) | (x3<<8);
        x4 = (x4 >>8) | (x4<<8);
#endif
        do {
                MUL(x1,*key++);
                x2 += *key++;
                x3 += *key++;
                MUL(x4, *key++);

                s3 = x3;
                x3 ^= x1;
                MUL(x3, *key++);
                s2 = x2;
                x2 ^= x4;
                x2 += x3;
                MUL(x2, *key++);
                x3 += x2;

                x1 ^= x2;  x4 ^= x3;

                x2 ^= s3;  x3 ^= s2;
        } while (--r);
        MUL(x1, *key++);
        x3 += *key++;
        x2 += *key++;
        MUL(x4, *key);

        out = (word16 *)outbuf;
#ifdef HIGHFIRST
        *out++ = x1;
        *out++ = x3;
        *out++ = x2;
        *out = x4;
#else /* !HIGHFIRST */
        *out++ = (x1 >>8) | (x1<<8);
        *out++ = (x3 >>8) | (x3<<8);
        *out++ = (x2 >>8) | (x2<<8);
        *out = (x4 >>8) | (x4<<8);
```

```
#endif
} /* ideaCipher */

void idea_key(idea_ctx *c, unsigned char *key){
        ideaExpandKey(key,c->ek);
        ideaInvertKey(c->ek,c->dk);
}

void idea_enc(idea_ctx *c, unsigned char *data, int blocks){
        int i;
        unsigned char *d = data;

        for(i=0;i<blocks;i++){
                ideaCipher(d,d,c->ek);
                d+=8;
        }
}

void idea_dec(idea_ctx *c, unsigned char *data, int blocks){
        int i;
        unsigned char *d = data;

        for(i=0;i<blocks;i++){
                ideaCipher(d,d,c->dk);
                d+=8;
        }
}

#include <stdio.h>

#ifndef BLOCKS
#ifndef KBYTES
#define KBYTES 1024
#endif
#define BLOCKS (64*KBYTES)
#endif

int
main(void)
{       /* Test driver for IDEA cipher */
        int i, j, k;
        idea_ctx c;
        byte userkey[16];
        word16 EK[IDEAKEYLEN], DK[IDEAKEYLEN];
        byte XX[8], YY[8], ZZ[8];
        word32 long_block[10]; /* 5 blocks */
        long l;
        char *lbp;

        /* Make a sample user key for testing... */
        for(i=0; i<16; i++)
                userkey[i] = i+1;

        idea_key(&c,userkey);

        /* Make a sample plaintext pattern for testing... */
```

```
        for (k=0; k<8; k++)
                XX[k] = k;

        idea_enc(&c,XX,1); /* encrypt */

        lbp = (unsigned char *) long_block;
        for(i=0;i<10;i++) long_block[i] = i;
        idea_enc(&c,lbp,5);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                i/2,long_block[i],long_block[i+1]);

        idea_dec(&c,lbp,3);
        idea_dec(&c,lbp+24,2);

        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                i/2,long_block[i],long_block[i+1]);

        return 0;       /* normal exit */
} /* main */
```

GOST

```
typedef unsigned long u4;
typedef unsigned char byte;

typedef struct {
        u4 k[8];
        /* Constant s-boxes -- set up in gost_init(). */
        char k87[256],k65[256],k43[256],k21[256];
} gost_ctx;

/* Note:  encrypt and decrypt expect full blocks--padding blocks is
          caller's responsibility.  All bulk encryption is done in
          ECB mode by these calls.  Other modes may be added easily
          enough.                                                    */
void gost_enc(gost_ctx *, u4 *, int);
void gost_dec(gost_ctx *, u4 *, int);
void gost_key(gost_ctx *, u4 *);
void gost_init(gost_ctx *);
void gost_destroy(gost_ctx *);

#ifdef __alpha  /* Any other 64-bit machines? */
typedef unsigned int word32;
#else
typedef unsigned long word32;
#endif

kboxinit(gost_ctx *c)
{
        int i;

        byte k8[16] = {14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6,
                       12,  5,  9,  0,  7 };
        byte k7[16] = {15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2,
```

```
                        13, 12,  0,  5, 10 };
        byte k6[16] = {10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,
                         7, 11,  4,  2,  8 };
        byte k5[16] = { 7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,
                         5, 11, 12,  4, 15 };
        byte k4[16] = { 2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3,
                        15, 13,  0, 14,  9 };
        byte k3[16] = {12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,
                         4, 14,  7,  5, 11 };
        byte k2[16] = { 4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,
                         7,  5, 10,  6,  1 };
        byte k1[16] = {13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3,
                        14,  5,  0, 12,  7 };

    for (i = 0; i < 256; i++) {
            c->k87[i] = k8[i >> 4] << 4 | k7[i & 15];
            c->k65[i] = k6[i >> 4] << 4 | k5[i & 15];
            c->k43[i] = k4[i >> 4] << 4 | k3[i & 15];
            c->k21[i] = k2[i >> 4] << 4 | k1[i & 15];
    }
}

static word32
f(gost_ctx *c,word32 x)
{
        x = c->k87[x>>24 & 255] << 24 | c->k65[x>>16 & 255] << 16 |
            c->k43[x>> 8 & 255] <<  8 | c->k21[x & 255];

    /* Rotate left 11 bits */
    return x<<11 | x>>(32-11);
}

void gostcrypt(gost_ctx *c, word32 *d){
    register word32 n1, n2; /* As named in the GOST */

       n1 = d[0];
       n2 = d[1];

    /* Instead of swapping halves, swap names each round */
       n2 ^= f(c,n1+c->k[0]); n1 ^= f(c,n2+c->k[1]);
       n2 ^= f(c,n1+c->k[2]); n1 ^= f(c,n2+c->k[3]);
       n2 ^= f(c,n1+c->k[4]); n1 ^= f(c,n2+c->k[5]);
       n2 ^= f(c,n1+c->k[6]); n1 ^= f(c,n2+c->k[7]);

       n2 ^= f(c,n1+c->k[0]); n1 ^= f(c,n2+c->k[1]);
       n2 ^= f(c,n1+c->k[2]); n1 ^= f(c,n2+c->k[3]);
       n2 ^= f(c,n1+c->k[4]); n1 ^= f(c,n2+c->k[5]);
       n2 ^= f(c,n1+c->k[6]); n1 ^= f(c,n2+c->k[7]);

       n2 ^= f(c,n1+c->k[0]); n1 ^= f(c,n2+c->k[1]);
       n2 ^= f(c,n1+c->k[2]); n1 ^= f(c,n2+c->k[3]);
       n2 ^= f(c,n1+c->k[4]); n1 ^= f(c,n2+c->k[5]);
       n2 ^= f(c,n1+c->k[6]); n1 ^= f(c,n2+c->k[7]);

       n2 ^= f(c,n1+c->k[7]); n1 ^= f(c,n2+c->k[6]);
       n2 ^= f(c,n1+c->k[5]); n1 ^= f(c,n2+c->k[4]);
```

```
        n2 ^= f(c,n1+c->k[3]); n1 ^= f(c,n2+c->k[2]);
        n2 ^= f(c,n1+c->k[1]); n1 ^= f(c,n2+c->k[0]);

        d[0] = n2; d[1] = n1;
}

void
gostdecrypt(gost_ctx *c, u4 *d){
     register word32 n1, n2; /* As named in the GOST */

        n1 = d[0]; n2 = d[1];

        n2 ^= f(c,n1+c->k[0]); n1 ^= f(c,n2+c->k[1]);
        n2 ^= f(c,n1+c->k[2]); n1 ^= f(c,n2+c->k[3]);
        n2 ^= f(c,n1+c->k[4]); n1 ^= f(c,n2+c->k[5]);
        n2 ^= f(c,n1+c->k[6]); n1 ^= f(c,n2+c->k[7]);

        n2 ^= f(c,n1+c->k[7]); n1 ^= f(c,n2+c->k[6]);
        n2 ^= f(c,n1+c->k[5]); n1 ^= f(c,n2+c->k[4]);
        n2 ^= f(c,n1+c->k[3]); n1 ^= f(c,n2+c->k[2]);
        n2 ^= f(c,n1+c->k[1]); n1 ^= f(c,n2+c->k[0]);

        n2 ^= f(c,n1+c->k[7]); n1 ^= f(c,n2+c->k[6]);
        n2 ^= f(c,n1+c->k[5]); n1 ^= f(c,n2+c->k[4]);
        n2 ^= f(c,n1+c->k[3]); n1 ^= f(c,n2+c->k[2]);
        n2 ^= f(c,n1+c->k[1]); n1 ^= f(c,n2+c->k[0]);

        n2 ^= f(c,n1+c->k[7]); n1 ^= f(c,n2+c->k[6]);
        n2 ^= f(c,n1+c->k[5]); n1 ^= f(c,n2+c->k[4]);
        n2 ^= f(c,n1+c->k[3]); n1 ^= f(c,n2+c->k[2]);
        n2 ^= f(c,n1+c->k[1]); n1 ^= f(c,n2+c->k[0]);

        d[0] = n2; d[1] = n1;
}

void gost_enc(gost_ctx *c, u4 *d, int blocks){
        int i;

        for(i=0;i<blocks;i++){
                gostcrypt(c,d);
                d+=2;
        }
}

void gost_dec(gost_ctx *c, u4 *d, int blocks){
        int i;

        for(i=0;i<blocks;i++){
                gostdecrypt(c,d);
                d+=2;
        }
}

void gost_key(gost_ctx *c, u4 *k){
        int i;
        for(i=0;i<8;i++) c->k[i]=k[i];
```

```
}

void gost_init(gost_ctx *c){
        kboxinit(c);
}

void gost_destroy(gost_ctx *c){
        int i;
        for(i=0;i<8;i++) c->k[i]=0;
}

void main(void){
        gost_ctx gc;
        u4 k[8],data[10];
        int i;

        /* Initialize GOST context. */
        gost_init(&gc);

        /* Prepare key--a simple key should be OK, with this many rounds! */
        for(i=0;i<8;i++) k[i] = i;
        gost_key(&gc,k);

        /* Try some test vectors. */
        data[0] = 0; data[1] = 0;
        gostcrypt(&gc,data);
        printf("Enc of zero vector:  %08lx %08lx\n",data[0],data[1]);
        gostcrypt(&gc,data);
        printf("Enc of above:        %08lx %08lx\n",data[0],data[1]);
        data[0] = 0xffffffff; data[1] = 0xffffffff;
        gostcrypt(&gc,data);
        printf("Enc of ones vector:  %08lx %08lx\n",data[0],data[1]);
        gostcrypt(&gc,data);
        printf("Enc of above:        %08lx %08lx\n",data[0],data[1]);

        /* Does gost_dec() properly reverse gost_enc()?  Do
           we deal OK with single-block lengths passed in gost_dec()?
           Do we deal OK with different lengths passed in? */

        /* Init data */
        for(i=0;i<10;i++) data[i]=i;

        /* Encrypt data as 5 blocks. */
        gost_enc(&gc,data,5);

        /* Display encrypted data. */
        for(i=0;i<10;i+=2) printf("Block %02d = %08lx %08lx\n",
                                  i/2,data[i],data[i+1]);

        /* Decrypt in different sized chunks. */
        gost_dec(&gc,data,1);
        gost_dec(&gc,data+2,4);
        printf("\n");

        /* Display decrypted data. */
```

```
        for(i=0;i<10;i+=2) printf("Block %02d = %08lx %08lx\n",
                                  i/2,data[i],data[i+1]);

        gost_destroy(&gc);
}
```

BLOWFISH

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#ifdef little_endian   /* Eg: Intel */
   #include <alloc.h>
#endif

#include <ctype.h>

#ifdef little_endian   /* Eg: Intel */
   #include <dir.h>
   #include <bios.h>
#endif

#ifdef big_endian
   #include <Types.h>
#endif

typedef struct {
        unsigned long S[4][256],P[18];
} blf_ctx;

#define MAXKEYBYTES 56          /* 448 bits */
// #define little_endian 1              /* Eg: Intel */
#define big_endian 1            /* Eg: Motorola */

void Blowfish_encipher(blf_ctx *,unsigned long *xl, unsigned long *xr);
void Blowfish_decipher(blf_ctx *,unsigned long *xl, unsigned long *xr);

#define N               16
#define noErr            0
#define DATAERROR         -1
#define KEYBYTES         8

FILE*         SubkeyFile;

unsigned long F(blf_ctx *bc, unsigned long x)
{
   unsigned short a;
   unsigned short b;
   unsigned short c;
   unsigned short d;
   unsigned long  y;
```

```
    d = x & 0x00FF;
    x >>= 8;
    c = x & 0x00FF;
    x >>= 8;
    b = x & 0x00FF;
    x >>= 8;
    a = x & 0x00FF;
    //y = ((S[0][a] + S[1][b]) ^ S[2][c]) + S[3][d];
    y = bc->S[0][a] + bc->S[1][b];
    y = y ^ bc->S[2][c];
    y = y + bc->S[3][d];

    return y;
}

void Blowfish_encipher(blf_ctx *c,unsigned long *xl, unsigned long *xr)
{
    unsigned long  Xl;
    unsigned long  Xr;
    unsigned long  temp;
    short          i;

    Xl = *xl;
    Xr = *xr;

    for (i = 0; i < N; ++i) {
        Xl = Xl ^ c->P[i];
        Xr = F(c,Xl) ^ Xr;

        temp = Xl;
        Xl = Xr;
        Xr = temp;
    }

    temp = Xl;
    Xl = Xr;
    Xr = temp;

    Xr = Xr ^ c->P[N];
    Xl = Xl ^ c->P[N + 1];

    *xl = Xl;
    *xr = Xr;
}

void Blowfish_decipher(blf_ctx *c, unsigned long *xl, unsigned long *xr)
{
    unsigned long  Xl;
    unsigned long  Xr;
    unsigned long  temp;
    short          i;

    Xl = *xl;
    Xr = *xr;
```

```
    for (i = N + 1; i > 1; --i) {
       Xl = Xl ^ c->P[i];
       Xr = F(c,Xl) ^ Xr;

       /* Exchange Xl and Xr */
       temp = Xl;
       Xl = Xr;
       Xr = temp;
    }

    /* Exchange Xl and Xr */
    temp = Xl;
    Xl = Xr;
    Xr = temp;

    Xr = Xr ^ c->P[1];
    Xl = Xl ^ c->P[0];

    *xl = Xl;
    *xr = Xr;
}

short InitializeBlowfish(blf_ctx *c, char key[], short keybytes)
{
   short          i;
   short          j;
   short          k;
   short          error;
   short          numread;
   unsigned long  data;
   unsigned long  datal;
   unsigned long  datar;

unsigned long ks0[] = {
0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7, 0xb8e1afed, 0x6a267e96,
0xba7c9045, 0xf12c7f99, 0x24a19947, 0xb3916cf7, 0x0801f2e2, 0x858efc16,
0x636920d8, 0x71574e69, 0xa458fea3, 0xf4933d7e, 0x0d95748f, 0x728eb658,
0x718bcd58, 0x82154aee, 0x7b54a41d, 0xc25a59b5, 0x9c30d539, 0x2af26013,
0xc5d1b023, 0x286085f0, 0xca417918, 0xb8db38ef, 0x8e79dcb0, 0x603a180e,
0x6c9e0e8b, 0xb01e8a3e, 0xd71577c1, 0xbd314b27, 0x78af2fda, 0x55605c60,
0xe65525f3, 0xaa55ab94, 0x57489862, 0x63e81440, 0x55ca396a, 0x2aab10b6,
0xb4cc5c34, 0x1141e8ce, 0xa15486af, 0x7c72e993, 0xb3ee1411, 0x636fbc2a,
0x2ba9c55d, 0x741831f6, 0xce5c3e16, 0x9b87931e, 0xafd6ba33, 0x6c24cf5c,
0x7a325381, 0x28958677, 0x3b8f4898, 0x6b4bb9af, 0xc4bfe81b, 0x66282193,
0x61d809cc, 0xfb21a991, 0x487cac60, 0x5dec8032, 0xef845d5d, 0xe98575b1,
0xdc262302, 0xeb651b88, 0x23893e81, 0xd396acc5, 0x0f6d6ff3, 0x83f44239,
0x2e0b4482, 0xa4842004, 0x69c8f04a, 0x9e1f9b5e, 0x21c66842, 0xf6e96c9a,
0x670c9c61, 0xabd388f0, 0x6a51a0d2, 0xd8542f68, 0x960fa728, 0xab5133a3,
0x6eef0b6c, 0x137a3be4, 0xba3bf050, 0x7efb2a98, 0xa1f1651d, 0x39af0176,
0x66ca593e, 0x82430e88, 0x8cee8619, 0x456f9fb4, 0x7d84a5c3, 0x3b8b5ebe,
0xe06f75d8, 0x85c12073, 0x401a449f, 0x56c16aa6, 0x4ed3aa62, 0x363f7706,
0x1bfedf72, 0x429b023d, 0x37d0d724, 0xd00a1248, 0xdb0fead3, 0x49f1c09b,
0x075372c9, 0x80991b7b, 0x25d479d8, 0xf6e8def7, 0xe3fe501a, 0xb6794c3b,
```

```
0x976ce0bd, 0x04c006ba, 0xc1a94fb6, 0x409f60c4, 0x5e5c9ec2, 0x196a2463,
0x68fb6faf, 0x3e6c53b5, 0x1339b2eb, 0x3b52ec6f, 0x6dfc511f, 0x9b30952c,
0xcc814544, 0xaf5ebd09, 0xbee3d004, 0xde334afd, 0x660f2807, 0x192e4bb3,
0xc0cba857, 0x45c8740f, 0xd20b5f39, 0xb9d3fbdb, 0x5579c0bd, 0x1a60320a,
0xd6a100c6, 0x402c7279, 0x679f25fe, 0xfb1fa3cc, 0x8ea5e9f8, 0xdb3222f8,
0x3c7516df, 0xfd616b15, 0x2f501ec8, 0xad0552ab, 0x323db5fa, 0xfd238760,
0x53317b48, 0x3e00df82, 0x9e5c57bb, 0xca6f8ca0, 0x1a87562e, 0xdf1769db,
0xd542a8f6, 0x287effc3, 0xac6732c6, 0x8c4f5573, 0x695b27b0, 0xbbca58c8,
0xe1ffa35d, 0xb8f011a0, 0x10fa3d98, 0xfd2183b8, 0x4afcb56c, 0x2dd1d35b,
0x9a53e479, 0xb6f84565, 0xd28e49bc, 0x4bfb9790, 0xe1ddf2da, 0xa4cb7e33,
0x62fb1341, 0xcee4c6e8, 0xef20cada, 0x36774c01, 0xd07e9efe, 0x2bf11fb4,
0x95dbda4d, 0xae909198, 0xeaad8e71, 0x6b93d5a0, 0xd08ed1d0, 0xafc725e0,
0x8e3c5b2f, 0x8e7594b7, 0x8ff6e2fb, 0xf2122b64, 0x8888b812, 0x900df01c,
0x4fad5ea0, 0x688fc31c, 0xd1cff191, 0xb3a8c1ad, 0x2f2f2218, 0xbe0e1777,
0xea752dfe, 0x8b021fa1, 0xe5a0cc0f, 0xb56f74e8, 0x18acf3d6, 0xce89e299,
0xb4a84fe0, 0xfd13e0b7, 0x7cc43b81, 0xd2ada8d9, 0x165fa266, 0x80957705,
0x93cc7314, 0x211a1477, 0xe6ad2065, 0x77b5fa86, 0xc75442f5, 0xfb9d35cf,
0xebcdaf0c, 0x7b3e89a0, 0xd6411bd3, 0xae1e7e49, 0x00250e2d, 0x2071b35e,
0x226800bb, 0x57b8e0af, 0x2464369b, 0xf009b91e, 0x5563911d, 0x59dfa6aa,
0x78c14389, 0xd95a537f, 0x207d5ba2, 0x02e5b9c5, 0x83260376, 0x6295cfa9,
0x11c81968, 0x4e734a41, 0xb3472dca, 0x7b14a94a, 0x1b510052, 0x9a532915,
0xd60f573f, 0xbc9bc6e4, 0x2b60a476, 0x81e67400, 0x08ba6fb5, 0x571be91f,
0xf296ec6b, 0x2a0dd915, 0xb6636521, 0xe7b9f9b6, 0xff34052e, 0xc5855664,
0x53b02d5d, 0xa99f8fa1, 0x08ba4799, 0x6e85076a};
unsigned long ks1[] = {
0x4b7a70e9, 0xb5b32944, 0xdb75092e, 0xc4192623, 0xad6ea6b0, 0x49a7df7d,
0x9cee60b8, 0x8fedb266, 0xecaa8c71, 0x699a17ff, 0x5664526c, 0xc2b19ee1,
0x193602a5, 0x75094c29, 0xa0591340, 0xe4183a3e, 0x3f54989a, 0x5b429d65,
0x6b8fe4d6, 0x99f73fd6, 0xa1d29c07, 0xefe830f5, 0x4d2d38e6, 0xf0255dc1,
0x4cdd2086, 0x8470eb26, 0x6382e9c6, 0x021ecc5e, 0x09686b3f, 0x3ebaefc9,
0x3c971814, 0x6b6a70a1, 0x687f3584, 0x52a0e286, 0xb79c5305, 0xaa500737,
0x3e07841c, 0x7fdeae5c, 0x8e7d44ec, 0x5716f2b8, 0xb03ada37, 0xf0500c0d,
0xf01c1f04, 0x0200b3ff, 0xae0cf51a, 0x3cb574b2, 0x25837a58, 0xdc0921bd,
0xd19113f9, 0x7ca92ff6, 0x94324773, 0x22f54701, 0x3ae5e581, 0x37c2dadc,
0xc8b57634, 0x9af3dda7, 0xa9446146, 0x0fd0030e, 0xecc8c73e, 0xa4751e41,
0xe238cd99, 0x3bea0e2f, 0x3280bba1, 0x183eb331, 0x4e548b38, 0x4f6db908,
0x6f420d03, 0xf60a04bf, 0x2cb81290, 0x24977c79, 0x5679b072, 0xbcaf89af,
0xde9a771f, 0xd9930810, 0xb38bae12, 0xdccf3f2e, 0x5512721f, 0x2e6b7124,
0x501adde6, 0x9f84cd87, 0x7a584718, 0x7408da17, 0xbc9f9abc, 0xe94b7d8c,
0xec7aec3a, 0xdb851dfa, 0x63094366, 0xc464c3d2, 0xef1c1847, 0x3215d908,
0xdd433b37, 0x24c2ba16, 0x12a14d43, 0x2a65c451, 0x50940002, 0x133ae4dd,
0x71dff89e, 0x10314e55, 0x81ac77d6, 0x5f11199b, 0x043556f1, 0xd7a3c76b,
0x3c11183b, 0x5924a509, 0xf28fe6ed, 0x97f1fbfa, 0x9ebabf2c, 0x1e153c6e,
0x86e34570, 0xeae96fb1, 0x860e5e0a, 0x5a3e2ab3, 0x771fe71c, 0x4e3d06fa,
0x2965dcb9, 0x99e71d0f, 0x803e89d6, 0x5266c825, 0x2e4cc978, 0x9c10b36a,
0xc6150eba, 0x94e2ea78, 0xa5fc3c53, 0x1e0a2df4, 0xf2f74ea7, 0x361d2b3d,
0x1939260f, 0x19c27960, 0x5223a708, 0xf71312b6, 0xebadfe6e, 0xeac31f66,
0xe3bc4595, 0xa67bc883, 0xb17f37d1, 0x018cff28, 0xc332ddef, 0xbe6c5aa5,
0x65582185, 0x68ab9802, 0xeeecea50f, 0xdb2f953b, 0x2aef7dad, 0x5b6e2f84,
0x1521b628, 0x29076170, 0xecdd4775, 0x619f1510, 0x13cca830, 0xeb61bd96,
0x0334fe1e, 0xaa0363cf, 0xb5735c90, 0x4c70a239, 0xd59e9e0b, 0xcbaade14,
0xeecc86bc, 0x60622ca7, 0x9cab5cab, 0xb2f3846e, 0x648b1eaf, 0x19bdf0ca,
0xa02369b9, 0x655abb50, 0x40685a32, 0x3c2ab4b3, 0x319ee9d5, 0xc021b8f7,
0x9b540b19, 0x875fa099, 0x95f7997e, 0x623d7da8, 0xf837889a, 0x97e32d77,
```

```
0x11ed935f, 0x16681281, 0x0e358829, 0xc7e61fd6, 0x96dedfa1, 0x7858ba99,
0x57f584a5, 0x1b227263, 0x9b83c3ff, 0x1ac24696, 0xcdb30aeb, 0x532e3054,
0x8fd948e4, 0x6dbc3128, 0x58ebf2ef, 0x34c6ffea, 0xfe28ed61, 0xee7c3c73,
0x5d4a14d9, 0xe864b7e3, 0x42105d14, 0x203e13e0, 0x45eee2b6, 0xa3aaabea,
0xdb6c4f15, 0xfacb4fd0, 0xc742f442, 0xef6abbb5, 0x654f3b1d, 0x41cd2105,
0xd81e799e, 0x86854dc7, 0xe44b476a, 0x3d816250, 0xcf62a1f2, 0x5b8d2646,
0xfc8883a0, 0xc1c7b6a3, 0x7f1524c3, 0x69cb7492, 0x47848a0b, 0x5692b285,
0x095bbf00, 0xad19489d, 0x1462b174, 0x23820e00, 0x58428d2a, 0x0c55f5ea,
0x1dadf43e, 0x233f7061, 0x3372f092, 0x8d937e41, 0xd65fecf1, 0x6c223bdb,
0x7cde3759, 0xcbee7460, 0x4085f2a7, 0xce77326e, 0xa6078084, 0x19f8509e,
0xe8efd855, 0x61d99735, 0xa969a7aa, 0xc50c06c2, 0x5a04abfc, 0x800bcadc,
0x9e447a2e, 0xc3453484, 0xfdd56705, 0x0e1e9ec9, 0xdb73dbd3, 0x105588cd,
0x675fda79, 0xe3674340, 0xc5c43465, 0x713e38d8, 0x3d28f89e, 0xf16dff20,
0x153e21e7, 0x8fb03d4a, 0xe6e39f2b, 0xdb83adf7};
unsigned long ks2[] = {
0xe93d5a68, 0x948140f7, 0xf64c261c, 0x94692934, 0x411520f7, 0x7602d4f7,
0xbcf46b2e, 0xd4a20068, 0xd4082471, 0x3320f46a, 0x43b7d4b7, 0x500061af,
0x1e39f62e, 0x97244546, 0x14214f74, 0xbf8b8840, 0x4d95fc1d, 0x96b591af,
0x70f4ddd3, 0x66a02f45, 0xbfbc09ec, 0x03bd9785, 0x7fac6dd0, 0x31cb8504,
0x96eb27b3, 0x55fd3941, 0xda2547e6, 0xabca0a9a, 0x28507825, 0x530429f4,
0x0a2c86da, 0xe9b66dfb, 0x68dc1462, 0xd7486900, 0x680ec0a4, 0x27a18dee,
0x4f3ffea2, 0xe887ad8c, 0xb58ce006, 0x7af4d6b6, 0xaace1e7c, 0xd3375fec,
0xce78a399, 0x406b2a42, 0x20fe9e35, 0xd9f385b9, 0xee39d7ab, 0x3b124e8b,
0x1dc9faf7, 0x4b6d1856, 0x26a36631, 0xeae397b2, 0x3a6efa74, 0xdd5b4332,
0x6841e7f7, 0xca7820fb, 0xfb0af54e, 0xd8feb397, 0x454056ac, 0xba489527,
0x55533a3a, 0x20838d87, 0xfe6ba9b7, 0xd096954b, 0x55a867bc, 0xa1159a58,
0xcca92963, 0x99e1db33, 0xa62a4a56, 0x3f3125f9, 0x5ef47e1c, 0x9029317c,
0xfdf8e802, 0x04272f70, 0x80bb155c, 0x05282ce3, 0x95c11548, 0xe4c66d22,
0x48c1133f, 0xc70f86dc, 0x07f9c9ee, 0x41041f0f, 0x404779a4, 0x5d886e17,
0x325f51eb, 0xd59bc0d1, 0xf2bcc18f, 0x41113564, 0x257b7834, 0x602a9c60,
0xdff8e8a3, 0x1f636c1b, 0x0e12b4c2, 0x02e1329e, 0xaf664fd1, 0xcad18115,
0x6b2395e0, 0x333e92e1, 0x3b240b62, 0xeebeb922, 0x85b2a20e, 0xe6ba0d99,
0xde720c8c, 0x2da2f728, 0xd0127845, 0x95b794fd, 0x647d0862, 0xe7ccf5f0,
0x5449a36f, 0x877d48fa, 0xc39dfd27, 0xf33e8d1e, 0x0a476341, 0x992eff74,
0x3a6f6eab, 0xf4f8fd37, 0xa812dc60, 0xa1ebddf8, 0x991be14c, 0xdb6e6b0d,
0xc67b5510, 0x6d672c37, 0x2765d43b, 0xdcd0e804, 0xf1290dc7, 0xcc00ffa3,
0xb5390f92, 0x690fed0b, 0x667b9ffb, 0xcedb7d9c, 0xa091cf0b, 0xd9155ea3,
0xbb132f88, 0x515bad24, 0x7b9479bf, 0x763bd6eb, 0x37392eb3, 0xcc115979,
0x8026e297, 0xf42e312d, 0x6842ada7, 0xc66a2b3b, 0x12754ccc, 0x782ef11c,
0x6a124237, 0xb79251e7, 0x06a1bbe6, 0x4bfb6350, 0x1a6b1018, 0x11caedfa,
0x3d25bdd8, 0xe2e1c3c9, 0x44421659, 0x0a121386, 0xd90cec6e, 0xd5abea2a,
0x64af674e, 0xda86a85f, 0xbebfe988, 0x64e4c3fe, 0x9dbc8057, 0xf0f7c086,
0x60787bf8, 0x6003604d, 0xd1fd8346, 0xf6381fb0, 0x7745ae04, 0xd736fccc,
0x83426b33, 0xf01eab71, 0xb0804187, 0x3c005e5f, 0x77a057be, 0xbde8ae24,
0x55464299, 0xbf582e61, 0x4e58f48f, 0xf2ddfda2, 0xf474ef38, 0x8789bdc2,
0x5366f9c3, 0xc8b38e74, 0xb475f255, 0x46fcd9b9, 0x7aeb2661, 0x8b1ddf84,
0x846a0e79, 0x915f95e2, 0x466e598e, 0x20b45770, 0x8cd55591, 0xc902de4c,
0xb90bace1, 0xbb8205d0, 0x11a86248, 0x7574a99e, 0xb77f19b6, 0xe0a9dc09,
0x662d09a1, 0xc4324633, 0xe85a1f02, 0x09f0be8c, 0x4a99a025, 0x1d6efe10,
0x1ab93d1d, 0x0ba5a4df, 0xa186f20f, 0x2868f169, 0xdcb7da83, 0x573906fe,
0xa1e2ce9b, 0x4fcd7f52, 0x50115e01, 0xa70683fa, 0xa002b5c4, 0x0de6d027,
0x9af88c27, 0x773f8641, 0xc3604c06, 0x61a806b5, 0xf0177a28, 0xc0f586e0,
0x006058aa, 0x30dc7d62, 0x11e69ed7, 0x2338ea63, 0x53c2dd94, 0xc2c21634,
0xbbcbee56, 0x90bcb6de, 0xebfc7da1, 0xce591d76, 0x6f05e409, 0x4b7c0188,
```

```
0x39720a3d, 0x7c927c24, 0x86e3725f, 0x724d9db9, 0x1ac15bb4, 0xd39eb8fc,
0xed545578, 0x08fca5b5, 0xd83d7cd3, 0x4dad0fc4, 0x1e50ef5e, 0xb161e6f8,
0xa28514d9, 0x6c51133c, 0x6fd5c7e7, 0x56e14ec4, 0x362abfce, 0xddc6c837,
0xd79a3234, 0x92638212, 0x670efa8e, 0x406000e0};
unsigned long ks3[] = {
0x3a39ce37, 0xd3faf5cf, 0xabc27737, 0x5ac52d1b, 0x5cb0679e, 0x4fa33742,
0xd3822740, 0x99bc9bbe, 0xd5118e9d, 0xbf0f7315, 0xd62d1c7e, 0xc700c47b,
0xb78c1b6b, 0x21a19045, 0xb26eb1be, 0x6a366eb4, 0x5748ab2f, 0xbc946e79,
0xc6a376d2, 0x6549c2c8, 0x530ff8ee, 0x468dde7d, 0xd5730a1d, 0x4cd04dc6,
0x2939bbdb, 0xa9ba4650, 0xac9526e8, 0xbe5ee304, 0xa1fad5f0, 0x6a2d519a,
0x63ef8ce2, 0x9a86ee22, 0xc089c2b8, 0x43242ef6, 0xa51e03aa, 0x9cf2d0a4,
0x83c061ba, 0x9be96a4d, 0x8fe51550, 0xba645bd6, 0x2826a2f9, 0xa73a3ae1,
0x4ba99586, 0xef5562e9, 0xc72fefd3, 0xf752f7da, 0x3f046f69, 0x77fa0a59,
0x80e4a915, 0x87b08601, 0x9b09e6ad, 0x3b3ee593, 0xe990fd5a, 0x9e34d797,
0x2cf0b7d9, 0x022b8b51, 0x96d5ac3a, 0x017da67d, 0xd1cf3ed6, 0x7c7d2d28,
0x1f9f25cf, 0xadf2b89b, 0x5ad6b472, 0x5a88f54c, 0xe029ac71, 0xe019a5e6,
0x47b0acfd, 0xed93fa9b, 0xe8d3c48d, 0x283b57cc, 0xf8d56629, 0x79132e28,
0x785f0191, 0xed756055, 0xf7960e44, 0xe3d35e8c, 0x15056dd4, 0x88f46dba,
0x03a16125, 0x0564f0bd, 0xc3eb9e15, 0x3c9057a2, 0x97271aec, 0xa93a072a,
0x1b3f6d9b, 0x1e6321f5, 0xf59c66fb, 0x26dcf319, 0x7533d928, 0xb155fdf5,
0x03563482, 0x8aba3cbb, 0x28517711, 0xc20ad9f8, 0xabcc5167, 0xccad925f,
0x4de81751, 0x3830dc8e, 0x379d5862, 0x9320f991, 0xea7a90c2, 0xfb3e7bce,
0x5121ce64, 0x774fbe32, 0xa8b6e37e, 0xc3293d46, 0x48de5369, 0x6413e680,
0xa2ae0810, 0xdd6db224, 0x69852dfd, 0x09072166, 0xb39a460a, 0x6445c0dd,
0x586cdecf, 0x1c20c8ae, 0x5bbef7dd, 0x1b588d40, 0xccd2017f, 0x6bb4e3bb,
0xdda26a7e, 0x3a59ff45, 0x3e350a44, 0xbcb4cdd5, 0x72eacea8, 0xfa6484bb,
0x8d6612ae, 0xbf3c6f47, 0xd29be463, 0x542f5d9e, 0xaec2771b, 0xf64e6370,
0x740e0d8d, 0xe75b1357, 0xf8721671, 0xaf537d5d, 0x4040cb08, 0x4eb4e2cc,
0x34d2466a, 0x0115af84, 0xe1b00428, 0x95983a1d, 0x06b89fb4, 0xce6ea048,
0x6f3f3b82, 0x3520ab82, 0x011a1d4b, 0x277227f8, 0x611560b1, 0xe7933fdc,
0xbb3a792b, 0x344525bd, 0xa08839e1, 0x51ce794b, 0x2f32c9b7, 0xa01fbac9,
0xe01cc87e, 0xbcc7d1f6, 0xcf0111c3, 0xa1e8aac7, 0x1a908749, 0xd44fbd9a,
0xd0dadecb, 0xd50ada38, 0x0339c32a, 0xc6913667, 0x8df9317c, 0xe0b12b4f,
0xf79e59b7, 0x43f5bb3a, 0xf2d519ff, 0x27d9459c, 0xbf97222c, 0x15e6fc2a,
0x0f91fc71, 0x9b941525, 0xfae59361, 0xceb69ceb, 0xc2a86459, 0x12baa8d1,
0xb6c1075e, 0xe3056a0c, 0x10d25065, 0xcb03a442, 0xe0ec6e0e, 0x1698db3b,
0x4c98a0be, 0x3278e964, 0x9f1f9532, 0xe0d392df, 0xd3a0342b, 0x8971f21e,
0x1b0a7441, 0x4ba3348c, 0xc5be7120, 0xc37632d8, 0xdf359f8d, 0x9b992f2e,
0xe60b6f47, 0x0fe3f11d, 0xe54cda54, 0x1edad891, 0xce6279cf, 0xcd3e7e6f,
0x1618b166, 0xfd2c1d05, 0x848fd2c5, 0xf6fb2299, 0xf523f357, 0xa6327623,
0x93a83531, 0x56cccd02, 0xacf08162, 0x5a75ebb5, 0x6e163697, 0x88d273cc,
0xde966292, 0x81b949d0, 0x4c50901b, 0x71c65614, 0xe6c6c7bd, 0x327a140a,
0x45e1d006, 0xc3f27b9a, 0xc9aa53fd, 0x62a80f00, 0xbb25bfe2, 0x35bdd2f6,
0x71126905, 0xb2040222, 0xb6cbcf7c, 0xcd769c2b, 0x53113ec0, 0x1640e3d3,
0x38abbd60, 0x2547adf0, 0xba38209c, 0xf746ce76, 0x77afa1c5, 0x20756060,
0x85cbfe4e, 0x8ae88dd8, 0x7aaaf9b0, 0x4cf9aa7e, 0x1948c25c, 0x02fb8a8c,
0x01c36ae4, 0xd6ebe1f9, 0x90d4f869, 0xa65cdea0, 0x3f09252d, 0xc208e69f,
0xb74e6132, 0xce77e25b, 0x578fdfe3, 0x3ac372e6};

/* Initialize s-boxes without file read. */
        for(i=0;i<256;i++){
                c->S[0][i] = ks0[i];
                c->S[1][i] = ks1[i];
                c->S[2][i] = ks2[i];
```

```
                c->S[3][i] = ks3[i];
        }

        j = 0;
        for (i = 0; i < N + 2; ++i) {
                data = 0x00000000;
                for (k = 0; k < 4; ++k) {
                        data = (data << 8) | key[j];
                        j = j + 1;
                        if (j >= keybytes) {
                                j = 0;
                        }
                }
        c->P[i] = c->P[i] ^ data;
        }

      datal = 0x00000000;
        datar = 0x00000000;

        for (i = 0; i < N + 2; i += 2) {
                Blowfish_encipher(c,&datal, &datar);

                c->P[i] = datal;
                c->P[i + 1] = datar;
        }

        for (i = 0; i < 4; ++i) {
                for (j = 0; j < 256; j += 2) {

                        Blowfish_encipher(c,&datal, &datar);

                        c->S[i][j] = datal;
                        c->S[i][j + 1] = datar;
                }
        }
}

void blf_key(blf_ctx *c, char *k, int len){
        InitializeBlowfish(c,k,len);
}

void blf_enc(blf_ctx *c, unsigned long *data, int blocks){
        unsigned long *d;
        int i;

        d = data;
        for(i=0;i<blocks;i++){
                Blowfish_encipher(c,d,d+1);
                d += 2;
        }
}

void blf_dec(blf_ctx *c, unsigned long *data, int blocks){
        unsigned long *d;
        int i;
```

```
        d = data;
        for(i=0;i<blocks;i++){
                Blowfish_decipher(c,d,d+1);
                d += 2;
        }
}

void main(void){
        blf_ctx c;
        char key[]="AAAAA";
        unsigned long data[10];
        int i;

        for(i=0;i<10;i++) data[i] = i;

        blf_key(&c,key,5);
        blf_enc(&c,data,5);
        blf_dec(&c,data,1);
        blf_dec(&c,data+2,4);
        for(i=0;i<10;i+=2) printf("Block %01d decrypts to: %08lx %08lx.\n",
                                  i/2,data[i],data[i+1]);
}
```

3-Way

```
#define   STRT_E   0x0b0b /* round constant of first encryption round */
#define   STRT_D   0xb1b1 /* round constant of first decryption round */
#define     NMBR       11 /* number of rounds is 11                    */

typedef   unsigned long int  word32 ;
                /* the program only works correctly if long = 32bits */
typedef unsigned long u4;
typedef unsigned char u1;

typedef struct {
        u4 k[3],ki[3], ercon[NMBR+1],drcon[NMBR+1];
} twy_ctx;

/* Note:  encrypt and decrypt expect full blocks--padding blocks is
          caller's responsibility.  All bulk encryption is done in
          ECB mode by these calls.  Other modes may be added easily
          enough.                                                   */

/* destroy:  Context. */
/* Scrub context of all sensitive data. */
void twy_destroy(twy_ctx *);

/* encrypt:  Context, ptr to data block, # of blocks. */
void twy_enc(twy_ctx *, u4 *, int);

/* decrypt:  Context, ptr to data block, # of blocks. */
void twy_dec(twy_ctx *, u4 *, int);
```

```
/* key:  Context, ptr to key data. */
void twy_key(twy_ctx *, u4 *);

/* ACCODE--------------------------------------------------------------- */
/* End of AC code prototypes and structures.                           */
/* --------------------------------------------------------------------- */

void mu(word32 *a)       /* inverts the order of the bits of a */
{
int i ;
word32 b[3] ;

b[0] = b[1] = b[2] = 0 ;
for( i=0 ; i<32 ; i++ )
   {
   b[0] <<= 1 ; b[1] <<= 1 ; b[2] <<= 1 ;
   if(a[0]&1) b[2] |= 1 ;
   if(a[1]&1) b[1] |= 1 ;
   if(a[2]&1) b[0] |= 1 ;
   a[0] >>= 1 ; a[1] >>= 1 ; a[2] >>= 1 ;
   }

a[0] = b[0] ;      a[1] = b[1] ;      a[2] = b[2] ;
}

void gamma(word32 *a)   /* the nonlinear step */
{
word32 b[3] ;

b[0] = a[0] ^ (a[1]|(~a[2])) ;
b[1] = a[1] ^ (a[2]|(~a[0])) ;
b[2] = a[2] ^ (a[0]|(~a[1])) ;

a[0] = b[0] ;      a[1] = b[1] ;      a[2] = b[2] ;
}

void theta(word32 *a)    /* the linear step */
{
word32 b[3];

b[0] = a[0] ^  (a[0]>>16) ^ (a[1]<<16) ^     (a[1]>>16) ^ (a[2]<<16) ^
               (a[1]>>24) ^ (a[2]<<8)  ^     (a[2]>>8)  ^ (a[0]<<24) ^
               (a[2]>>16) ^ (a[0]<<16) ^     (a[2]>>24) ^ (a[0]<<8)  ;
b[1] = a[1] ^  (a[1]>>16) ^ (a[2]<<16) ^     (a[2]>>16) ^ (a[0]<<16) ^
               (a[2]>>24) ^ (a[0]<<8)  ^     (a[0]>>8)  ^ (a[1]<<24) ^
               (a[0]>>16) ^ (a[1]<<16) ^     (a[0]>>24) ^ (a[1]<<8)  ;
b[2] = a[2] ^  (a[2]>>16) ^ (a[0]<<16) ^     (a[0]>>16) ^ (a[1]<<16) ^
               (a[0]>>24) ^ (a[1]<<8)  ^     (a[1]>>8)  ^ (a[2]<<24) ^
               (a[1]>>16) ^ (a[2]<<16) ^     (a[1]>>24) ^ (a[2]<<8)  ;

a[0] = b[0] ;      a[1] = b[1] ;      a[2] = b[2] ;
}

void pi_1(word32 *a)
```

```
{
a[0] = (a[0]>>10) ^ (a[0]<<22);
a[2] = (a[2]<<1)  ^ (a[2]>>31);
}

void pi_2(word32 *a)
{
a[0] = (a[0]<<1)  ^ (a[0]>>31);
a[2] = (a[2]>>10) ^ (a[2]<<22);
}

void rho(word32 *a)    /* the round function       */
{
theta(a) ;
pi_1(a) ;
gamma(a) ;
pi_2(a) ;
}

void rndcon_gen(word32 strt,word32 *rtab)
{                              /* generates the round constants */
int i ;

for(i=0 ; i<=NMBR ; i++ )
   {
   rtab[i] = strt ;
   strt <<= 1 ;
   if( strt&0x10000 ) strt ^= 0x11011 ;
   }
}

/* Modified slightly to fit the caller's needs. */
void encrypt(twy_ctx *c, word32 *a)
{
char i ;
for( i=0 ; i<NMBR ; i++ )
   {
   a[0] ^= c->k[0] ^ (c->ercon[i]<<16) ;
   a[1] ^= c->k[1] ;
   a[2] ^= c->k[2] ^ c->ercon[i] ;
   rho(a) ;
   }
a[0] ^= c->k[0] ^ (c->ercon[NMBR]<<16) ;
a[1] ^= c->k[1] ;
a[2] ^= c->k[2] ^ c->ercon[NMBR] ;
theta(a) ;
}

/* Modified slightly to meet caller's needs. */
void decrypt(twy_ctx *c, word32 *a)
{
char i ;

mu(a) ;
```

```
for( i=0 ; i<NMBR ; i++ )
    {
    a[0] ^= c->ki[0] ^ (c->drcon[i]<<16) ;
    a[1] ^= c->ki[1] ;
    a[2] ^= c->ki[2] ^ c->drcon[i] ;
    rho(a) ;
    }
a[0] ^= c->ki[0] ^ (c->drcon[NMBR]<<16) ;
a[1] ^= c->ki[1] ;
a[2] ^= c->ki[2] ^ c->drcon[NMBR] ;
theta(a) ;
mu(a) ;
}

void twy_key(twy_ctx *c, u4 *key){
        c->ki[0] = c->k[0] = key[0];
        c->ki[1] = c->k[1] = key[1];
        c->ki[2] = c->k[2] = key[2];
        theta(c->ki);
        mu(c->ki);
        rndcon_gen(STRT_E,c->ercon);
        rndcon_gen(STRT_D,c->drcon);

}

/* Encrypt in ECB mode. */
void twy_enc(twy_ctx *c, u4 *data, int blkcnt){
        u4 *d;
        int i;

        d = data;
        for(i=0;i<blkcnt;i++) {
                encrypt(c,d);
                d +=3;
        }
}

/* Decrypt in ECB mode. */
void twy_dec(twy_ctx *c, u4 *data, int blkcnt){
        u4 *d;
        int i;

        d = data;
        for(i=0;i<blkcnt;i++){
                decrypt(c,d);
                d+=3;
        }
}

/* Scrub sensitive values from memory before deallocating. */
void twy_destroy(twy_ctx *c){
        int i;

        for(i=0;i<3;i++) c->k[i] = c->ki[i] = 0;
```

```
}

void printvec(char *chrs, word32 *d){
        printf("%20s : %08lx %08lx %08lx \n",chrs,d[2],d[1],d[0]);
}

main()
{
twy_ctx gc;
word32 a[9],k[3];
int i;

/* Test vector 1. */

k[0]=k[1]=k[2]=0;
a[0]=a[1]=a[2]=1;
twy_key(&gc,k);

printf("**********\n");
printvec("KEY = ",k);
printvec("PLAIN = ",a);
encrypt(&gc,a);
printvec("CIPHER = ",a);

/* Test vector 2. */

k[0]=6;k[1]=5;k[2]=4;
a[0]=3;a[1]=2;a[2]=1;
twy_key(&gc,k);

printf("**********\n");
printvec("KEY = ",k);
printvec("PLAIN = ",a);
encrypt(&gc,a);
printvec("CIPHER = ",a);

/* Test vector 3. */

k[2]=0xbcdef012;k[1]=0x456789ab;k[0]=0xdef01234;
a[2]=0x01234567;a[1]=0x9abcdef0;a[0]=0x23456789;
twy_key(&gc,k);

printf("**********\n");
printvec("KEY = ",k);
printvec("PLAIN = ",a);
encrypt(&gc,a);
printvec("CIPHER = ",a);

/* Test vector 4. */

k[2]=0xcab920cd;k[1]=0xd6144138;k[0]=0xd2f05b5e;
a[2]=0xad21ecf7;a[1]=0x83ae9dc4;a[0]=0x4059c76e;
twy_key(&gc,k);

printf("**********\n");
```

```
printvec("KEY = ",k);
printvec("PLAIN = ",a);
encrypt(&gc,a);
printvec("CIPHER = ",a);

/*  TEST VALUES

key        : 00000000 00000000 00000000
plaintext  : 00000001 00000001 00000001
ciphertext : ad21ecf7 83ae9dc4 4059c76e

key        : 00000004 00000005 00000006
plaintext  : 00000001 00000002 00000003
ciphertext : cab920cd d6144138 d2f05b5e

key        : bcdef012 456789ab def01234
plaintext  : 01234567 9abcdef0 23456789
ciphertext : 7cdb76b2 9cdddb6d 0aa55dbb

key        : cab920cd d6144138 d2f05b5e
plaintext  : ad21ecf7 83ae9dc4 4059c76e
ciphertext : 15b155ed 6b13f17c 478ea871

*/

/* Enc/dec test: */
for(i=0;i<9;i++) a[i]=i;
twy_enc(&gc,a,3);
for(i=0;i<9;i+=3) printf("Block %01d encrypts to %08lx %08lx %08lx\n",
                         i/3,a[i],a[i+1],a[i+2]);

twy_dec(&gc,a,2);
twy_dec(&gc,a+6,1);

 for(i=0;i<9;i+=3) printf("Block %01d decrypts to %08lx %08lx %08lx\n",
                         i/3,a[i],a[i+1],a[i+2]);
}
```

RC5

```
#include <stdio.h>

/* An RC5 context needs to know how many rounds it has, and its subkeys. */
typedef struct {
        u4 *xk;
        int nr;
} rc5_ctx;

/* Where possible, these should be replaced with actual rotate instructions.
   For Turbo C++, this is done with _lrotl and _lrotr. */

#define ROTL32(X,C) (((X)<<(C))|((X)>>(32-(C))))
#define ROTR32(X,C) (((X)>>(C))|((X)<<(32-(C))))
```

```
/* Function prototypes for dealing with RC5 basic operations. */
void rc5_init(rc5_ctx *, int);
void rc5_destroy(rc5_ctx *);
void rc5_key(rc5_ctx *, u1 *, int);
void rc5_encrypt(rc5_ctx *, u4 *, int);
void rc5_decrypt(rc5_ctx *, u4 *, int);

/* Function implementations for RC5. */

/* Scrub out all sensitive values. */
void rc5_destroy(rc5_ctx *c){
        int i;
     for(i=0;i<(c->nr)*2+2;i++) c->xk[i]=0;
     free(c->xk);
}

/* Allocate memory for rc5 context's xk and such. */
void rc5_init(rc5_ctx *c, int rounds){
     c->nr = rounds;
     c->xk = (u4 *) malloc(4*(rounds*2+2));
}

void rc5_encrypt(rc5_ctx *c, u4 *data, int blocks){
        u4 *d,*sk;
        int h,i,rc;

     d = data;
        sk = (c->xk)+2;
        for(h=0;h<blocks;h++){
                d[0] += c->xk[0];
                d[1] += c->xk[1];
                for(i=0;i<c->nr*2;i+=2){
                        d[0] ^= d[1];
                        rc = d[1] & 31;
                        d[0] = ROTL32(d[0],rc);
                        d[0] += sk[i];
               d[1] ^= d[0];
                        rc = d[0] & 31;
                        d[1] = ROTL32(d[1],rc);
                        d[1] += sk[i+1];
/*printf("Round %03d : %08lx %08lx  sk= %08lx %08lx\n",i/2,
                                d[0],d[1],sk[i],sk[i+1]);*/
                }
          d+=2;
        }
}

void rc5_decrypt(rc5_ctx *c, u4 *data, int blocks){
     u4 *d,*sk;
        int h,i,rc;

     d = data;
        sk = (c->xk)+2;
     for(h=0;h<blocks;h++){
                for(i=c->nr*2-2;i>=0;i-=2){
```

```
/*printf("Round %03d: %08lx %08lx  sk: %08lx %08lx\n",
        i/2,d[0],d[1],sk[i],sk[i+1]); */
                        d[1] -= sk[i+1];
                        rc = d[0] & 31;
                        d[1] = ROTR32(d[1],rc);
                        d[1] ^= d[0];

                        d[0] -= sk[i];
                        rc = d[1] & 31;
                        d[0] = ROTR32(d[0],rc);
                d[0] ^= d[1];
                 }
                d[0] -= c->xk[0];
                d[1] -= c->xk[1];
        d+=2;
     }
}

void rc5_key(rc5_ctx *c, u1 *key, int keylen){
     u4 *pk,A,B; /* padded key */
     int xk_len, pk_len, i, num_steps,rc;
     u1 *cp;

     xk_len = c->nr*2 + 2;
     pk_len = keylen/4;
     if((keylen%4)!=0) pk_len += 1;

     pk = (u4 *) malloc(pk_len * 4);
     if(pk==NULL) {
          printf("An error occurred!\n");
          exit(-1);
     }

     /* Initialize pk -- this should work on Intel machines, anyway.... */
     for(i=0;i<pk_len;i++) pk[i]=0;
     cp = (u1 *)pk;
     for(i=0;i<keylen;i++) cp[i]=key[i];

     /* Initialize xk. */
     c->xk[0] = 0xb7e15163; /* P32 */
     for(i=1;i<xk_len;i++) c->xk[i] = c->xk[i-1] + 0x9e3779b9; /* Q32 */

     /* TESTING */
     A = B = 0;
     for(i=0;i<xk_len;i++) {
          A = A + c->xk[i];
          B = B ^ c->xk[i];
     }

     /* Expand key into xk. */
     if(pk_len>xk_len) num_steps = 3*pk_len;else num_steps = 3*xk_len;

     A = B = 0;
     for(i=0;i<num_steps;i++){
          A = c->xk[i%xk_len] = ROTL32(c->xk[i%xk_len] + A + B,3);
          rc = (A+B) & 31;
```

```
            B = pk[i%pk_len] = ROTL32(pk[i%pk_len] + A + B,rc);

        }

        /* Clobber sensitive data before deallocating memory. */
        for(i=0;i<pk_len;i++) pk[i] =0;

        free(pk);
}

void main(void){
        rc5_ctx c;
        u4 data[8];
        char key[] = "ABCDE";
        int i;

        printf("------------------------------------------------\n");

           for(i=0;i<8;i++) data[i] = i;
        rc5_init(&c,10); /* 10 rounds */
        rc5_key(&c,key,5);

           rc5_encrypt(&c,data,4);
           printf("Encryptions:\n");
           for(i=0;i<8;i+=2) printf("Block %01d = %08lx %08lx\n",
                                    i/2,data[i],data[i+1]);
           rc5_decrypt(&c,data,2);
        rc5_decrypt(&c,data+4,2);
           printf("Decryptions:\n");
           for(i=0;i<8;i+=2) printf("Block %01d = %08lx %08lx\n",
                                    i/2,data[i],data[i+1]);

}
```

A5

```
typedef struct {
        unsigned long r1,r2,r3;
} a5_ctx;

static int threshold(r1, r2, r3)
unsigned int r1;
unsigned int r2;
unsigned int r3;
{
int total;

  total = (((r1 >>  9) & 0x1) == 1) +
          (((r2 >> 11) & 0x1) == 1) +
          (((r3 >> 11) & 0x1) == 1);

  if (total > 1)
    return (0);
```

```
    else
      return (1);
  }

  unsigned long clock_r1(ctl, r1)
  int ctl;
  unsigned long r1;
  {
  unsigned long feedback;

    ctl ^= ((r1 >> 9) & 0x1);
    if (ctl)
    {
      feedback = (r1 >> 18) ^ (r1 >> 17) ^ (r1 >> 16) ^ (r1 >> 13);
      r1 = (r1 << 1) & 0x7ffff;
      if (feedback & 0x01)
        r1 ^= 0x01;
    }
    return (r1);
  }

  unsigned long clock_r2(ctl, r2)
  int ctl;
  unsigned long r2;
  {
  unsigned long feedback;

    ctl ^= ((r2 >> 11) & 0x1);
    if (ctl)
    {
      feedback = (r2 >> 21) ^ (r2 >> 20) ^ (r2 >> 16) ^ (r2 >> 12);
      r2 = (r2 << 1) & 0x3fffff;
      if (feedback & 0x01)
        r2 ^= 0x01;
    }
    return (r2);
  }

  unsigned long clock_r3(ctl, r3)
  int ctl;
  unsigned long r3;
  {
  unsigned long feedback;

    ctl ^= ((r3 >> 11) & 0x1);
    if (ctl)
    {
      feedback = (r3 >> 22) ^ (r3 >> 21) ^ (r3 >> 18) ^ (r3 >> 17);
      r3 = (r3 << 1) & 0x7fffff;
      if (feedback & 0x01)
        r3 ^= 0x01;
    }
    return (r3);
  }
```

```
int keystream(key, frame, alice, bob)
unsigned char *key;   /* 64 bit session key              */
unsigned long frame;  /* 22 bit frame sequence number    */
unsigned char *alice; /* 114 bit Alice to Bob key stream */
unsigned char *bob;   /* 114 bit Bob to Alice key stream */
{
unsigned long r1;   /* 19 bit shift register */
unsigned long r2;   /* 22 bit shift register */
unsigned long r3;   /* 23 bit shift register */
int i;              /* counter for loops     */
int clock_ctl;      /* xored with clock enable on each shift register */
unsigned char *ptr; /* current position in keystream */
unsigned char byte; /* byte of keystream being assembled */
unsigned int bits;  /* number of bits of keystream in byte */
unsigned int bit;   /* bit output from keystream generator */

  /* Initialise shift registers from session key */

  r1 = (key[0] | (key[1] << 8) | (key[2] << 16) ) & 0x7ffff;
  r2 = ((key[2] >> 3) | (key[3] << 5) | (key[4] << 13) | (key[5] << 21)) &
0x3fffff;
  r3 = ((key[5] >> 1) | (key[6] << 7) | (key[7] << 15) ) & 0x7fffff;

  /* Merge frame sequence number into shift register state, by xor'ing it
   * into the feedback path
   */

  for (i=0;i<22;i++)
  {
    clock_ctl = threshold(r1, r2, r2);
    r1 = clock_r1(clock_ctl, r1);
    r2 = clock_r2(clock_ctl, r2);
    r3 = clock_r3(clock_ctl, r3);
    if (frame & 1)
    {
      r1 ^= 1;
      r2 ^= 1;
      r3 ^= 1;
    }
    frame = frame >> 1;
  }

  /* Run shift registers for 100 clock ticks to allow frame number to
   * be diffused into all the bits of the shift registers
   */

  for (i=0;i<100;i++)
  {
    clock_ctl = threshold(r1, r2, r2);
    r1 = clock_r1(clock_ctl, r1);
    r2 = clock_r2(clock_ctl, r2);
    r3 = clock_r3(clock_ctl, r3);
  }

  /* Produce 114 bits of Alice->Bob key stream */
```

```
ptr = alice;
bits = 0;
byte = 0;
for (i=0;i<114;i++)
{
  clock_ctl = threshold(r1, r2, r2);
  r1 = clock_r1(clock_ctl, r1);
  r2 = clock_r2(clock_ctl, r2);
  r3 = clock_r3(clock_ctl, r3);

  bit = ((r1 >> 18) ^ (r2 >> 21) ^ (r3 >> 22)) & 0x01;
  byte = (byte << 1) | bit;
  bits++;
  if (bits == 8)
  {
    *ptr = byte;
    ptr++;
    bits = 0;
    byte = 0;
  }
}
if (bits)
  *ptr = byte;

/* Run shift registers for another 100 bits to hide relationship between
 * Alice->Bob key stream and Bob->Alice key stream.
 */

for (i=0;i<100;i++)
{
  clock_ctl = threshold(r1, r2, r2);
  r1 = clock_r1(clock_ctl, r1);
  r2 = clock_r2(clock_ctl, r2);
  r3 = clock_r3(clock_ctl, r3);
}

/* Produce 114 bits of Bob->Alice key stream */

ptr = bob;
bits = 0;
byte = 0;
for (i=0;i<114;i++)
{
  clock_ctl = threshold(r1, r2, r2);
  r1 = clock_r1(clock_ctl, r1);
  r2 = clock_r2(clock_ctl, r2);
  r3 = clock_r3(clock_ctl, r3);

  bit = ((r1 >> 18) ^ (r2 >> 21) ^ (r3 >> 22)) & 0x01;
  byte = (byte << 1) | bit;
  bits++;
  if (bits == 8)
  {
    *ptr = byte;
```

```
        ptr++;
        bits = 0;
        byte = 0;
      }
    }
    if (bits)
      *ptr = byte;

    return (0);

}

void a5_key(a5_ctx *c, char *k){
        c->r1 = k[0]<<11|k[1]<<3 | k[2]>>5          ; /* 19 */
        c->r2 = k[2]<<17|k[3]<<9 | k[4]<<1 | k[5]>>7; /* 22 */
        c->r3 = k[5]<<15|k[6]<<8 | k[7]             ; /* 23 */
}

/* Step one bit in A5, return 0 or 1 as output bit. */
int a5_step(a5_ctx *c){
        int control;
        control = threshold(c->r1,c->r2,c->r3);
        c->r1 = clock_r1(control,c->r1);
        c->r2 = clock_r2(control,c->r2);
        c->r3 = clock_r3(control,c->r3);
        return( (c->r1^c->r2^c->r3)&1);
}

/* Encrypts a buffer of len bytes. */
void a5_encrypt(a5_ctx *c, char *data, int len){
        int i,j;
        char t;

        for(i=0;i<len;i++){
                for(j=0;j<8;j++) t = t<<1 | a5_step(c);
                data[i]^=t;
        }
}

void a5_decrypt(a5_ctx *c, char *data, int len){
        a5_encrypt(c,data,len);
}

void main(void){
        a5_ctx c;
        char data[100];
        char key[] = {1,2,3,4,5,6,7,8};
        int i,flag;

        for(i=0;i<100;i++) data[i] = i;

        a5_key(&c,key);
        a5_encrypt(&c,data,100);

        a5_key(&c,key);
```

```
        a5_decrypt(&c,data,1);
        a5_decrypt(&c,data+1,99);

        flag = 0;
        for(i=0;i<100;i++) if(data[i]!=i)flag = 1;
        if(flag)printf("Decrypt failed\n"); else printf("Decrypt succeeded\n");
}
```

SEAL

```
#undef SEAL_DEBUG

#define ALG_OK 0
#define ALG_NOTOK 1
#define WORDS_PER_SEAL_CALL 1024

typedef struct {
     unsigned long t[520]; /* 512 rounded up to a multiple of 5 + 5*/
     unsigned long s[265]; /* 256 rounded up to a multiple of 5 + 5*/
     unsigned long r[20];  /* 16 rounded up to multiple of 5 */
        unsigned long counter; /* 32-bit synch value. */
        unsigned long ks_buf[WORDS_PER_SEAL_CALL];
        int ks_pos;
} seal_ctx;

#define ROT2(x) (((x) >> 2) | ((x) << 30))
#define ROT9(x) (((x) >> 9) | ((x) << 23))
#define ROT8(x) (((x) >> 8) | ((x) << 24))
#define ROT16(x) (((x) >> 16) | ((x) << 16))
#define ROT24(x) (((x) >> 24) | ((x) << 8))
#define ROT27(x) (((x) >> 27) | ((x) << 5))

#define WORD(cp)  ((cp[0] << 24)|(cp[1] << 16)|(cp[2] << 8)|(cp[3]))

#define F1(x, y, z) (((x) & (y)) | ((~(x)) & (z)))
#define F2(x, y, z) ((x)^(y)^(z))
#define F3(x, y, z) (((x) & (y)) | ((x) & (z)) | ((y) & (z)))
#define F4(x, y, z) ((x)^(y)^(z))

int g(in, i, h)
unsigned char *in;
int i;
unsigned long *h;
{
unsigned long h0;
unsigned long h1;
unsigned long h2;
unsigned long h3;
unsigned long h4;
unsigned long a;
unsigned long b;
unsigned long c;
unsigned long d;
unsigned long e;
```

```
unsigned char *kp;
unsigned long w[80];
unsigned long temp;

     kp = in;
     h0 = WORD(kp); kp += 4;
     h1 = WORD(kp); kp += 4;
     h2 = WORD(kp); kp += 4;
     h3 = WORD(kp); kp += 4;
     h4 = WORD(kp); kp += 4;

     w[0] = i;
     for (i=1;i<16;i++)
          w[i] = 0;
     for (i=16;i<80;i++)
          w[i] = w[i-3]^w[i-8]^w[i-14]^w[i-16];

     a = h0;
     b = h1;
     c = h2;
     d = h3;
     e = h4;

     for (i=0;i<20;i++)
     {
          temp = ROT27(a) + F1(b, c, d) + e + w[i] + 0x5a827999;
          e = d;
          d = c;
          c = ROT2(b);
          b = a;
          a = temp;
     }
     for (i=20;i<40;i++)
     {
          temp = ROT27(a) + F2(b, c, d) + e + w[i] + 0x6ed9eba1;
          e = d;
          d = c;
          c = ROT2(b);
          b = a;
          a = temp;
     }
     for (i=40;i<60;i++)
     {
          temp = ROT27(a) + F3(b, c, d) + e + w[i] + 0x8f1bbcdc;
          e = d;
          d = c;
          c = ROT2(b);
          b = a;
          a = temp;
     }
     for (i=60;i<80;i++)
     {
          temp = ROT27(a) + F4(b, c, d) + e + w[i] + 0xca62c1d6;
          e = d;
          d = c;
```

```
            c = ROT2(b);
            b = a;
            a = temp;
        }
        h[0] = h0+a;
        h[1] = h1+b;
        h[2] = h2+c;
        h[3] = h3+d;
        h[4] = h4+e;

        return (ALG_OK);
}

unsigned long gamma(a, i)
unsigned char *a;
int i;
{
unsigned long h[5];

     (void) g(a, i/5, h);
     return h[i % 5];
}

int  seal_init(seal_ctx *result, unsigned char *key)
{
int i;
unsigned long h[5];

     for (i=0;i<510;i+=5)
          g(key, i/5, &(result->t[i]));
     /* horrible special case for the end */
     g(key, 510/5, h);
     for (i=510;i<512;i++)
          result->t[i] = h[i-510];
     /* 0x1000 mod 5 is +1, so have horrible special case for the start */
     g(key, (-1+0x1000)/5, h);
     for (i=0;i<4;i++)
          result->s[i] = h[i+1];
     for (i=4;i<254;i+=5)
          g(key, (i+0x1000)/5, &(result->s[i]));
     /* horrible special case for the end */
     g(key, (254+0x1000)/5, h);
     for (i=254;i<256;i++)
          result->s[i] = h[i-254];
     /* 0x2000 mod 5 is +2, so have horrible special case at the start */
     g(key, (-2+0x2000)/5, h);
     for (i=0;i<3;i++)
          result->r[i] = h[i+2];
     for (i=3;i<13;i+=5)
          g(key, (i+0x2000)/5, &(result->r[i]));
     /* horrible special case for the end */
     g(key, (13+0x2000)/5, h);
     for (i=13;i<16;i++)
          result->r[i] = h[i-13];
     return (ALG_OK);
```

```
}

int seal(seal_ctx *key, unsigned long in, unsigned long *out)
{
int i;
int j;
int l;
unsigned long a;
unsigned long b;
unsigned long c;
unsigned long d;
unsigned short p;
unsigned short q;
unsigned long n1;
unsigned long n2;
unsigned long n3;
unsigned long n4;
unsigned long *wp;

     wp = out;

     for (l=0;l<4;l++)
     {
          a = in ^ key->r[4*l];
          b = ROT8(in) ^ key->r[4*l+1];
          c = ROT16(in) ^ key->r[4*l+2];
          d = ROT24(in) ^ key->r[4*l+3];

          for (j=0;j<2;j++)
          {
               p = a & 0x7fc;
               b += key->t[p/4];
               a = ROT9(a);

               p = b & 0x7fc;
               c += key->t[p/4];
               b = ROT9(b);

               p = c & 0x7fc;
               d += key->t[p/4];
               c = ROT9(c);

               p = d & 0x7fc;
               a += key->t[p/4];
               d = ROT9(d);

          }
          n1 = d;
          n2 = b;
          n3 = a;
          n4 = c;

          p = a & 0x7fc;
          b += key->t[p/4];
```

```
    a = ROT9(a);

    p = b & 0x7fc;
    c += key->t[p/4];
    b = ROT9(b);

    p = c & 0x7fc;
    d += key->t[p/4];
    c = ROT9(c);

    p = d & 0x7fc;
    a += key->t[p/4];
    d = ROT9(d);

  /* This generates 64 32-bit words, or 256 bytes of keystream. */
    for (i=0;i<64;i++)
    {
         p = a & 0x7fc;
         b += key->t[p/4];
         a = ROT9(a);
         b ^= a;

         q = b & 0x7fc;
         c ^= key->t[q/4];
         b = ROT9(b);
         c += b;

         p = (p+c) & 0x7fc;
         d += key->t[p/4];
         c = ROT9(c);
         d ^= c;

         q = (q+d) & 0x7fc;
         a ^= key->t[q/4];
         d = ROT9(d);
         a += d;

         p = (p+a) & 0x7fc;
         b ^= key->t[p/4];
         a = ROT9(a);

         q = (q+b) & 0x7fc;
         c += key->t[q/4];
         b = ROT9(b);

         p = (p+c) & 0x7fc;
         d ^= key->t[p/4];
         c = ROT9(c);

         q = (q+d) & 0x7fc;
         a += key->t[q/4];
         d = ROT9(d);

         *wp = b + key->s[4*i];
```

```
                wp++;
                *wp = c ^ key->s[4*i+1];
                wp++;
                *wp = d + key->s[4*i+2];
                wp++;
                *wp = a ^ key->s[4*i+3];
                wp++;

                if (i & 1)
                {
                     a += n3;
                     c += n4;
                }
                else
                {
                     a += n1;
                     c += n2;
                }

           }
     }
     return (ALG_OK);
}

/* Added call to refill ks_buf and reset counter and ks_pos. */
void seal_refill_buffer(seal_ctx *c){
        seal(c,c->counter,c->ks_buf);
        c->counter++;
        c->ks_pos = 0;
}

void seal_key(seal_ctx *c, unsigned char *key){
        seal_init(c,key);
        c->counter = 0;  /* By default, init to zero. */
        c->ks_pos = WORDS_PER_SEAL_CALL;
                /* Refill keystream buffer on next call. */
}

/* This encrypts the next w words with SEAL. */
void seal_encrypt(seal_ctx *c, unsigned long *data_ptr, int w){
        int i;

        for(i=0;i<w;i++){
                if(c->ks_pos>=WORDS_PER_SEAL_CALL) seal_refill_buffer(c);
                data_ptr[i]^=c->ks_buf[c->ks_pos];
                c->ks_pos++;
        }
}

void seal_decrypt(seal_ctx *c, unsigned long *data_ptr, int w) {
        seal_encrypt(c,data_ptr,w);
}

void seal_resynch(seal_ctx *c, unsigned long synch_word){
        c->counter = synch_word;
```

```
        c->ks_pos = WORDS_PER_SEAL_CALL;
}

void main(void){
        seal_ctx sc;
        unsigned long buf[1000],t;
        int i,flag;
        unsigned char key[] =
                {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};

        printf("1\n");
        seal_key(&sc,key);

        printf("2\n");
        for(i=0;i<1000;i++) buf[i]=0;
        printf("3\n");
        seal_encrypt(&sc,buf,1000);
        printf("4\n");
        t = 0;
        for(i=0;i<1000;i++) t = t ^ buf[i];
                printf("XOR of buf is %08lx.\n",t);

        seal_key(&sc,key);
        seal_decrypt(&sc,buf,1);
        seal_decrypt(&sc,buf+1,999);
        flag = 0;
        for(i=0;i<1000;i++) if(buf[i]!=0)flag=1;
        if(flag) printf("Decrypt failed.\n");
        else printf("Decrypt succeeded.\n");

}
```

•

References

1. ABA Bank Card Standard, "Management and Use of Personal Information Numbers," Aids from ABA, Catalog no. 207213, American Bankers Association, 1979.
2. ABA Document 4.3, "Key Management Standard," American Bankers Association, 1980.
3. M. Abadi, J. Feigenbaum, and J. Kilian, "On Hiding Information from an Oracle," *Proceedings of the 19th ACM Symposium on the Theory of Computing*, 1987, pp. 195–203.
4. M. Abadi, J. Feigenbaum, and J. Kilian, "On Hiding Information from an Oracle," *Journal of Computer and System Sciences*, v. 39, n. 1, Aug 1989, pp. 21–50.
5. M. Abadi and R. Needham, "Prudent Engineering Practice for Cryptographic Protocols," Research Report 125, Digital Equipment Corp Systems Research Center, Jun 1994.
6. C.M. Adams, "On Immunity Against Biham and Shamir's 'Differential Cryptanalysis,' " *Information Processing Letters*, v. 41, 14 Feb 1992, pp. 77–80.
7. C.M. Adams, "Simple and Effective Key Scheduling for Symmetric Ciphers," *Workshop on Selected Areas in Cryptography—Workshop Record*, Kingston, Ontario, 5–6 May 1994, pp. 129–133.
8. C.M. Adams and H. Meijer, "Security-Related Comments Regarding McEliece's Public-Key Cryptosystem," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 224–230.
9. C.M. Adams and S.E. Tavares, "The Structured Design of Cryptographically Good S-Boxes," *Journal of Cryptology*, v. 3, n. 1, 1990, pp. 27–41.
10. C.M. Adams and S.E. Tavares, "Designing S-Boxes for Ciphers Resistant to Differential Cryptanalysis," *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography*, Rome, Italy, 15–16 Feb 1993, pp. 181–190.
11. W. Adams and D. Shanks, "Strong Primality Tests That Are Not Sufficient," *Mathematics of Computation*, v. 39, 1982, pp. 255–300.
12. W.W. Adams and L.J. Goldstein, *Introduction to Number Theory*, Englewood Cliffs, N.J.: Prentice-Hall, 1976.
13. B.S. Adiga and P. Shankar, "Modified Lu-Lee Cryptosystem," *Electronics Letters*, v. 21, n. 18, 29 Aug 1985, pp. 794–795.
14. L.M. Adleman, "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography," *Proceedings of the IEEE 20th Annual Symposium of Foundations of Computer Science*, 1979, pp. 55–60.
15. L.M. Adleman, "On Breaking Generalized Knapsack Public Key Cryptosystems," *Proceedings of the 15th ACM Symposium on Theory of Computing*, 1983, pp. 402–412.

16. L.M. Adleman, "Factoring Numbers Using Singular Integers," *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing*, 1991, pp. 64–71.
17. L.M. Adleman, "Molecular Computation of Solutions to Combinatorial Problems," *Science*, v. 266, n. 11, Nov 1994, p. 1021.
18. L.M. Adleman, D. Estes, and K. McCurley, "Solving Bivariate Quadratic Congruences in Random Polynomial Time," *Mathematics of Computation*, v. 48, n. 177, Jan 1987, pp. 17–28.
19. L.M. Adleman, C. Pomerance, and R.S. Rumley, "On Distinguishing Prime Numbers from Composite Numbers," *Annals of Mathematics*, v. 117, n. 1, 1983, pp. 173–206.
20. L.M. Adleman and R.L. Rivest, "How to Break the Lu-Lee (COMSAT) Public-Key Cryptosystem," MIT Laboratory for Computer Science, Jul 1979.
21. G.B. Agnew, "Random Sources for Cryptographic Systems," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 77–81.
22. G.B. Agnew, R.C. Mullin, I.M. Onyszchuk, and S.A. Vanstone, "An Implementation for a Fast Public-Key Cryptosystem," *Journal of Cryptology*, v. 3, n. 2, 1991, pp. 63–79.
23. G.B. Agnew, R.C. Mullin, and S.A. Vanstone, "A Fast Elliptic Curve Cryptosystem," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 706–708.
24. G.B. Agnew, R.C. Mullin, and S.A. Vanstone, "Improved Digital Signature Scheme Based on Discrete Exponentiation," *Electronics Letters*, v. 26, n. 14, 5 Jul 1990, pp. 1024–1025.
25. G.B. Agnew, R.C. Mullin, and S.A. Vanstone, "On the Development of a Fast Elliptic Curve Cryptosystem," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 482–287.
26. G.B. Agnew, R.C. Mullin, and S.A. Vanstone, "An Implementation of Elliptic Curve Cryptosystems over F_2155," *IEEE Selected Areas of Communications*, v. 11, n. 5, Jun 1993, pp. 804–813.
27. A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
28. S.G. Akl, "Digital Signatures: A Tutorial Survey," *Computer*, v. 16, n. 2, Feb 1983, pp. 15–24.
29. S.G. Akl, "On the Security of Compressed Encodings," *Advances in Cryptology: Proceedings of Crypto 83*, Plenum Press, 1984, pp. 209–230.
30. S.G. Akl and H. Meijer, "A Fast Pseudo-Random Permutation Generator with Applications to Cryptology," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 269–275.
31. M. Alabbadi and S.B. Wicker, "Security of Xinmei Digital Signature Scheme," *Electronics Letters*, v. 28, n. 9, 23 Apr 1992, pp. 890–891.
32. M. Alabbadi and S.B. Wicker, "Digital Signature Schemes Based on Error-Correcting Codes," *Proceedings of the 1993 IEEE-ISIT*, IEEE Press, 1993, p. 199.
33. M. Alabbadi and S.B. Wicker, "Cryptanalysis of the Harn and Wang Modification of the Xinmei Digital Signature Scheme," *Electronics Letters*, v. 28, n. 18, 27 Aug 1992, pp. 1756–1758.
34. K. Alagappan and J. Tardo, "SPX Guide: Prototype Public Key Authentication Service," Digital Equipment Corp., May 1991.
35. W. Alexi, B.-Z. Chor, O. Goldreich, and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts Are as Hard as the Whole," *Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science*, 1984, pp. 449–457.
36. W. Alexi, B.-Z. Chor, O. Goldreich, and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts are as Hard as the Whole," *SIAM Journal on Computing*, v. 17, n. 2, Apr 1988, pp. 194–209.
37. Ameritech Mobile Communications et al., "Cellular Digital Packet Data System Specifications: Part 406: Airlink Security," CDPD Industry Input Coordinator, Costa Mesa, Calif., Jul 1993.
38. H.R. Amirazizi, E.D. Karnin, and J.M. Reyneri, "Compact Knapsacks are Polynomial Solvable," *ACM SIGACT News*, v. 15, 1983, pp. 20–22.
39. R.J. Anderson, "Solving a Class of Stream Ciphers," *Cryptologia*, v. 14, n. 3, Jul 1990, pp. 285–288.
40. R.J. Anderson, "A Second Generation Electronic Wallet," *ESORICS 92, Proceedings of the Second European Symposium on*

Research in Computer Security, Springer-Verlag, 1992, pp. 411–418.

41. R.J. Anderson, "Faster Attack on Certain Stream Ciphers," *Electronics Letters*, v. 29, n. 15, 22 Jul 1993, pp. 1322–1323.
42. R.J. Anderson, "Derived Sequence Attacks on Stream Ciphers," presented at the rump session of CRYPTO '93, Aug 1993.
43. R.J. Anderson, "Why Cryptosystems Fail," *1st ACM Conference on Computer and Communications Security*, ACM Press, 1993, pp. 215–227.
44. R.J. Anderson, "Why Cryptosystems Fail," *Communications of the ACM*, v. 37, n. 11, Nov 1994, pp. 32–40.
45. R.J. Anderson, "On Fibonacci Keystream Generators," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
46. R.J. Anderson, "Searching for the Optimum Correlation Attack," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
47. R.J. Anderson and T.M.A. Lomas, "Fortifying Key Negotiation Schemes with Poorly Chosen Passwords," *Electronics Letters*, v. 30, n. 13, 23 Jun 1994, pp. 1040–1041.
48. R.J. Anderson and R. Needham, "Robustness Principles for Public Key Protocols," *Advances in Cryptology—CRYPTO '95 Proceedings*, Springer-Verlag, 1995, to appear.
49. D. Andleman and J. Reeds, "On the Cryptanalysis of Rotor Machines and Substitution-Permutation Networks," *IEEE Transactions on Information Theory*, v. IT-28, n. 4, Jul 1982, pp. 578–584.
50. ANSI X3.92, "American National Standard for Data Encryption Algorithm (DEA)," American National Standards Institute, 1981.
51. ANSI X3.105, "American National Standard for Information Systems—Data Link Encryption," American National Standards Institute, 1983.
52. ANSI X3.106, "American National Standard for Information Systems—Data Encryption Algorithm—Modes of Operation," American National Standards Institute, 1983.
53. ANSI X9.8, "American National Standard for Personal Information Number (PIN) Management and Security," American Bankers Association, 1982.
54. ANSI X9.9 (Revised), "American National Standard for Financial Institution Message Authentication (Wholesale)," American Bankers Association, 1986.
55. ANSI X9.17 (Revised), "American National Standard for Financial Institution Key Management (Wholesale)," American Bankers Association, 1985.
56. ANSI X9.19, "American National Standard for Retail Message Authentication," American Bankers Association, 1985.
57. ANSI X9.23, "American National Standard for Financial Institution Message Encryption," American Bankers Association, 1988.
58. ANSI X9.24, "Draft Proposed American National Standard for Retail Key Management," American Bankers Association, 1988.
59. ANSI X9.26 (Revised), "American National Standard for Financial Institution Sign-On Authentication for Wholesale Financial Transaction," American Bankers Association, 1990.
60. ANSI X9.30, "Working Draft: Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry," American Bankers Association, Aug 1994.
61. ANSI X9.31, "Working Draft: Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry," American Bankers Association, Mar 1993.
62. K. Aoki and K. Ohta, "Differential-Linear Cryptanalysis of FEAL-8," *Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS 95)*, Inuyama, Japan, 24–27 Jan 1995, pp. A3.4.1-11. (In Japanese.)
63. K. Araki and T. Sekine, "On the Conspiracy Problem of the Generalized Tanaka's Cryptosystem," *IEICE Transactions*, v. E74, n. 8, Aug 1991, pp. 2176–2178.
64. S. Araki, K. Aoki, and K. Ohta, "The Best Linear Expression Search for FEAL," *Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS 95)*, Inuyama, Japan, 24–27 Jan 1995, pp. A4.4.1-10.
65. C. Asmuth and J. Bloom, "A Modular Approach to Key Safeguarding," *IEEE Transactions on Information Theory*, v. IT-29, n. 2, Mar 1983, pp. 208–210.

66. D. Atkins, M. Graff, A.K. Lenstra, and P.C. Leyland, "The Magic Words are Squeamish Ossifrage," *Advances in Cryptology—ASIACRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 263–277.
67. AT&T, "T7001 Random Number Generator," Data Sheet, Aug 1986.
68. AT&T, "AT&T Readying New Spy-Proof Phone for Big Military and Civilian Markets," *The Report on AT&T*, 2 Jun 1986, pp. 6–7.
69. AT&T, "T7002/T7003 Bit Slice Multiplier," product announcement, 1987.
70. AT&T, "Telephone Security Device TSD 3600—User's Manual," AT&T, 20 Sep 1992.
71. Y. Aumann and U. Feige, "On Message Proof Systems with Known Space Verifiers," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 85–99.
72. R.G. Ayoub, *An Introduction to the Theory of Numbers*, Providence, RI: American Mathematical Society, 1963.
73. A. Aziz and W. Diffie, "Privacy and Authentication for Wireless Local Area Networks," *IEEE Personal Communications*, v. 1, n. 1, 1994, pp. 25–31.
74. A. Bahreman and J.D. Tygar, "Certified Electronic Mail," *Proceedings of the Internet Society 1994 Workshop on Network and Distributed System Security*, The Internet Society, 1994, pp. 3–19.
75. D. Balenson, "Automated Distribution of Cryptographic Keys Using the Financial Institution Key Management Standard," *IEEE Communications Magazine*, v. 23, n. 9, Sep 1985, pp. 41–46.
76. D. Balenson, "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers," RFC 1423, Feb 1993.
77. D. Balenson, C.M. Ellison, S.B. Lipner, and S.T. Walker, "A New Approach to Software Key Escrow Encryption," TIS Report #520, Trusted Information Systems, Aug 94.
78. R. Ball, *Mathematical Recreations and Essays*, New York: MacMillan, 1960.
79. J Bamford, *The Puzzle Palace*, Boston: Houghton Mifflin, 1982.
80. J. Bamford and W. Madsen, *The Puzzle Palace*, Second Edition, Penguin Books, 1995.
81. S.K. Banerjee, "High Speed Implementation of DES," *Computers & Security*, v. 1, 1982, pp. 261–267.
82. Z. Baodong, "MC-Veiled Linear Transform Public Key Cryptosystem," *Acta Electronica Sinica*, v. 20, n. 4, Apr 1992, pp. 21–24. (In Chinese.)
83. P.H. Bardell, "Analysis of Cellular Automata Used as Pseudorandom Pattern Generators," *Proceedings of 1990 International Test Conference*, pp. 762–768.
84. T. Baritaud, H. Gilbert, and M. Girault, "FFT Hashing is not Collision-Free," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 35–44.
85. C. Barker, "An Industry Perspective of the CCEP," *2nd Annual AIAA Computer Security Conference Proceedings*, 1986.
86. W.G. Barker, *Cryptanalysis of the Hagelin Cryptograph*, Aegean Park Press, 1977.
87. P. Barrett, "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 311–323.
88. T.C. Bartee and D.I. Schneider, "Computation with Finite Fields," *Information and Control*, v. 6, n. 2, Jun 1963, pp. 79–98.
89. U. Baum and S. Blackburn, "Clock-Controlled Pseudorandom Generators on Finite Groups," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
90. K.R. Bauer, T.A. Bersen, and R.J. Feiertag, "A Key Distribution Protocol Using Event Markers," *ACM Transactions on Computer Systems*, v. 1, n. 3, 1983, pp. 249–255.
91. F. Bauspiess and F. Damm, "Requirements for Cryptographic Hash Functions," *Computers & Security*, v. 11, n. 5, Sep 1992, pp. 427–437.
92. D. Bayer, S. Haber, and W.S. Stornetta, "Improving the Efficiency and Reliability of Digital Time-Stamping," *Sequences '91: Methods in Communication, Security, and Computer Science*, Springer-Verlag, 1992, pp. 329–334.
93. R. Bayer and J.K. Metzger, "On the Encipherment of Search Trees and Random Access Files," *ACM Transactions on Database Systems*, v. 1, n. 1, Mar 1976, pp. 37–52.

94. M. Beale and M.F. Monaghan, "Encrytion Using Random Boolean Functions," *Cryptography and Coding*, H.J. Beker and F.C. Piper, eds., Oxford: Clarendon Press, 1989, pp. 219–230.
95. P. Beauchemin and G. Brassard, "A Generalization of Hellman's Extension to Shannon's Approach to Cryptography," *Journal of Cryptology*, v. 1, n. 2, 1988, pp. 129–132.
96. P. Beauchemin, G. Brassard, C. Crépeau, C. Goutier, and C. Pomerance, "The Generation of Random Numbers that are Probably Prime," *Journal of Cryptology*, v. 1, n. 1, 1988, pp. 53–64.
97. D. Beaver, J. Feigenbaum, and V. Shoup, "Hiding Instances in Zero-Knowledge Proofs," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 326–338.
98. H. Beker, J. Friend, and P. Halliden, "Simplifying Key Management in Electronic Funds Transfer Points of Sale Systems," *Electronics Letters*, v. 19, n. 12, Jun 1983, pp. 442–444.
99. H. Beker and F. Piper, *Cipher Systems: The Protection of Communications*, London: Northwood Books, 1982.
100. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Mathematical Foundations," Report ESD-TR-73-275, MITRE Corp., 1973.
101. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: A Mathematical Model," Report MTR-2547, MITRE Corp., 1973.
102. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: A Refinement of the Mathematical Model," Report ESD-TR-73-278, MITRE Corp., 1974.
103. D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," Report ESD-TR-75-306, MITRE Corp., 1976.
104. M. Bellare and S. Goldwasser, "New Paradigms for Digital Signatures and Message Authentication Based on Non-Interactive Zero Knowledge Proofs," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 194–211.
105. M. Bellare and S. Micali, "Non-Interactive Oblivious Transfer and Applications," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 547–557.
106. M. Bellare, S. Micali, and R. Ostrovsky, "Perfect Zero-Knowledge in Constant Rounds," *Proceedings of the 22th ACM Symposium on the Theory of Computing*, 1990, pp. 482–493.
107. S.M. Bellovin, "A Preliminary Technical Analysis of Clipper and Skipjack," unpublished manuscript, 20 Apr 1993.
108. S.M. Bellovin and M. Merritt, "Limitations of the Kerberos Protocol," *Winter 1991 USENIX Conference Proceedings*, USENIX Association, 1991, pp. 253–267.
109. S.M. Bellovin and M. Merritt, "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks," *Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy*, 1992, pp. 72–84.
110. S.M. Bellovin and M. Merritt, "An Attack on the Interlock Protocol When Used for Authentication," *IEEE Transactions on Information Theory*, v. 40, n. 1, Jan 1994, pp. 273–275.
111. S.M. Bellovin and M. Merritt, "Cryptographic Protocol for Secure Communications," U.S. Patent #5,241,599, 31 Aug 93.
112. I. Ben-Aroya and E. Biham, "Differential Cryptanalysis of Lucifer," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 187–199.
113. J.C. Benaloh, "Cryptographic Capsules: A Disjunctive Primitive for Interactive Protocols," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, 213–222.
114. J.C. Benaloh, "Secret Sharing Homorphisms: Keeping Shares of a Secret Secret," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 251–260.
115. J.C. Benaloh, "Verifiable Secret-Ballot Elections," Ph.D. dissertation, Yale University, YALEU/DCS/TR-561, Dec 1987.
116. J.C. Benaloh and M. de Mare, "One-Way Accumulators: A Decentralized Alternative to Digital Signatures," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 274–285.
117. J.C. Benaloh and D. Tuinstra, "Receipt-Free Secret Ballot Elections," *Proceedings of the 26th ACM Symposium on the Theory of Computing*, 1994, pp. 544–553.
118. J.C. Benaloh and M. Yung, "Distributing the Power of a Government to Enhance

the Privacy of Voters," *Proceedings of the 5th ACM Symposium on the Principles in Distributed Computing*, 1986, pp. 52–62.

119. A. Bender and G. Castagnoli, "On the Implementation of Elliptic Curve Cryptosystems," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 186–192.
120. S. Bengio, G. Brassard, Y.G. Desmedt, C. Goutier, and J.-J. Quisquater, "Secure Implementation of Identification Systems," *Journal of Cryptology*, v. 4, n. 3, 1991, pp. 175–184.
121. C.H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental Quantum Cryptography," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 253–265.
122. C.H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental Quantum Cryptography," *Journal of Cryptology*, v. 5, n. 1, 1992, pp. 3–28.
123. C.H. Bennett and G. Brassard, "Quantum Cryptography: Public Key Distribution and Coin Tossing," *Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing*, Banjalore, India, Dec 1984, pp. 175–179.
124. C.H. Bennett and G. Brassard, "An Update on Quantum Cryptography," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 475–480.
125. C.H. Bennett and G. Brassard, "Quantum Public-Key Distribution System," *IBM Technical Disclosure Bulletin*, v. 28, 1985, pp. 3153–3163.
126. C.H. Bennett and G. Brassard, "Quantum Public Key Distribution Reinvented," *SIGACT News*, v. 18, n. 4, 1987, pp. 51–53.
127. C.H. Bennett and G. Brassard, "The Dawn of a New Era for Quantum Cryptography: The Experimental Prototype is Working!" *SIGACT News*, v. 20, n. 4, Fall 1989, pp. 78–82.
128. C.H. Bennett, G. Brassard, and S. Breidbart, *Quantum Cryptography II: How to Re-Use a One-Time Pad Safely Even if* ***P=NP***, unpublished manuscript, Nov 1982.
129. C.H. Bennett, G. Brassard, S. Breidbart, and S. Weisner, "Quantum Cryptography, or Unforgeable Subway Tokens," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 267–275.
130. C.H. Bennett, G. Brassard, C. Crépeau, and M.-H. Skubiszewska, "Practical Quantum Oblivious Transfer," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 351–366.
131. C.H. Bennett, G. Brassard, and A.K. Ekert, "Quantum Cryptography," *Scientific American*, v. 267, n. 4, Oct 1992, pp. 50–57.
132. C.H. Bennett, G. Brassard, and N.D. Mermin, "Quantum Cryptography Without Bell's Theorem," *Physical Review Letters*, v. 68, n. 5, 3 Feb 1992, pp. 557–559.
133. C.H. Bennett, G. Brassard, and J.-M. Robert, "How to Reduce Your Enemy's Information," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 468–476.
134. C.H. Bennett, G. Brassard, and J.-M. Robert, "Privacy Amplification by Public Discussion," *SIAM Journal on Computing*, v. 17, n. 2, Apr 1988, pp. 210–229.
135. J. Bennett, "Analysis of the Encryption Algorithm Used in WordPerfect Word Processing Program," *Cryptologia*, v. 11, n. 4, Oct 1987, pp. 206–210.
136. M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation," *Proceedings of the 20th ACM Symposium on the Theory of Computing*, 1988, pp. 1–10.
137. M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, "Everything Provable is Provable in Zero-Knowledge," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 37–56.
138. M. Ben-Or, O. Goldreich, S. Micali, and R.L. Rivest, "A Fair Protocol for Signing Contracts," *IEEE Transactions on Information Theory*, v. 36, n. 1, Jan 1990, pp. 40–46.
139. H.A. Bergen and W.J. Caelli, "File Security in WordPerfect 5.0," *Cryptologia*, v. 15, n. 1, Jan 1991, pp. 57–66.
140. E.R. Berklecamp, *Algebraic Coding Theory*, Aegean Park Press, 1984.
141. S. Berkovitz, "How to Broadcast a Secret," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 535–541.
142. S. Berkovitz, J. Kowalchuk, and B. Schanning, "Implementing Public-Key Scheme," *IEEE Communications Magazine*, v. 17, n. 3, May 1979, pp. 2–3.

143. D.J. Bernstein, Bernstein vs. U.S. Department of State et al., Civil Action No. C95-0582-MHP, United States District Court for the Northern District of California, 21 Feb 1995.
144. T. Berson, "Differential Cryptanalysis Mod 2^{32} with Applications to MD5," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, 1992, pp. 71–80.
145. T. Beth, *Verfahren der schnellen Fourier-Transformation*, Teubner, Stuttgart, 1984. (In German.)
146. T. Beth, "Efficient Zero-Knowledge Identification Scheme for Smart Cards," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 77–84.
147. T. Beth, B.M. Cook, and D. Gollmann, "Architectures for Exponentiation in GF(2^n)," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 302–310.
148. T. Beth and Y. Desmedt, "Identification Tokens—or: Solving the Chess Grandmaster Problem," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 169–176.
149. T. Beth and C. Ding, "On Almost Nonlinear Permutations," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 65–76.
150. T. Beth, M. Frisch, and G.J. Simmons, eds., *Lecture Notes in Computer Science 578; Public Key Cryptography: State of the Art and Future Directions*, Springer-Verlag, 1992.
151. T. Beth and F.C. Piper, "The Stop-and-Go Generator," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1984, pp. 88–92.
152. T. Beth and F. Schaefer, "Non Supersingular Elliptic Curves for Public Key Cryptosystems," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 316–327.
153. A. Beutelspacher, "How to Say 'No'," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 491–496.
154. J. Bidzos, letter to NIST regarding DSS, 20 Sep 1991.
155. J. Bidzos, personal communication, 1993.
156. P. Bieber, "A Logic of Communication in a Hostile Environment," *Proceedings of the Computer Security Foundations Workshop III*, IEEE Computer Society Press, 1990, pp. 14–22.
157. E. Biham, "Cryptanalysis of the Chaotic-Map Cryptosystem Suggested at EUROCRYPT '91," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 532–534.
158. E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," Technical Report #753, Computer Science Department, Technion—Israel Institute of Technology, Sep 1992.
159. E. Biham, "On the Applicability of Differential Cryptanalysis to Hash Functions," lecture at EIES Workshop on Cryptographic Hash Functions, Mar 1992.
160. E. Biham, personal communication, 1993.
161. E. Biham, "Higher Order Differential Cryptanalysis," unpublished manuscript, Jan 1994.
162. E. Biham, "On Modes of Operation," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 116–120.
163. E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," *Journal of Cryptology*, v. 7, n. 4, 1994, pp. 229–246.
164. E. Biham, "On Matsui's Linear Cryptanalysis," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 398–412.
165. E. Biham and A. Biryukov, "How to Strengthen DES Using Existing Hardware," *Advances in Cryptology—ASIACRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
166. E. Biham and P.C. Kocher, "A Known Plaintext Attack on the PKZIP Encryption," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
167. E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 2–21.
168. E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology*, v. 4, n. 1, 1991, pp 3–72.
169. E. Biham and A. Shamir, "Differential Cryptanalysis of Feal and N-Hash," *Advances in Cryptology—EUROCRYPT*

'91 Proceedings, Springer-Verlag, 1991, pp. 1–16.

170. E. Biham and A. Shamir, "Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI, and Lucifer," *Advances in Cryptology—CRYPTO '91 Proceedings*, 1992, pp. 156–171.
171. E. Biham and A. Shamir, "Differential Cryptanalysis of the Full 16-Round DES," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, 487–496.
172. E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
173. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "Systematic Design of Two-Party Authentication Protocols," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 44–61.
174. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "Systematic Design of a Family of Attack-Resistant Authentication Protocols," *IEEE Journal of Selected Areas in Communication*, to appear.
175. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "A Modular Family of Secure Protocols for Authentication and Key Distribution," *IEEE/ACM Transactions on Networking*, to appear.
176. M. Bishop, "An Application for a Fast Data Encryption Standard Implementation," *Computing Systems*, v. 1, n. 3, 1988, pp. 221–254.
177. M. Bishop, "Privacy-Enhanced Electronic Mail," *Distributed Computing and Cryptography*, J. Feigenbaum and M. Merritt, eds., American Mathematical Society, 1991, pp. 93–106.
178. M. Bishop, "Privacy-Enhanced Electronic Mail," *Internetworking: Research and Experience*, v. 2, n. 4, Dec 1991, pp. 199–233.
179. M. Bishop, "Recent Changes to Privacy Enhanced Electronic Mail," *Internetworking: Research and Experience*, v. 4, n. 1, Mar 1993, pp. 47–59.
180. I.F. Blake, R. Fuji-Hara, R.C. Mullin, and S.A. Vanstone, "Computing Logarithms in Finite Fields of Characteristic Two," *SIAM Journal on Algebraic Discrete Methods*, v. 5, 1984, pp. 276–285.
181. I.F. Blake, R.C. Mullin, and S.A. Vanstone, "Computing Logarithms in GF (2^n)," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 73–82.
182. G.R. Blakley, "Safeguarding Cryptographic Keys," *Proceedings of the National Computer Conference, 1979*, American Federation of Information Processing Societies, v. 48, 1979, pp. 242–268.
183. G.R. Blakley, "One-Time Pads are Key Safeguarding Schemes, Not Cryptosystems—Fast Key Safeguarding Schemes (Threshold Schemes) Exist," *Proceedings of the 1980 Symposium on Security and Privacy*, IEEE Computer Society, Apr 1980, pp. 108–113.
184. G.R. Blakley and I. Borosh, "Rivest-Shamir-Adleman Public Key Cryptosystems Do Not Always Conceal Messages," *Computers and Mathematics with Applications*, v. 5, n. 3, 1979, pp. 169–178.
185. G.R. Blakley and C. Meadows, "A Database Encryption Scheme which Allows the Computation of Statistics Using Encrypted Data," *Proceedings of the 1985 Symposium on Security and Privacy*, IEEE Computer Society, Apr 1985, pp. 116–122.
186. M. Blaze, "A Cryptographic File System for UNIX," *1st ACM Conference on Computer and Communications Security*, ACM Press, 1993, pp. 9–16.
187. M. Blaze, "Protocol Failure in the Escrowed Encryption Standard," *2nd ACM Conference on Computer and Communications Security*, ACM Press, 1994, pp. 59–67.
188. M. Blaze, "Key Management in an Encrypting File System," *Proceedings of the Summer 94 USENIX Conference*, USENIX Association, 1994, pp. 27–35.
189. M. Blaze and B. Schneier, "The MacGuffin Block Cipher Algorithm," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
190. U. Blöcher and M. Dichtl, "Fish: A Fast Software Stream Cipher," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 41–44.
191. R. Blom, "Non-Public Key Distribution," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 231–236.

192. K.J. Blow and S.J.D. Phoenix, "On a Fundamental Theorem of Quantum Cryptography," *Journal of Modern Optics*, v. 40, n. 1, Jan 1993, pp. 33–36.
193. L. Blum, M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal on Computing*, v. 15, n. 2, 1986, pp. 364–383.
194. M. Blum, "Coin Flipping by Telephone: A Protocol for Solving Impossible Problems," *Proceedings of the 24th IEEE Computer Conference (CompCon)*, 1982, pp. 133–137.
195. M. Blum, "How to Exchange (Secret) Keys," *ACM Transactions on Computer Systems*, v. 1, n. 2, May 1983, pp. 175–193.
196. M. Blum, "How to Prove a Theorem So No One Else Can Claim It," *Proceedings of the International Congress of Mathematicians*, Berkeley, CA, 1986, pp. 1444–1451.
197. M. Blum, A. De Santis, S. Micali, and G. Persiano, "Noninteractive Zero-Knowledge," *SIAM Journal on Computing*, v. 20, n. 6, Dec 1991, pp. 1084–1118.
198. M. Blum, P. Feldman, and S. Micali, "Non-Interactive Zero-Knowledge and Its Applications," *Proceedings of the 20th ACM Symposium on Theory of Computing*, 1988, pp. 103–112.
199. M. Blum and S. Goldwasser, "An *Efficient* Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 289–299.
200. M. Blum and S. Micali, "How to Generate Cryptographically-Strong Sequences of Pseudo-Random Bits," *SIAM Journal on Computing*, v. 13, n. 4, Nov 1984, pp. 850–864.
201. B. den Boer, "Cryptanalysis of F.E.A.L.," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 293–300.
202. B. den Boer and A. Bosselaers, "An Attack on the Last Two Rounds of MD4," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 194–203.
203. B. den Boer and A. Bosselaers, "Collisions for the Compression Function of MD5," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 293–304.
204. J.-P. Boly, A. Bosselaers, R. Cramer, R. Michelsen, S. Mjølsnes, F. Muller, T. Pedersen, B. Pfitzmann, P. de Rooij, B. Schoenmakers, M. Schunter, L. Vallée, and M. Waidner, "Digital Payment Systems in the ESPRIT Project CAFE," *Securicom 94*, Paris, France, 2–6 Jan 1994, pp. 35–45.
205. J.-P. Boly, A. Bosselaers, R. Cramer, R. Michelsen, S. Mjølsnes, F. Muller, T. Pedersen, B. Pfitzmann, P. de Rooij, B. Schoenmakers, M. Schunter, L. Vallée, and M. Waidner, "The ESPRIT Project CAFE—High Security Digital Payment System," *Computer Security—ESORICS 94*, Springer-Verlag, 1994, pp. 217–230.
206. D.J. Bond, "Practical Primality Testing," *Proceedings of IEE International Conference on Secure Communications Systems*, 22–23 Feb 1984, pp. 50–53.
207. H. Bonnenberg, *Secure Testing of VSLI Cryptographic Equipment*, Series in Microelectronics, Vol. 25, Konstanz: Hartung Gorre Verlag, 1993.
208. H. Bonnenberg, A. Curiger, N. Felber, H. Kaeslin, and X. Lai, "VLSI Implementation of a New Block Cipher," *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD 91)*, Oct 1991, pp. 510–513.
209. K.S. Booth, "Authentication of Signatures Using Public Key Encryption," *Communications of the ACM*, v. 24, n. 11, Nov 1981, pp. 772–774.
210. A. Bosselaers, R. Govaerts, and J. Vanderwalle, *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 175–186.
211. D.P. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*, Englewood Cliffs, N.J.: Prentice-Hall, 1994.
212. J. Boyar, "Inferring Sequences Produced by a Linear Congruential Generator Missing Low-Order Bits," *Journal of Cryptology*, v. 1, n. 3, 1989, pp. 177–184.
213. J. Boyar, D. Chaum, and I. Damgård, "Convertible Undeniable Signatures," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 189–205.
214. J. Boyar, K. Friedl, and C. Lund, "Practical Zero-Knowledge Proofs: Giving Hints and Using Deficiencies," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 155–172.

215. J. Boyar, C. Lund, and R. Peralta, "On the Communication Complexity of Zero-Knowledge Proofs," *Journal of Cryptology*, v. 6, n. 2, 1993, pp. 65–85.
216. J. Boyar and R. Peralta, "On the Concrete Complexity of Zero-Knowledge Proofs," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 507–525.
217. C. Boyd, "Some Applications of Multiple Key Ciphers," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 455–467.
218. C. Boyd, "Digital Multisignatures," *Cryptography and Coding*, H.J. Beker and F.C. Piper, eds., Oxford: Clarendon Press, 1989, pp. 241–246.
219. C. Boyd, "A New Multiple Key Cipher and an Improved Voting Scheme," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 617–625.
220. C. Boyd, "Multisignatures Revisited," *Cryptography and Coding III*, M.J. Ganley, ed., Oxford: Clarendon Press, 1993, pp. 21–30.
221. C. Boyd and W. Mao, "On the Limitation of BAN Logic," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 240–247.
222. C. Boyd and W. Mao, "Designing Secure Key Exchange Protocols," *Computer Security—ESORICS 94*, Springer-Verlag, 1994, pp. 217–230.
223. B.O. Brachtl, D. Coppersmith, M.M. Hyden, S.M. Matyas, C.H. Meyer, J. Oseas, S. Pilpel, and M. Schilling, "Data Authentication Using Modification Detection Codes Based on a Public One Way Function," U.S. Patent #4,908,861, 13 Mar 1990.
224. J. Brandt, I.B. Damgård, P. Landrock, and T. Pederson, "Zero-Knowledge Authentication Scheme with Secret Key Exchange," *Advances in Cryptology—CRYPTO '88*, Springer-Verlag, 1990, pp. 583–588.
225. S.A. Brands, "An Efficient Off-Line Electronic Cash System Based on the Representation Problem," Report CS-R9323, Computer Science/Department of Algorithms and Architecture, CWI, Mar 1993.
226. S.A. Brands, "Untraceable Off-line Cash in Wallet with Observers," *Advances in Cryptology—CRYPTO '93*, Springer-Verlag, 1994, pp. 302–318.
227. S.A. Brands, "Electronic Cash on the Internet," *Proceedings of the Internet Society 1995 Symposium on Network and Distributed Systems Security*, IEEE Computer Society Press 1995, pp 64–84.
228. D.K. Branstad, "Hellman's Data Does Not Support His Conclusion," *IEEE Spectrum*, v. 16, n. 7, Jul 1979, p. 39.
229. D.K. Branstad, J. Gait, and S. Katzke, "Report on the Workshop on Cryptography in Support of Computer Security," NBSIR 77-1291, National Bureau of Standards, Sep 21–22, 1976, September 1977.
230. G. Brassard, "A Note on the Complexity of Cryptography," *IEEE Transactions on Information Theory*, v. IT-25, n. 2, Mar 1979, pp. 232–233.
231. G. Brassard, "Relativized Cryptography," *Proceedings of the IEEE 20th Annual Symposium on the Foundations of Computer Science*, 1979, pp. 383–391.
232. G. Brassard, "A Time-Luck Tradeoff in Relativized Cryptography," *Proceedings of the IEEE 21st Annual Symposium on the Foundations of Computer Science*, 1980, pp. 380–386.
233. G. Brassard, "A Time-Luck Tradeoff in Relativized Cryptography," *Journal of Computer and System Sciences*, v. 22, n. 3, Jun 1981, pp. 280–311.
234. G. Brassard, "An Optimally Secure Relativized Cryptosystem," *SIGACT News*, v. 15, n. 1, 1983, pp. 28–33.
235. G. Brassard, "Relativized Cryptography," *IEEE Transactions on Information Theory*, v. IT-29, n. 6, Nov 1983, pp. 877–894.
236. G. Brassard, *Modern Cryptology: A Tutorial*, Springer-Verlag, 1988.
237. G. Brassard, "Quantum Cryptography: A Bibliography," *SIGACT News*, v. 24, n. 3, Oct 1993, pp. 16–20.
238. G. Brassard, D. Chaum, and C. Crépeau, "An Introduction to Minimum Disclosure," *CWI Quarterly*, v. 1, 1988, pp. 3–17.
239. G. Brassard, D. Chaum, and C. Crépeau, "Minimum Disclosure Proofs of Knowledge," *Journal of Computer and System Sciences*, v. 37, n. 2, Oct 1988, pp. 156–189.
240. G. Brassard and C. Crépeau, "Non-Transitive Transfer of Confidence: A *Perfect* Zero-Knowledge Interactive Protocol for SAT and Beyond," *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986, pp. 188–195.

241. G. Brassard and C. Crépeau, "Zero-Knowledge Simulation of Boolean Circuits," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 223–233.
242. G. Brassard and C. Crépeau, "Sorting Out Zero-Knowledge," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 181–191.
243. G. Brassard and C. Crépeau, "Quantum Bit Commitment and Coin Tossing Protocols," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 49–61.
244. G. Brassard, C. Crépeau, R. Jozsa, and D. Langlois, "A Quantum Bit Commitment Scheme Provably Unbreakable by Both Parties," *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, 1993, pp. 362–371.
245. G. Brassard, C. Crépeau, and J.-M. Robert, "Information Theoretic Reductions Among Disclosure Problems," *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986, pp. 168–173.
246. G. Brassard, C. Crépeau, and J.-M. Robert, "All-or-Nothing Disclosure of Secrets," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 234–238.
247. G. Brassard, C. Crépeau, and M. Yung, "Everything in **NP** Can Be Argued in Perfect Zero-Knowledge in a Bounded Number of Rounds," *Proceedings on the 16th International Colloquium on Automata, Languages, and Programming*, Springer-Verlag, 1989, pp. 123–136.
248. R.P. Brent, "An Improved Monte-Carlo Factorization Algorithm," *BIT*, v. 20, n. 2, 1980, pp. 176–184.
249. R.P. Brent, "On the Periods of Generalized Fibonacci Recurrences, *Mathematics of Computation*, v. 63, n. 207, Jul 1994, pp. 389–401.
250. R.P. Brent, "Parallel Algorithms for Integer Factorization," Research Report CMA-R49-89, Computer Science Laboratory, The Australian National University, Oct 1989.
251. D.M. Bressoud, *Factorization and Primality Testing*, Springer-Verlag, 1989.
252. E.F. Brickell, "A Fast Modular Multiplication Algorithm with Applications to Two Key Cryptography," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1982, pp. 51–60.
253. E.F. Brickell, "Are Most Low Density Polynomial Knapsacks Solvable in Polynomial Time?" *Proceedings of the 14th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, 1983.
254. E.F. Brickell, "Solving Low Density Knapsacks," *Advances in Cryptology: Proceedings of Crypto 83*, Plenum Press, 1984, pp. 25–37.
255. E.F. Brickell, "Breaking Iterated Knapsacks," *Advances in Cryptology: Proceedings of Crypto 84*, Springer-Verlag, 1985, pp. 342–358.
256. E.F. Brickell, "Cryptanalysis of the Uagisawa Public Key Cryptosystem," *Abstracts of Papers, EUROCRYPT '86*, 20–22 May 1986.
257. E.F. Brickell, "The Cryptanalysis of Knapsack Cryptosystems," *Applications of Discrete Mathematics*, R.D. Ringeisen and F.S. Roberts, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1988, pp. 3–23.
258. E.F. Brickell, "Survey of Hardware Implementations of RSA," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 368–370.
259. E.F. Brickell, D. Chaum, I.B. Damgård, and J. van de Graff, "Gradual and Verifiable Release of a Secret," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 156–166.
260. E.F. Brickell, J.A. Davis, and G.J. Simmons, "A Preliminary Report on the Cryptanalysis of Merkle-Hellman Knapsack," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 289–303.
261. E.F. Brickell and J. DeLaurentis, "An Attack on a Signature Scheme Proposed by Okamoto and Shiraishi," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 28–32.
262. E.F. Brickell, D.E. Denning, S.T. Kent, D.P. Maher, and W. Tuchman, "SKIPJACK Review—Interim Report," unpublished manuscript, 28 Jul 1993.
263. E.F. Brickell, J.C. Lagarias, and A.M. Odlyzko, "Evaluation of the Adleman Attack of Multiple Iterated Knapsack Cryptosystems," *Advances in Cryptology:*

Proceedings of Crypto 83, Plenum Press, 1984, pp. 39–42.

264. E.F. Brickell, P.J. Lee, and Y. Yacobi, "Secure Audio Teleconference," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 418–426.
265. E.F. Brickell and K.S. McCurley, "An Interactive Identification Scheme Based on Discrete Logarithms and Factoring," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 63–71.
266. E.F. Brickell, J.H. Moore, and M.R. Purtill, "Structure in the S-Boxes of the DES," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 3–8.
267. E.F. Brickell and A.M. Odlyzko, "Cryptanalysis: A Survey of Recent Results," *Proceedings of the IEEE*, v. 76, n. 5, May 1988, pp. 578–593.
268. E.F. Brickell and A.M. Odlyzko, "Cryptanalysis: A Survey of Recent Results," *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1991, pp. 501–540.
269. E.F. Brickell and G.J. Simmons, "A Status Report on Knapsack Based Public Key Cryptosystems," *Congressus Numerantium*, v. 7, 1983, pp. 3–72.
270. E.F. Brickell and D.R. Stinson, "The Detection of Cheaters in Threshold Schemes," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 564–577.
271. A.G. Broscius and J.M. Smith, "Exploiting Parallelism in Hardware Implementation of the DES," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 367–376.
272. L. Brown, M. Kwan, J. Pieprzyk, and J. Seberry, "Improving Resistance to Differential Cryptanalysis and the Redesign of LOKI," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1993, pp. 36–50.
273. L. Brown, J. Pieprzyk, and J. Seberry, "LOKI: A Cryptographic Primitive for Authentication and Secrecy Applications," *Advances in Cryptology—AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 229–236.
274. L. Brown, J. Pieprzyk, and J. Seberry, "Key Scheduling in DES Type Cryptosystems," *Advances in Cryptology—AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 221–228.
275. L. Brown and J. Seberry, "On the Design of Permutation P in DES Type Cryptosystems," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 696–705.
276. W. Brown, "A Quantum Leap in Secret Communications," *New Scientist*, n. 1585, 30 Jan 1993, p. 21.
277. J.O. Brüer, "On Pseudo Random Sequences as Crypto Generators," *Proceedings of the International Zurich Seminar on Digital Communication*, Switzerland, 1984.
278. L. Brynielsson "On the Linear Complexity of Combined Shift Register Sequences," *Advances in Cryptology—EUROCRYPT '85*, Springer-Verlag, 1986, pp. 156–166.
279. J. Buchmann, J. Loho, and J. Zayer, "An Implementation of the General Number Field Sieve," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 159–165.
280. M. Burmester and Y. Desmedt, "Broadcast Interactive Proofs," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 81–95.
281. M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
282. D. Burnham, "NSA Seeking 500,000 'Secure' Telephones," *The New York Times*, 6 Oct 1994.
283. M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," Research Report 39, Digital Equipment Corp. Systems Research Center, Feb 1989.
284. M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," *ACM Transactions on Computer Systems*, v. 8, n. 1, Feb 1990, pp. 18–36.
285. M. Burrows, M. Abadi, and R. Needham, "Rejoinder to Nessett," *Operating System Review*, v. 20, n. 2, Apr 1990, pp. 39–40.
286. J.J. Cade, "A Modification of a Broken Public-Key Cipher," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 64–83.
287. T.R. Cain and A.T. Sherman, "How to Break Gifford's Cipher," *Proceedings of the 2nd Annual ACM Conference on*

Computer and Communications Security, ACM Press, 1994, pp. 198–209.

288. C. Calvelli and V. Varadharajan, "An Analysis of Some Delegation Protocols for Distributed Systems," *Proceedings of the Computer Security Foundations Workshop V*, IEEE Computer Society Press, 1992, pp. 92–110.
289. J.L. Camenisch, J.-M. Piveteau, and M.A. Stadler, "An Efficient Electronic Payment System Protecting Privacy," *Computer Security—ESORICS 94*, Springer-Verlag, 1994, pp. 207–215.
290. P. Camion and J. Patarin, "The Knapsack Hash Function Proposed at Crypto '89 Can Be Broken," *Advances in Cryptology—EUROCRYPT '91*, Springer-Verlag, 1991, pp. 39–53.
291. C.M. Campbell, "Design and Specification of Cryptographic Capabilities," *IEEE Computer Society Magazine*, v. 16, n. 6, Nov 1978, pp. 15–19.
292. E.A. Campbell, R. Safavi-Naini, and P.A. Pleasants, "Partial Belief and Probabilistic Reasoning in the Analysis of Secure Protocols," *Proceedings of the Computer Security Foundations Workshop V*, IEEE Computer Society Press, 1992, pp. 92–110.
293. K.W. Campbell and M.J. Wiener, "DES Is Not a Group," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, pp. 512–520.
294. Z.F. Cao and G. Zhao, "Some New MC Knapsack Cryptosystems," *CHINACRYPT '94*, Xidian, China, 11–15 Nov 1994, pp. 70–75. (In Chinese).
295. C. Carlet, "Partially-Bent Functions," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 280–291.
296. C. Carlet, "Partially Bent Functions," *Designs, Codes and Cryptography*, v. 3, 1993, pp. 135–145.
297. C. Carlet, "Two New Classes of Bent Functions" *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 77–101.
298. C. Carlet, J. Seberry, and X.M. Zhang, "Comments on 'Generating and Counting Binary Bent Sequences,' " *IEEE Transactions on Information Theory*, v. IT-40, n. 2, Mar 1994, p. 600.
299. J.M. Carroll, *Computer Security*, 2nd edition, Butterworths, 1987.
300. J.M. Carroll, "The Three Faces of Information Security," *Advances in Cryptology—AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 433–450.
301. J.M. Carroll, " 'Do-it-yourself' Cryptography," *Computers & Security*, v. 9, n. 7, Nov 1990, pp. 613–619.
302. T.R. Caron and R.D. Silverman, "Parallel Implementation of the Quadratic Scheme," *Journal of Supercomputing*, v. 1, n. 3, 1988, pp. 273–290.
303. CCITT, Draft Recommendation X.509, "The Directory—Authentication Framework," Consultation Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva, 1987.
304. CCITT, Recommendation X.509, "The Directory—Authentication Framework," Consultation Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva, 1989.
305. CCITT, Recommendation X.800, "Security Architecture for Open Systems Interconnection for CCITT Applications," International Telephone and Telegraph, International Telecommunications Union, Geneva, 1991.
306. F. Chabaud, "On the Security of Some Cryptosystems Based on Error-Correcting Codes," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
307. F. Chabaud and S. Vaudenay, "Links Between Differential and Linear Cryptanalysis," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
308. W.G. Chambers and D. Gollmann, "Generators for Sequences with Near-Maximal Linear Equivalence," *IEE Proceedings*, V. 135, Pt. E, n. 1, Jan 1988, pp. 67–69.
309. W.G. Chambers and D. Gollmann, "Lock-In Effect in Cascades of Clock-Controlled Shirt Registers," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 331–343.
310. A. Chan and R. Games, "On the Linear Span of Binary Sequences from Finite Geometries," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 405–417.
311. J.P. Chandler, D.C. Arrington, D.R. Berkelhammer, and W.L. Gill, "Identification and

Analysis of Foreign Laws and Regulations Pertaining to the Use of Commercial Encryption Products for Voice and Data Communications," National Intellectual Property Law Institute, George Washington University, Washington, D.C., Jan 1994.

312. C.C. Chang and S.J. Hwang, "Cryptographic Authentication of Passwords," *Proceedings of the 25th Annual 1991 IEEE International Carnahan Conference on Security Technology*, Taipei, Taiwan, 1–3 Oct 1991, pp. 126–130.
313. C.C. Chang and S.J. Hwang, "A Strategy for Transforming Public-Key Cryptosystems into Identity-Based Cryptosystems," *Proceedings of the 25th Annual 1991 IEEE International Carnahan Conference on Security Technology*, Taipei, Taiwan, 1–3 Oct 1991, pp. 68–72.
314. C.C. Chang and C.H. Lin, "An ID-Based Signature Scheme Based upon Rabin's Public Key Cryptosystem," *Proceedings of the 25th Annual 1991 IEEE International Carnahan Conference on Security Technology*, Taipei, Taiwan, 1–3 Oct 1991, pp. 139–141.
315. C. Charnes and J. Pieprzyk, "Attacking the SL_2 Hashing Scheme," *Advances in Cryptology—ASIACRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 322–330.
316. D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, v. 24, n. 2, Feb 1981, pp. 84–88.
317. D. Chaum, "Blind Signatures for Untraceable Payments," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 199–203.
318. D. Chaum, "Security Without Identification: Transaction Systems to Make Big Brother Obsolete," *Communications of the ACM*, v. 28, n. 10, Oct 1985, pp. 1030–1044.
319. D. Chaum, "Demonstrating that a Public Predicate Can Be Satisfied without Revealing Any Information about How," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 159–199.
320. D. Chaum, "Blinding for Unanticipated Signatures," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 227–233.
321. D. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Receiver Untraceability," *Journal of Cryptology*, v. 1, n. 1, 1988, pp. 65–75.
322. D. Chaum, "Elections with Unconditionally Secret Ballots and Disruptions Equivalent to Breaking RSA," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 177–181.
323. D. Chaum, "Blind Signature Systems," U.S. Patent #4,759,063, 19 Jul 1988.
324. D. Chaum, "Blind Unanticipated Signature Systems," U.S. Patent #4,759,064, 19 Jul 1988.
325. D. Chaum, "Online Cash Checks," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 288–293.
326. D. Chaum, "One-Show Blind Signature Systems," U.S. Patent #4,914,698, 3 Apr 1990.
327. D. Chaum, "Undeniable Signature Systems," U.S. Patent #4,947,430, 7 Aug 1990.
328. D. Chaum, "Returned-Value Blind Signature Systems," U.S. Patent #4,949,380, 14 Aug 1990.
329. D. Chaum, "Zero-Knowledge Undeniable Signatures," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 458–464.
330. D. Chaum, "Group Signatures," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 257–265.
331. D. Chaum, "Unpredictable Blind Signature Systems," U.S. Patent #4,991,210, 5 Feb 1991.
332. D. Chaum, "Achieving Electronic Privacy," *Scientific American*, v. 267, n. 2, Aug 1992, pp. 96–101.
333. D. Chaum, "Designated Confirmer Signatures," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
334. D. Chaum, C. Crépeau, and I.B. Damgård, "Multiparty Unconditionally Secure Protocols," Proceedings of the 20th ACM Symposium on the Theory of Computing, 1988, pp. 11–19.
335. D. Chaum, B. den Boer, E. van Heyst, S. Mjølsnes, and A. Steenbeek, "Efficient Offline Electronic Checks," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 294–301.

336. D. Chaum and J.-H. Evertse, "Cryptanalysis of DES with a Reduced Number of Rounds; Sequences of Linear Factors in Block Ciphers," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 192–211.

337. D. Chaum, J.-H. Evertse, and J. van de Graff, "An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 127–141.

338. D. Chaum, J.-H. Evertse, J. van de Graff, and R. Peralta, "Demonstrating Possession of a Discrete Logarithm without Revealing It," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 200–212.

339. D. Chaum, A. Fiat, and M. Naor, "Untraceable Electronic Cash," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 319–327.

340. D. Chaum and T. Pedersen, "Transferred Cash Grows in Size," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 391–407.

341. D. Chaum and T. Pedersen, "Wallet Databases with Observers," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 89–105.

342. D. Chaum and I. Schaumuller-Bichel, eds., *Smart Card 2000*, North Holland: Elsevier Science Publishers, 1989.

343. D. Chaum and H. van Antwerpen, "Undeniable Signatures," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 212–216.

344. D. Chaum, E. van Heijst, and B. Pfitzmann, "Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 470–484.

345. T.M. Chee, "The Cryptanalysis of a New Public-Key Cryptosystem Based on Modular Knapsacks," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 204–212.

346. L. Chen, "Oblivious Signatures," *Computer Security—ESORICS 94*, Springer-Verlag, 1994, pp. 161–172.

347. L. Chen and M. Burminster, "A Practical Secret Voting Scheme which Allows Voters to Abstain," *CHINACRYPT '94*, Xidian, China, 11–15 Nov 1994, pp. 100–107.

348. L. Chen and T.P. Pedersen "New Group Signature Schemes," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.

349. J. Chenhui, "Spectral Characteristics of Partially-Bent Functions," *CHINACRYPT '94*, Xidian, China, 11–15 Nov 1994, pp. 48–51.

350. V. Chepyzhov and B. Smeets, "On a Fast Correlation Attack on Certain Stream Ciphers," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 176–185.

351. T.C. Cheung, "Management of PEM Public Key Certificates Using X.500 Directory Service: Some Problems and Solutions," *Proceedings of the Internet Society 1994 Workshop on Network and Distributed System Security*, The Internet Society, 1994, pp. 35–42.

352. G.C. Chiou and W.C. Chen, "Secure Broadcasting Using the Secure Lock," *IEEE Transactions on Software Engineering*, v. SE-15, n. 8, Aug 1989, pp. 929–934.

353. Y.J. Choie and H.S. Hwoang, "On the Cryptosystem Using Elliptic Curves," *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 105–113.

354. B. Chor and O. Goldreich, "RSA/Rabin Least Significant Bits are 1/2+1/poly(log N) Secure," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 303–313.

355. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults," *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, 1985, pp. 383–395.

356. B. Chor and R.L. Rivest, "A Knapsack Type Public Key Cryptosystem Based on Arithmetic in Finite Fields," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 54–65.

357. P. Christoffersson, S.-A. Ekahll, V. Fåk, S. Herda, P. Mattila, W. Price, and H.-O. Widman, *Crypto Users' Handbook: A Guide for Implementors of Cryptographic Protection in Computer Systems*, North Holland: Elsevier Science Publishers, 1988.

358. R. Cleve, "Controlled Gradual Disclosure Schemes for Random Bits and Their Applications," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 572–588.

359. J.D. Cohen, "Improving Privacy in Cryptographic Elections," Yale University Computer Science Department Technical Report YALEU/DCS/TR-454, Feb 1986.

360. J.D. Cohen and M.H. Fischer, "A Robust and Verifiable Cryptographically Secure Election Scheme," *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, 1985, pp. 372–382.

361. R. Cole, "A Model for Security in Distributed Systems," *Computers and Security*, v. 9, n. 4, Apr 1990, pp. 319–330.

362. Comptroller General of the United States, "Matter of National Institute of Standards and Technology—Use of Electronic Data Interchange Technology to Create Valid Obligations," File B-245714, 13 Dec 1991.

363. M.S. Conn, letter to Joe Abernathy, National Security Agency, Ser: Q43-111-92, 10 Jun 1992.

364. C. Connell, "An Analysis of NewDES: A Modified Version of DES," *Cryptologia*, v. 14, n. 3, Jul 1990, pp. 217–223.

365. S.A. Cook, "The Complexity of Theorem-Proving Procedures," *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, 1971, pp. 151–158.

366. R.H. Cooper and W. Patterson, "A Generalization of the Knapsack Method Using Galois Fields," *Cryptologia*, v. 8, n. 4, Oct 1984, pp. 343–347.

367. R.H. Cooper and W. Patterson, "RSA as a Benchmark for Multiprocessor Machines," *Advances in Cryptology—AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 356–359.

368. D. Coppersmith, "Fast Evaluation of Logarithms in Fields of Characteristic Two," *IEEE Transactions on Information Theory*, v. 30, n. 4, Jul 1984, pp. 587–594.

369. D. Coppersmith, "Another Birthday Attack," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 14–17.

370. D. Coppersmith, "Cheating at Mental Poker," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 104–107.

371. D. Coppersmith, "The Real Reason for Rivest's Phenomenon," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 535–536.

372. D. Coppersmith, "Two Broken Hash Functions," Research Report RD 18397, IBM T.J. Watson Center, Oct 1992.

373. D. Coppersmith, "The Data Encryption Standard (DES) and Its Strength against Attacks," Technical Report RC 18613, IBM T.J. Watson Center, Dec 1992.

374. D. Coppersmith, "The Data Encryption Standard (DES) and its Strength against Attacks," *IBM Journal of Research and Development*, v. 38, n. 3, May 1994, pp. 243–250.

375. D. Coppersmith, "Attack on the Cryptographic Scheme NIKS-TAS," *Advances in Cryptology—CRYPTO '94 Proceedings*, Springer-Verlag, 1994, pp. 294–307.

376. D. Coppersmith, personal communication, 1994.

377. D. Coppersmith and E. Grossman, "Generators for Certain Alternating Groups with Applications to Cryptography," *SIAM Journal on Applied Mathematics*, v. 29, n. 4, Dec 1975, pp. 624–627.

378. D. Coppersmith, H. Krawczyk, and Y. Mansour, "The Shrinking Generator," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 22–39.

379. D. Coppersmith, A. Odlykzo, and R. Schroeppel, "Discrete Logarithms in GF[p]," *Algorithmica*, v. 1, n. 1, 1986, pp. 1–16.

380. D. Coppersmith and P. Rogaway, "Software Efficient Pseudo Random Function and the Use Thereof for Encryption," U.S. Patent pending, 1995.

381. D. Coppersmith, J. Stern, and S. Vaudenay, "Attacks on the Birational Signature Schemes," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 435–443.

382. V. Cordonnier and J.-J. Quisquater, eds., *CARDIS '94—Proceedings of the First Smart Card Research and Advanced Application Conference*, Lille, France, 24–26 Oct 1994.

383. C. Couvreur and J.-J. Quisquater, "An Introduction to Fast Generation of Large Prime Numbers," *Philips Journal Research*, v. 37, n. 5–6, 1982, pp. 231–264.

384. C. Couvreur and J.-J. Quisquater, "An Introduction to Fast Generation of Large Prime Numbers," *Philips Journal Research*, v. 38, 1983, p. 77.

385. C. Coveyou and R.D. MacPherson, "Fourier Analysis of Uniform Random Number Generators," *Journal of the ACM*, v. 14, n. 1, 1967, pp. 100–119.

386. T.M. Cover and R.C. King, "A Convergent Gambling Estimate of the Entropy of English," *IEEE Transactions on Information Theory*, v. IT-24, n. 4, Jul 1978, pp. 413–421.

387. R.J.F. Cramer and T.P. Pedersen, "Improved Privacy in Wallets with Observers," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 329–343.

388. R.E. Crandell, "Method and Apparatus for Public Key Exchange in a Cryptographic System," U.S. Patent #5,159,632, 27 Oct 1992.

389. C. Crépeau, "A Secure Poker Protocol That Minimizes the Effect of Player Coalitions," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 73–86.

390. C. Crépeau, "A Zero-Knowledge Poker Protocol that Achieves Confidentiality of the Players' Strategy, or How to Achieve an Electronic Poker Face," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 239–247.

391. C. Crépeau, "Equivalence Between Two Flavours of Oblivious Transfer," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 350–354.

392. C. Crépeau, "Correct and Private Reductions among Oblivious Transfers," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1990.

393. C. Crépeau, "Quantum Oblivious Transfer," *Journal of Modern Optics*, v. 41, n. 12, Dec 1994, pp. 2445–2454.

394. C. Crépeau and J. Kilian, "Achieving Oblivious Transfer Using Weakened Security Assumptions," *Proceedings of the 29th Annual Symposium on the Foundations of Computer Science*, 1988, pp. 42–52.

395. C. Crépeau and J. Kilian, "Weakening Security Assumptions and Oblivious Transfer," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 2–7.

396. C. Crépeau and L. Salvail, "Quantum Oblivious Mutual Identification," *Advances in Cryptology—EUROCRYPT '95 Proceedings*, Springer-Verlag, 1995, pp. 133–146.

397. A. Curiger, H. Bonnenberg, R. Zimmermann, N. Felber, H. Kaeslin and W. Fichtner, "VINCI: VLSI Implementation of the New Block Cipher IDEA," *Proceedings of IEEE CICC '93*, San Diego, CA, May 1993, pp. 15.5.1–15.5.4.

398. A. Curiger and B. Stuber, "Specification for the IDEA Chip," Technical Report No. 92/03, Institut für Integrierte Systeme, ETH Zurich, Feb 1992.

399. T. Cusick, "Boolean Functions Satisfying a Higher Order Strict Avalanche Criterion," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 102–117.

400. T.W. Cusick and M.C. Wood, "The REDOC-II Cryptosystem," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 545–563.

401. Cylink Corporation, Cylink Corporation vs. RSA Data Security, Inc., Civil Action No. C94-02332-CW, United States District Court for the Northern District of California, 30 Jun 1994.

402. J. Daeman, "Cipher and Hash Function Design," Ph.D. Thesis, Katholieke Universiteit Leuven, Mar 95.

403. J. Daeman, A. Bosselaers, R. Govaerts, and J. Vandewalle, "Collisions for Schnorr's Hash Function FFT-Hash Presented at Crypto '91," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1993, pp. 477–480.

404. J. Daeman, R. Govaerts, and J. Vandewalle, "A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård's One-Way Function Based on Cellular Automata," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1993, pp. 82–96.

405. J. Daeman, R. Govaerts, and J. Vandewalle, "A Hardware Design Model for Cryptographic Algorithms," *ESORICS 92, Proceedings of the Second European Symposium on Research in Computer Security*, Springer-Verlag, 1992, pp. 419–434.

406. J. Daemen, R. Govaerts, and J. Vandewalle, "Block Ciphers Based on Modular Arith-

metic," *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography*, Rome, Italy, 15–16 Feb 1993, pp. 80–89.

407. J. Daemen, R. Govaerts, and J. Vandewalle, "Fast Hashing Both in Hardware and Software," presented at the rump session of CRYPTO '93, Aug 1993.
408. J. Daeman, R. Govaerts, and J. Vandewalle, "Resynchronization Weaknesses in Synchronous Stream Ciphers," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 159–167.
409. J. Daeman, R. Govaerts, and J. Vandewalle, "Weak Keys for IDEA," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 224–230.
410. J. Daemen, R. Govaerts, and J. Vandewalle, "A New Approach to Block Cipher Design," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 18–32.
411. Z.-D. Dai, "Proof of Rueppel's Linear Complexity Conjecture," *IEEE Transactions on Information Theory*, v. IT-32, n. 3, May 1986, pp. 440–443.
412. I.B. Damgård, "Collision Free Hash Functions and Public Key Signature Schemes," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 203–216.
413. I.B. Damgård, "Payment Systems and Credential Mechanisms with Provable Security Against Abuse by Individuals," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 328–335.
414. I.B. Damgård, "A Design Principle for Hash Functions," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 416–427.
415. I.B. Damgård, "Practical and Provably Secure Release of a Secret and Exchange of Signatures," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 200–217.
416. I.B. Damgård and L.R. Knudsen, "The Breaking of the AR Hash Function," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 286–292.
417. I.B. Damgård and P. Landrock, "Improved Bounds for the Rabin Primality Test," *Cryptography and Coding III*, M.J. Ganley, ed., Oxford: Clarendon Press, 1993, pp. 117–128.
418. I.B. Damgård, P. Landrock and C. Pomerance, "Average Case Error Estimates for the Strong Probable Prime Test," *Mathematics of Computation*, v. 61, n. 203, Jul 1993, pp. 177–194.
419. H.E. Daniels, Jr., letter to Datapro Research Corporation regarding CCEP, 23 Dec 1985.
420. H. Davenport, *The Higher Arithmetic*, Dover Books, 1983.
421. G.I. Davida, "Inverse of Elements of a Galois Field," *Electronics Letters*, v. 8, n. 21, 19 Oct 1972, pp. 518–520.
422. G.I. Davida, "Hellman's Scheme Breaks DES in Its Basic Form," *IEEE Spectrum*, v. 16, n. 7, Jul 1979, p. 39.
423. G.I. Davida, "Chosen Signature Cryptanalysis of the RSA (MIT) Public Key Cryptosystem," *Technical Report TR-CS-82-2*, Department of EECS, University of Wisconsin, 1982.
424. G.I. Davida and G.G. Walter, "A Public Key Analog Cryptosystem," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 143–147.
425. G.I. Davida, D. Wells, and J. Kam, "A Database Encryption System with Subkeys," *ACM Transactions on Database Systems*, v. 6, n. 2, Jun 1981, pp. 312–328.
426. D.W. Davies, "Applying the RSA Digital Signature to Electronic Mail," *Computer*, v. 16, n. 2, Feb 1983, pp. 55–62.
427. D.W. Davies, "Some Regular Properties of the DES," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 89–96.
428. D.W. Davies, "A Message Authentication Algorithm Suitable for a Mainframe Computer," *Advances in Cryptology: Proceedings of Crypto 82*, Springer-Verlag, 1985, pp. 393–400.
429. D.W. Davies and S. Murphy, "Pairs and Triplets of DES S-boxes," *Cryptologia*, v. 8, n. 1, 1995, pp. 1–25.
430. D.W. Davies and G.I.P. Parkin, "The Average Size of the Key Stream in Output Feedback Encipherment," *Cryptography, Proceedings of the Workshop on Cryptography, Burg Feuerstein, Germany, March 29–April 2, 1982*, Springer-Verlag, 1983, pp. 263–279.
431. D.W. Davies and G.I.P. Parkin, "The Average Size of the Key Stream in Output Feed-

back Mode," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 97–98.

432. D.W. Davies and W.L. Price, "The Application of Digital Signatures Based on Public-Key Cryptosystems," *Proceedings of the Fifth International Computer Communications Conference*, Oct 1980, pp. 525–530.
433. D.W. Davies and W.L. Price, "The Application of Digital Signatures Based on Public-Key Cryptosystems," National Physical Laboratory Report DNACS 39/80, Dec 1980.
434. D.W. Davies and W.L. Price, "Digital Signature—An Update," *Proceedings of International Conference on Computer Communications, Sydney, Oct 1984*, North Holland: Elsevier, 1985, pp. 843–847.
435. D.W. Davies and W.L. Price, *Security for Computer Networks*, second edition, John Wiley & Sons, 1989.
436. M. Davio, Y. Desmedt, M. Fosseprez, R. Govaerts, J. Hulsbrosch, P. Neutjens, P. Piret, J.-J. Quisquater, J. Vandewalle, and S. Wouters, "Analytical Characteristics of the Data Encryption Standard," *Advances in Cryptology: Proceedings of Crypto 83*, Plenum Press, 1984, pp. 171–202.
437. M. Davio, Y. Desmedt, J. Goubert, F. Hoornaert, and J.-J. Quisquater, "Efficient Hardware and Software Implementation of the DES," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 144–146.
438. M. Davio, Y. Desmedt, and J.-J. Quisquater, "Propagation Characteristics of the DES," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, 62–73.
439. D. Davis, R. Ihaka, and P. Fenstermacher, "Cryptographic Randomness from Air Turbulence in Disk Drives," *Advances in Cryptology—CRYPTO '94 Proceedings*, Springer-Verlag, 1994, pp. 114–120.
440. J.A. Davis, D.B. Holdbridge, and G.J. Simmons, "Status Report on Factoring (at the Sandia National Laboratories)," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 183–215.
441. R.M. Davis, "The Data Encryption Standard in Perspective," *Computer Security and the Data Encryption Standard*, National Bureau of Standards Special Publication 500-27, Feb 1978.
442. E. Dawson and A. Clard, "Cryptanalysis of Universal Logic Sequences," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, to appear.
443. M.H. Dawson and S.E. Tavares, "An Expanded Set of Design Criteria for Substitution Boxes and Their Use in Strengthening DES-Like Cryptosystems," *IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing*, Victoria, BC, Canada, 9–10 May 1991, pp. 191–195.
444. M.H. Dawson and S.E. Tavares, "An Expanded Set of S-Box Design Criteria Based on Information Theory and Its Relation to Differential-like Attacks," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 352–367.
445. C.A. Deavours, "Unicity Points in Cryptanalysis," *Cryptologia*, v. 1, n. 1, 1977, pp. 46–68.
446. C.A. Deavours, "The Black Chamber: A Column; How the British Broke Enigma," *Cryptologia*, v. 4, n. 3, Jul 1980, pp. 129–132.
447. C.A. Deavours, "The Black Chamber: A Column; La Méthode des Bâtons," *Cryptologia*, v. 4, n. 4, Oct 1980, pp. 240–247.
448. C.A. Deavours and L. Kruh, *Machine Cryptography and Modern Cryptanalysis*, Norwood MA: Artech House, 1985.
449. J.M. DeLaurentis, "A Further Weakness in the Common Modulus Protocol for the RSA Cryptosystem," *Cryptologia*, v. 8, n. 3, Jul 1984, pp. 253–259.
450. P. Delsarte, Y. Desmedt, A. Odlyzko, and P. Piret, "Fast Cryptanalysis of the Matsumoto-Imai Public-Key Scheme," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 142–149.
451. P. Delsarte and P. Piret, "Comment on 'Extension of RSA Cryptostructure: A Galois Approach'," *Electronics Letters*, v. 18, n. 13, 24 Jun 1982, pp. 582–583.
452. R. DeMillo, N. Lynch, and M. Merritt, "Cryptographic Protocols," *Proceedings of the 14th Annual Symposium on the Theory of Computing*, 1982, pp. 383–400.
453. R. DeMillo and M. Merritt, "Protocols for Data Security," *Computer*, v. 16, n. 2, Feb 1983, pp. 39–50.
454. N. Demytko, "A New Elliptic Curve Based Analogue of RSA," *Advances in Cryptol-*

ogy—*EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 40–49.

455. D.E. Denning, "Secure Personal Computing in an Insecure Network," *Communications of the ACM*, v. 22, n. 8, Aug 1979, pp. 476–482.
456. D.E. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.
457. D.E. Denning, "Protecting Public Keys and Signature Keys," *Computer*, v. 16, n. 2, Feb 1983, pp. 27–35.
458. D.E. Denning, "Digital Signatures with RSA and Other Public-Key Cryptosystems," *Communications of the ACM*, v. 27, n. 4, Apr 1984, pp. 388–392.
459. D.E. Denning, "The Data Encryption Standard: Fifteen Years of Public Scrutiny," *Proceedings of the Sixth Annual Computer Security Applications Conference*, IEEE Computer Society Press, 1990.
460. D.E. Denning, "The Clipper Chip: A Technical Summary," unpublished manuscript, 21 Apr 1993.
461. D.E. Denning and G.M. Sacco, "Timestamps in Key Distribution Protocols," *Communications of the ACM*, v. 24, n. 8, Aug 1981, pp. 533–536.
462. D.E. Denning and M. Smid, "Key Escrowing Today," *IEEE Communications Magazine*, v. 32, n. 9, Sep 1994, pp. 58–68.
463. T. Denny, B. Dodson, A.K. Lenstra, and M.S. Manasse, "On the Factorization of RSA-120," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 166–174.
464. W.F. Denny, "Encryptions Using Linear and Non-Linear Codes: Implementations and Security Considerations," Ph.D. dissertation, The Center for Advanced Computer Studies, University of Southern Louisiana, Spring 1988.
465. Department of Defense, "Department of Defense Trusted Computer System Evaluation Criteria," DOD 5200.28-STD, Dec 1985.
466. Department of State, "International Traffic in Arms Regulations (ITAR)," 22 CFR 120–130, Office of Munitions Control, Nov 1989.
467. Department of State, "Defense Trade Regulations," 22 CFR 120–130, Office of Defense Trade Controls, May 1992.
468. Department of the Treasury, "Electronic Funds and Securities Transfer Policy," Department of the Treasury Directives Manual, Chapter TD 81, Section 80, Department of the Treasury, 16 Aug 1984.
469. Department of the Treasury, "Criteria and Procedures for Testing, Evaluating, and Certifying Message Authentication Decisions for Federal E.F.T. Use," Department of the Treasury, 1 May 1985.
470. Department of the Treasury, "Electronic Funds and Securities Transfer Policy—Message Authentication and Enhanced Security," Order No. 106-09, Department of the Treasury, 2 Oct 1986.
471. H. Dobbertin, "A Survey on the Construction of Bent Functions," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
472. B. Dodson and A.K. Lenstra, "NFS with Four Large Primes: An Explosive Experiment," draft manuscript.
473. D. Dolev and A. Yao, "On the Security of Public-Key Protocols," *Communications of the ACM*, v. 29, n. 8, Aug 1983, pp. 198–208.
474. J. Domingo-Ferrer, "Probabilistic Authentication Analysis," *CARDIS 94—Proceedings of the First Smart Card Research and Applications Conference*, Lille, France, 24–26 Oct 1994, pp. 49–60.
475. P. de Rooij, "On the Security of the Schnorr Scheme Using Preprocessing," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 71–80.
476. A. De Santis, G. Di Crescenzo, and G. Persiano, "Secret Sharing and Perfect Zero Knowledge," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 73–84.
477. A. De Santis, S. Micali, and G. Persiano, "Non-Interactive Zero-Knowledge Proof Systems," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 52–72.
478. A. De Santis, S. Micali, and G. Persiano, "Non-Interactive Zero-Knowledge with Preprocessing," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 269–282.
479. Y. Desmedt, "What Happened with Knapsack Cryptographic Schemes" *Performance Limits in Communication, Theory and Practice*, NATO ASI Series E: Applied Sciences, v. 142, Kluwer Academic Publishers, 1988, pp. 113–134.
480. Y. Desmedt, "Subliminal-Free Authentication and Signature," *Advances in Cryptol-*

ogy—EUROCRYPT '88 Proceedings, Springer-Verlag, 1988, pp. 23–33.

481. Y. Desmedt, "Abuses in Cryptography and How to Fight Them," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 375–389.
482. Y. Desmedt and M. Burmester, "An Efficient Zero-Knowledge Scheme for the Discrete Logarithm Based on Smooth Numbers," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1993, pp. 360–367.
483. Y. Desmedt and Y. Frankel, "Threshold Cryptosystems," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 307–315.
484. Y. Desmedt and Y. Frankel, "Shared Generation of Authentication and Signatures," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 3457–469.
485. Y. Desmedt, C. Goutier, and S. Bengio, "Special Uses and Abuses of the Fiat-Shamir Passport Protocol," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 21–39.
486. Y. Desmedt and A.M. Odlykzo, "A Chosen Text Attack on the RSA Cryptosystem and Some Discrete Logarithm Problems," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 516–522.
487. Y. Desmedt, J.-J. Quisquater, and M. Davio, "Dependence of Output on Input in DES: Small Avalanche Characteristics," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 359–376.
488. Y. Desmedt, J. Vandewalle, and R. Govaerts, "Critical Analysis of the Security of Knapsack Public Key Algorithms," *IEEE Transactions on Information Theory*, v. IT-30, n. 4, Jul 1984, pp. 601–611.
489. Y. Desmedt and M. Yung, "Weaknesses of Undeniable Signature Schemes," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 205–220.
490. W. Diffie, lecture at IEEE Information Theory Workshop, Ithaca, N.Y., 1977.
491. W. Diffie, "Cryptographic Technology: Fifteen Year Forecast," BNR Inc., Jan 1981.
492. W. Diffie, "The First Ten Years of Public-Key Cryptography," *Proceedings of the IEEE*, v. 76, n. 5, May 1988, pp. 560–577.
493. W. Diffie, "Authenticated Key Exchange and Secure Interactive Communication," *Proceedings of SECURICOM '90*, 1990.
494. W. Diffie, "The First Ten Years of Public-Key Cryptography," in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 135–175.
495. W. Diffie and M.E. Hellman, "Multiuser Cryptographic Techniques," *Proceedings of AFIPS National Computer Conference*, 1976, pp. 109–112.
496. W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, v. IT-22, n. 6, Nov 1976, pp. 644–654.
497. W. Diffie and M.E. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *Computer*, v. 10, n. 6, Jun 1977, pp. 74–84.
498. W. Diffie and M.E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," *Proceedings of the IEEE*, v. 67, n. 3, Mar 1979, pp. 397–427.
499. W. Diffie, L. Strawczynski, B. O'Higgins, and D. Steer, "An ISDN Secure Telephone Unit," *Proceedings of the National Telecommunications Forum*, v. 41, n. 1, 1987, pp. 473–477.
500. W. Diffie, P.C. van Oorschot, and M.J. Wiener, "Authentication and Authenticated Key Exchanges," *Designs, Codes and Cryptography*, v. 2, 1992, 107–125.
501. C. Ding, "The Differential Cryptanalysis and Design of Natural Stream Ciphers," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 101–115.
502. C. Ding, G. Xiao, and W. Shan, *The Stability Theory of Stream Ciphers*, Springer-Verlag, 1991.
503. A. Di Porto and W. Wolfowicz, "VINO: A Block Cipher Including Variable Permutations," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 205–210.
504. B. Dixon and A.K. Lenstra, "Factoring Integers Using SIMD Sieves," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 28–39.
505. J.D. Dixon, "Factorization and Primality Tests," *American Mathematical Monthly*, v. 91, n. 6, 1984, pp. 333–352.
506. D. Dolev and A. Yao, "On the Security of Public Key Protocols," *Proceedings of the*

22nd Annual Symposium on the Foundations of Computer Science, 1981, pp. 350–357.

507. L.X. Duan and C.C. Nian, "Modified Lu-Lee Cryptosystems," *Electronics Letters*, v. 25, n. 13, 22 Jun 1989, p. 826.
508. R. Durstenfeld, "Algorithm 235: Random Permutation," *Communications of the ACM*, v. 7, n. 7, Jul 1964, p. 420.
509. S. Dussé and B. Kaliski, Jr., "A Cryptographic Library for the Motorola DSP56000," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 230–244.
510. C. Dwork and L. Stockmeyer, "Zero-Knowledge with Finite State Verifiers," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 71–75.
511. D.E. Eastlake, S.D. Crocker, and J.I. Schiller, "Randomness Requirements for Security," RFC 1750, Dec 1994.
512. H. Eberle, "A High-Speed DES Implementation for Network Applications," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, pp. 521–539.
513. J. Edwards, "Implementing Electronic Poker: A Practical Exercise in Zero-Knowledge Interactive Proofs," Master's thesis, Department of Computer Science, University of Kentucky, May 1994.
514. W.F. Ehrsam, C.H.W. Meyer, R.L. Powers, J.L. Smith, and W.L. Tuchman, "Product Block Cipher for Data Security," U.S. Patent #3,962,539, 8 Jun 1976.
515. W.F. Ehrsam, C.H.W. Meyer, and W.L. Tuchman, "A Cryptographic Key Management Scheme for Implementing the Data Encryption Standard," *IBM Systems Journal*, v. 17, n. 2, 1978, pp. 106–125.
516. R. Eier and H. Lagger, "Trapdoors in Knapsack Cryptosystems," *Lecture Notes in Computer Science 149; Cryptography—Proceedings, Burg Feuerstein 1982*, Springer-Verlag, 1983, pp. 316–322.
517. A.K. Ekert, "Quantum Cryptography Based on Bell's Theorem," *Physical Review Letters*, v. 67, n. 6, Aug 1991, pp. 661–663.
518. T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 10–18.
519. T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, v. IT-31, n. 4, 1985, pp. 469–472.
520. T. ElGamal, "On Computing Logarithms Over Finite Fields," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 396–402.
521. T. ElGamal and B. Kaliski, letter to the editor regarding LUC, *Dr. Dobb's Journal*, v. 18, n. 5, May 1993, p. 10.
522. T. Eng and T. Okamoto, "Single-Term Divisible Electronic Coins," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
523. M.H. Er, D.J. Wong, A.A. Sethu, and K.S. Ngeow, "Design and Implementation of RSA Cryptosystem Using Multiple DSP Chips," *1991 IEEE International Symposium on Circuits and Systems*, v. 1, Singapore, 11–14 Jun 1991, pp. 49–52.
524. D. Estes, L.M. Adleman, K. Konpella, K.S. McCurley, and G.L. Miller, "Breaking the Ong-Schnorr-Shamir Signature Schemes for Quadratic Number Fields," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 3–13.
525. ETEBAC, "Échanges Télématiques Entre Les Banques et Leurs Clients," Standard ETEBAC 5, *Comité Français d'Organisation et de Normalisation Bancaires*, Apr 1989. (In French.)
526. A. Evans, W. Kantrowitz, and E. Weiss, "A User Identification Scheme Not Requiring Secrecy in the Computer," *Communications of the ACM*, v. 17, n. 8, Aug 1974, pp. 437–472.
527. S. Even and O. Goldreich, "DES-Like Functions Can Generate the Alternating Group," *IEEE Transactions on Information Theory*, v. IT-29, n. 6, Nov 1983, pp. 863–865.
528. S. Even and O. Goldreich, "On the Power of Cascade Ciphers," *ACM Transactions on Computer Systems*, v. 3, n. 2, May 1985, pp. 108–116.
529. S. Even, O. Goldreich, and A. Lempel, "A Randomizing Protocol for Signing Contracts," *Communications of the ACM*, v. 28, n. 6, Jun 1985, pp. 637–647.
530. S. Even and Y. Yacobi, "Cryptography and **NP**-Completeness," *Proceedings of the 7th International Colloquium on Automata,*

Languages, and Programming, Springer-Verlag, 1980, pp. 195–207.

531. H.-H. Evertse, "Linear Structures in Block Ciphers," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 249–266.
532. P. Fahn and M.J.B. Robshaw, "Results from the RSA Factoring Challenge," Technical Report TR-501, Version 1.3, RSA Laboratories, Jan 1995.
533. R.C. Fairfield, A. Matusevich, and J. Plany, "An LSI Digital Encryption Processor (DEP)," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 115–143.
534. R.C. Fairfield, A. Matusevich, and J. Plany, "An LSI Digital Encryption Processor (DEP)," *IEEE Communications*, v. 23, n. 7, Jul 1985, pp. 30–41.
535. R.C. Fairfield, R.L. Mortenson, and K.B. Koulthart, "An LSI Random Number Generator (RNG)," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 203–230.
536. "International Business Machines Corp. License Under Patents," *Federal Register*, v. 40, n. 52, 17 Mar 1975, p. 12067.
537. "Solicitation for Public Key Cryptographic Algorithms," *Federal Register*, v. 47, n. 126, 30 Jun 1982, p. 28445.
538. "Proposed Federal Information Processing Standard for Digital Signature Standard (DSS)," *Federal Register*, v. 56, n. 169, 30 Aug 1991, pp. 42980–42982.
539. "Proposed Federal Information Processing Standard for Secure Hash Standard," *Federal Register*, v. 57, n. 21, 31 Jan 1992, pp. 3747–3749.
540. "Proposed Reaffirmation of Federal Information Processing Standard (FIPS) 46-1, Data Encryption Standard (DES)," *Federal Register*, v. 57, n. 177, 11 Sep 1992, p. 41727.
541. "Notice of Proposal for Grant of Exclusive Patent License," *Federal Register*, v. 58, n. 108, 8 Jun 1993, pp. 23105–23106.
542. "Approval of Federal Information Processing Standards Publication 186, Digital Signature Standard (DSS)," *Federal Register*, v. 58, n. 96, 19 May 1994, pp. 26208–26211.
543. "Proposed Revision of Federal Information Processing Standard (FIPS) 180, Secure Hash Standard," *Federal Register*, v. 59, n. 131, 11 Jul 1994, pp. 35317–35318.
544. U. Feige, A. Fiat, and A. Shamir, "Zero Knowledge Proofs of Identity," *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, 1987, pp. 210–217.
545. U. Feige, A. Fiat, and A. Shamir, "Zero Knowledge Proofs of Identity," *Journal of Cryptology*, v. 1, n. 2, 1988, pp. 77–94.
546. U. Feige and A. Shamir, "Zero Knowledge Proofs of Knowledge in Two Rounds," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 526–544.
547. J. Feigenbaum, "Encrypting Problem Instances, or, . . . , Can You Take Advantage of Someone Without Having to Trust Him," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 477–488.
548. J. Feigenbaum, "Overview of Interactive Proof Systems and Zero-Knowledge," in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 423–439.
549. J. Feigenbaum, E. Grosse, and J.A. Reeds, "Cryptographic Protection of Membership Lists," *Newsletter of the International Association of Cryptologic Research*, v. 9, 1992, pp. 16–20.
550. J. Feigenbaum, M.Y. Liverman, and R.N. Wright, "Cryptographic Protection of Databases and Software," *Distributed Computing and Cryptography*, J. Feigenbaum and M. Merritt, eds., American Mathematical Society, 1991, pp. 161–172.
551. H. Feistel, "Cryptographic Coding for Data-Bank Privacy," RC 2827, Yorktown Heights, NY: IBM Research, Mar 1970.
552. H. Feistel, "Cryptography and Computer Privacy," *Scientific American*, v. 228, n. 5, May 1973, pp. 15–23.
553. H. Feistel, "Block Cipher Cryptographic System," U.S. Patent #3,798,359, 19 Mar 1974.
554. H. Feistel, "Step Code Ciphering System," U.S. Patent #3,798,360, 19 Mar 1974.
555. H. Feistel, "Centralized Verification System," U.S. Patent #3,798,605, 19 Mar 1974.
556. H. Feistel, W.A. Notz, and J.L. Smith, "Cryptographic Techniques for Machine to Machine Data Communications," RC 3663, Yorktown Heights, N.Y.: IBM Research, Dec 1971.

557. H. Feistel, W.A. Notz, and J.L. Smith, "Some Cryptographic Techniques for Machine to Machine Data Communications," *Proceedings of the IEEE*, v. 63, n. 11, Nov 1975, pp. 1545–1554.

558. P. Feldman, "A Practical Scheme for Non-interactive Verifiable Secret Sharing," *Proceedings of the 28th Annual Symposium on the Foundations of Computer Science*, 1987, pp. 427–437.

559. R.A. Feldman, "Fast Spectral Test for Measuring Nonrandomness and the DES," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 243–254.

560. R.A. Feldman, "A New Spectral Test for Nonrandomness and the DES," *IEEE Transactions on Software Engineering*, v. 16, n. 3, Mar 1990, pp. 261–267.

561. D.C. Feldmeier and P.R. Karn, "UNIX Password Security—Ten Years Later," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 44–63.

562. H. Fell and W. Diffie, "Analysis of a Public Key Approach Based on Polynomial Substitution," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 427–437.

563. N.T. Ferguson, "Single Term Off-Line Coins," Report CS-R9318, Computer Science/Department of Algorithms and Architecture, CWI, Mar 1993.

564. N.T. Ferguson, "Single Term Off-Line Coins," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 318–328.

565. N.T. Ferguson, "Extensions of Single-term Coins," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 292–301.

566. A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 186–194.

567. A. Fiat and A. Shamir, "Unforgeable Proofs of Identity," *Proceedings of Securicom 87*, Paris, 1987, pp. 147–153.

568. P. Finch, "A Study of the Blowfish Encryption Algorithm," Ph.D. dissertation, Department of Computer Science, City University of New York Graduate School and University Center, Feb 1995.

569. R. Flynn and A.S. Campasano, "Data Dependent Keys for Selective Encryption Terminal," *Proceedings of NCC, vol. 47*, AFIPS Press, 1978, pp. 1127–1129.

570. R.H. Follett, letter to NIST regarding DSS, 25 Nov 1991.

571. R. Forré, "The Strict Avalanche Criterion: Spectral Properties and an Extended Definition," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 450–468.

572. R. Forré, "A Fast Correlation Attack on Nonlinearity Feedforward Filtered Shift Register Sequences," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 568–595.

573. S. Fortune and M. Merritt, "Poker Protocols," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 454–464.

574. R.B. Fougner, "Public Key Standards and Licenses," RFC 1170, Jan 1991.

575. Y. Frankel and M. Yung, "Escrowed Encryption Systems Visited: Threats, Attacks, Analysis and Designs," *Advances in Cryptology—CRYPTO '95 Proceedings*, Springer-Verlag, 1995, to appear.

576. W.F. Friedman, *Methods for the Solution of Running-Key Ciphers*, Riverbank Publication No. 16, Riverbank Labs, 1918.

577. W.F. Friedman, *The Index of Coincidence and Its Applications in Cryptography*, Riverbank Publication No. 22, Riverbank Labs, 1920. Reprinted by Aegean Park Press, 1987.

578. W.F. Friedman, *Elements of Cryptanalysis*, Laguna Hills, CA: Aegean Park Press, 1976.

579. W.F. Friedman, "Cryptology," *Encyclopedia Britannica*, v. 6, pp. 844–851, 1967.

580. A.M. Frieze, J. Hastad, R. Kannan, J.C. Lagarias, and A. Shamir, "Reconstructing Truncated Integer Variables Satisfying Linear Congruences," *SIAM Journal on Computing*, v. 17, n. 2, Apr 1988, pp. 262–280.

581. A.M. Frieze, R. Kannan, and J.C. Lagarias, "Linear Congruential Generators Do not Produce Random Sequences," *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, 1984, pp. 480–484.

582. E. Fujiaski and T. Okamoto, "On Comparison of Practical Digitial Signature Schemes," *Proceedings of the 1992 Sym-*

posium on Cryptography and Information Security (SCIS 92), Tateshina, Japan, 2–4 Apr 1994, pp. 1A.1–12.

583. A. Fujioka, T. Okamoto, and S. Miyaguchi, "ESIGN: An Efficient Digital Signature Implementation for Smart Cards," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 446–457.
584. A. Fujioka, T. Okamoto, and K. Ohta, "Interactive Bi-Proof Systems and Undeniable Signature Schemes," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 243–256.
585. A. Fujioka, T. Okamoto, and K. Ohta, "A Practical Secret Voting Scheme for Large Scale Elections," *Advances in Cryptology—AUSCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 244–251.
586. K. Gaardner and E. Snekkenes, "Applying a Formal Analysis Technique to the CCITT X.509 Strong Two-Way Authentication Protocol," *Journal of Cryptology*, v. 3, n. 2, 1991, pp. 81–98.
587. H.F. Gaines, *Cryptanalysis*, American Photographic Press, 1937. [Reprinted by Dover Publications, 1956.]
588. J. Gait, "A New Nonlinear Pseudorandom Number Generator," *IEEE Transactions on Software Engineering*, v. SE-3, n. 5, Sep 1977, pp. 359–363.
589. J. Gait, "Short Cycling in the Kravitz-Reed Public Key Encryption System," *Electronics Letters*, v. 18, n. 16, 5 Aug 1982, pp. 706–707.
590. Z. Galil, S. Haber, and M. Yung, "A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystems," *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, 1985, pp. 360–371.
591. Z. Galil, S. Haber, and M. Yung, "Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 135–155.
592. Z. Galil, S. Haber, and M. Yung, "Minimum-Knowledge Interactive Proofs for Decision Problems," *SIAM Journal on Computing*, v. 18, n. 4, 1989, pp. 711–739.
593. R.G. Gallager, *Information Theory and Reliable Communications*, New York: John Wiley & Sons, 1968.
594. P. Gallay and E. Depret, "A Cryptography Microprocessor," *1988 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 1988, pp. 148–149.
595. R.A. Games, "There are no de Bruijn Sequences of Span n with Complexity $2^{n-1} + n + 1$," *Journal of Combinatorical Theory*, Series A, v. 34, n. 2, Mar 1983, pp. 248–251.
596. R.A. Games and A.H. Chan, "A Fast Algorithm for Determining the Complexity of a Binary Sequence with 2^n," *IEEE Transactions on Information Theory*, v. IT-29, n. 1, Jan 1983, pp. 144–146.
597. R.A. Games, A.H. Chan, and E.L. Key, "On the Complexity of de Bruijn Sequences," *Journal of Combinatorical Theory*, Series A, v. 33, n. 1, Nov 1982, pp. 233–246.
598. S.H. Gao and G.L. Mullen, "Dickson Polynomials and Irreducible Polynomials over Finite Fields," *Journal of Number Theory*, v. 49, n. 1, Oct 1994, pp. 18–132.
599. M. Gardner, "A New Kind of Cipher That Would Take Millions of Years to Break," *Scientific American*, v. 237, n. 8, Aug 1977, pp. 120–124.
600. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of **NP**-Completeness*, W.H. Freeman and Co., 1979.
601. S.L. Garfinkel, *PGP: Pretty Good Privacy*, Sebastopol, CA: O'Reilly and Associates, 1995.
602. C.W. Gardiner, "Distributed Public Key Certificate Management," *Proceedings of the Privacy and Security Research Group 1993 Workshop on Network and Distributed System Security*, The Internet Society, 1993, pp. 69–73.
603. G. Garon and R. Outerbridge, "DES Watch: An Examination of the Sufficiency of the Data Encryption Standard for Financial Institution Information Security in the 1990's," *Cryptologia*, v. 15, n. 3, Jul 1991, pp. 177–193.
604. M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson, "The Digital Distributed Systems Security Architecture," *Proceedings of the 12th National Computer Security Conference*, NIST, 1989, pp. 305–319.
605. J. von zur Gathen, D. Kozen, and S. Landau, "Functional Decomposition of Polynomials," *Proceedings of the 28th IEEE Symposium on the Foundations of Com-*

puter Science, IEEE Press, 1987, pp. 127–131.

606. P.R. Geffe, "How to Protect Data With Ciphers That are Really Hard to Break," *Electronics*, v. 46, n. 1, Jan 1973, pp. 99–101.
607. D.K. Gifford, D. Heitmann, D.A. Segal, R.G. Cote, K. Tanacea, and D.E. Burmaster, "Boston Community Information System 1986 Experimental Test Results," MIT/LCS/TR-397, MIT Laboratory for Computer Science, Aug 1987.
608. D.K. Gifford, J.M. Lucassen, and S.T. Berlin, "The Application of Digital Broadcast Communication to Large Scale Information Systems," *IEEE Journal on Selected Areas in Communications*, v. 3, n. 3, May 1985, pp. 457–467.
609. D.K. Gifford and D.A. Segal, "Boston Community Information System 1987–1988 Experimental Test Results," MIT/LCS/TR-422, MIT Laboratory for Computer Science, May 1989.
610. H. Gilbert and G. Chase, "A Statistical Attack on the Feal-8 Cryptosystem," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 22–33.
611. H. Gilbert and P. Chauvaud, "A Chosen Plaintext Attack of the 16-Round Khufu Cryptosystem," *Advances in Cryptology—CRYPTO '94 Proceedings*, Springer-Verlag, 1994, pp. 259–268.
612. M. Girault, "Hash-Functions Using Modulo-*N* Operations," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 217–226.
613. J. Gleick, "A New Approach to Protecting Secrets is Discovered," *The New York Times*, 18 Feb 1987, pp. C1 and C3.
614. J.-M. Goethals and C. Couvreur, "A Cryptanalytic Attack on the Lu-Lee Public-Key Cryptosystem," *Philips Journal of Research*, v. 35, 1980, pp. 301–306.
615. O. Goldreich, "A Uniform-Complexity Treatment of Encryption and Zero-Knowledge, *Journal of Cryptology*, v. 6, n. 1, 1993, pp. 21–53.
616. O. Goldreich and H. Krawczyk, "On the Composition of Zero Knowledge Proof Systems," *Proceedings on the 17th International Colloquium on Automata, Languages, and Programming*, Springer-Verlag, 1990, pp. 268–282.
617. O. Goldreich and E. Kushilevitz, "A Perfect Zero-Knowledge Proof for a Problem Equivalent to Discrete Logarithm," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 58–70.
618. O. Goldreich and E. Kushilevitz, "A Perfect Zero-Knowledge Proof for a Problem Equivalent to Discrete Logarithm," *Journal of Cryptology*, v. 6, n. 2, 1993, pp. 97–116.
619. O. Goldreich, S. Micali, and A. Wigderson, "Proofs That Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design," *Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science*, 1986, pp. 174–187.
620. O. Goldreich, S. Micali, and A. Wigderson, "How to Prove All **NP** Statements in Zero Knowledge and a Methodology of Cryptographic Protocol Design," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 171–185.
621. O. Goldreich, S. Micali, and A. Wigderson, "How to Play Any Mental Game," *Proceedings of the 19th ACM Symposium on the Theory of Computing*, 1987, pp. 218–229.
622. O. Goldreich, S. Micali, and A. Wigderson, "Proofs That Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design," *Journal of the ACM*, v. 38, n. 1, Jul 1991, pp. 691–729.
623. S. Goldwasser and J. Kilian, "Almost All Primes Can Be Quickly Certified," *Proceedings of the 18th ACM Symposium on the Theory of Computing*, 1986, pp. 316–329.
624. S. Goldwasser and S. Micali, "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information," *Proceedings of the 14th ACM Symposium on the Theory of Computing*, 1982, pp. 270–299.
625. S. Goldwasser and S. Micali, "Probabilistic Encryption," *Journal of Computer and System Sciences*, v. 28, n. 2, Apr 1984, pp. 270–299.
626. S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *Proceedings of the 17th ACM Symposium on Theory of Computing*, 1985, pp. 291–304.

627. S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *SIAM Journal on Computing*, v. 18, n. 1, Feb 1989, pp. 186–208.

628. S. Goldwasser, S. Micali, and R.L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," *SIAM Journal on Computing*, v. 17, n. 2, Apr 1988, pp. 281–308.

629. S. Goldwasser, S. Micali, and A.C. Yao, "On Signatures and Authentication," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 211–215.

630. J.D. Golić, "On the Linear Complexity of Functions of Periodic GF(*q*) Sequences," *IEEE Transactions on Information Theory*, v. IT-35, n. 1, Jan 1989, pp. 69–75.

631. J.D. Golić, "Linear Cryptanalysis of Stream Ciphers," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, pp. 262–282.

632. J.D. Golić, "Towards Fast Correlation Attacks on Irregularly Clocked Shift Registers," *Advances in Cryptology—EUROCRYPT '95 Proceedings*, Springer-Verlag, 1995, to appear.

633. J.D. Golić and M.J. Mihajlevič, "A Generalized Correlation Attack on a Class of Stream Ciphers Based on the Levenshtein Distance," *Journal of Cryptology*, v. 3, n. 3, 1991, pp. 201–212.

634. J.D. Golić and L. O'Connor, "Embedding and Probabilistic Correlation Attacks on Clock-Controlled Shift Registers," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.

635. R. Golliver, A.K. Lenstra, K.S. McCurley, "Lattice Sieving and Trial Division," *Proceedings of the Algorithmic Number Theory Symposium*, Cornell, 1994, to appear.

636. D. Gollmann, "Kaskadenschaltungen taktgesteuerter Schieberegister als Pseudozufallszahlengeneratoren," Ph.D. dissertation, Universität Linz, 1983. [In German.]

637. D. Gollmann, "Pseudo Random Properties of Cascade Connections of Clock Controlled Shift Registers," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 93–98.

638. D. Gollmann, "Correlation Analysis of Cascaded Sequences," *Cryptography and Coding*, H.J. Beker and F.C. Piper, eds., Oxford: Clarendon Press, 1989, pp. 289–297.

639. D. Gollmann, "Transformation Matrices of Clock-Controlled Shift Registers," *Cryptography and Coding III*, M.J. Ganley, ed., Oxford: Clarendon Press, 1993, pp. 197–210.

640. D. Gollmann and W.G. Chambers, "Lock-In Effect in Cascades of Clock-Controlled Shift-Registers," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 331–343.

641. D. Gollmann and W.G. Chambers, "Clock-Controlled Shift Registers: A Review," *IEEE Journal on Selected Areas in Communications*, v. 7, n. 4, May 1989, pp. 525–533.

642. D. Gollmann and W.G. Chambers, "A Cryptanalysis of $Step_{k,m}$-cascades," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 680–687.

643. S.W. Golomb, *Shift Register Sequences*, San Francisco: Holden-Day, 1967. [Reprinted by Aegean Park Press, 1982.]

644. L. Gong, "A Security Risk of Depending on Synchronized Clocks," *Operating Systems Review*, v. 26, n. 1, Jan 1992, pp. 49–53.

645. L. Gong, R. Needham, and R. Yahalom, "Reasoning About Belief in Cryptographic Protocols," *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, 1991, pp. 234–248.

646. R.M. Goodman and A.J. McAuley, "A New Trapdoor Knapsack Public Key Cryptosystem," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 150–158.

647. R.M. Goodman and A.J. McAuley, "A New Trapdoor Knapsack Public Key Cryptosystem," *IEE Proceedings*, v. 132, pt. E, n. 6, Nov 1985, pp. 289–292.

648. D.M. Gordon, "Discrete Logarithms Using the Number Field Sieve," Preprint, 28 Mar 1991.

649. D.M. Gordon and K.S. McCurley, "Computation of Discrete Logarithms in Fields of Characteristic Two," presented at the rump session of CRYPTO '91, Aug 1991.

650. D.M. Gordon and K.S. McCurley, "Massively Parallel Computation of Discrete Logarithms," *Advances in Cryptology—*

CRYPTO '92 Proceedings, Springer-Verlag, 1993, pp. 312–323.

651. J.A. Gordon, "Strong Primes are Easy to Find," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 216–223.
652. J.A. Gordon, "Very Simple Method to Find the Minimal Polynomial of an Arbitrary Non-Zero Element of a Finite Field," *Electronics Letters*, v. 12, n. 25, 9 Dec 1976, pp. 663–664.
653. J.A. Gordon and R. Retkin, "Are Big S-Boxes Best?" *Cryptography, Proceedings of the Workshop on Cryptography, Burg Feuerstein, Germany, March 29–April 2, 1982*, Springer-Verlag, 1983, pp. 257–262.
654. M. Goresky and A. Klapper, "Feedback Registers Based on Ramified Extension of the 2-adic Numbers," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
655. GOST, Gosudarstvennyi Standard 28147-89, "Cryptographic Protection for Data Processing Systems," Government Committee of the USSR for Standards, 1989. (In Russian.)
656. GOST R 34.10-94, Gosudarstvennyi Standard of Russian Federation, "Information technology. Cryptographic Data Security. Produce and check procedures of Electronic Digital Signature based on Asymmetric Cryptographic Algorithm." Government Committee of the Russia, 1994. (In Russian.)
657. GOST R 34.11-94, Gosudarstvennyi Standard of Russian Federation, "Information technology. Cryptographic Data Security. Hashing function." Government Committee of the Russia, 1994. (In Russian.)
658. R. Göttfert and H. Neiderreiter, "On the Linear Complexity of Products of Shift-Register Sequences," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 151–158.
659. R. Göttfert and H. Neiderreiter, "A General Lower Bound for the Linear Complexity of the Product of Shift-Register Sequences," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
660. J. van de Graaf and R. Peralta, "A Simple and Secure Way to Show the Validity of Your Public Key," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 128–134.
661. J. Grollman and A.L. Selman, "Complexity Measures for Public-Key Cryptosystems," *Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science*, 1984, pp. 495–503.
662. GSA Federal Standard 1026, "Telecommunications: General Security Requirements for Equipment Using the Data Encryption Standard," General Services Administration, Apr 1982.
663. GSA Federal Standard 1027, "Telecommunications: Interoperability and Security Requirements for Use of the Data Encryption Standard in the Physical and Data Link Layers of Data Communications," General Services Administration, Jan 1983.
664. GSA Federal Standard 1028, "Interoperability and Security Requirements for Use of the Data Encryption Standard with CCITT Group 3 Facsimile Equipment," General Services Administration, Apr 1985.
665. P. Guam, "Cellular Automaton Public Key Cryptosystems," *Complex Systems*, v. 1, 1987, pp. 51–56.
666. H. Guan, "An Analysis of the Finite Automata Public Key Algorithm," *CHINACRYPT '94*, Xidian, China, 11–15 Nov 1994, pp. 120–126. (In Chinese.)
667. G. Guanella, "Means for and Method for Secret Signalling," U.S. Patent #2,405,500, 6 Aug 1946.
668. M. Gude, "Concept for a High-Performance Random Number Generator Based on Physical Random Phenomena," *Frequenz*, v. 39, 1985, pp. 187–190.
669. M. Gude, "Ein quasi-idealer Gleichverteilungsgenerator basierend auf physikalischen Zufallsphänomenen," Ph.D. dissertation, Aachen University of Technology, 1987. (In German.)
670. L.C. Guillou and J.-J. Quisquater, "A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 123–128.
671. L.C. Guillou and J.-J. Quisquater, "A 'Paradoxical' Identity-Based Signature Scheme Resulting from Zero-Knowledge," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 216–231.
672. L.C. Guillou, M. Ugon, and J.-J. Quisquater, "The Smart Card: A Standardized

Security Device Dedicated to Public Cryptology," *Contemporary Cryptology: The Science of Information Integrity*, G. Simmons, ed., IEEE Press, 1992, pp. 561–613.

673. C.G. Günther, "Alternating Step Generators Controlled by de Bruijn Sequences," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 5–14.
674. C.G. Günther, "An Identity-based Key-exchange Protocol," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 29–37.
675. H. Gustafson, E. Dawson, and B. Caelli, "Comparison of Block Ciphers," *Advances in Cryptology—AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 208–220.
676. P. Gutmann, personal communication, 1993.
677. H. Gutowitz, "A Cellular Automaton Cryptosystem: Specification and Call for Attack," unpublished manuscript, Aug 1992.
678. H. Gutowitz, "Method and Apparatus for Encryption, Decryption, and Authentication Using Dynamical Systems," U.S. Patent #5,365,589, 15 Nov 1994.
679. H. Gutowitz, "Cryptography with Dynamical Systems," *Cellular Automata and Cooperative Phenomenon*, Kluwer Academic Press, 1993.
680. R.K. Guy, "How to Factor a Number," *Fifth Manitoba Conference on Numeral Mathematics Congressus Numerantium*, v. 16, 1976, pp. 49–89.
681. R.K. Guy, *Unsolved Problems in Number Theory*, Springer-Verlag, 1981.
682. S. Haber and W.S. Stornetta, "How to Time-Stamp a Digital Document," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 437–455.
683. S. Haber and W.S. Stornetta, "How to Time-Stamp a Digital Document," *Journal of Cryptology*, v. 3, n. 2, 1991, pp. 99–112.
684. S. Haber and W.S. Stornetta, "Digital Document Time-Stamping with Catenate Certificate," U.S. Patent #5,136,646, 4 Aug 1992.
685. S. Haber and W.S. Stornetta, "Method for Secure Time-Stamping of Digital Documents," U.S. Patent #5,136,647, 4 Aug 1992.
686. S. Haber and W.S. Stornetta, "Method of Extending the Validity of a Cryptographic Certificate," U.S. Patent #5,373,561, 13 Dec 1994.
687. T. Habutsu, Y. Nishio, I. Sasase, and S. Mori, "A Secret Key Cryptosystem by Iterating a Chaotic Map," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. E73, n. 7, Jul 1990, pp. 1041–1044.
688. T. Habutsu, Y. Nishio, I. Sasase, and S. Mori, "A Secret Key Cryptosystem by Iterating a Chaotic Map," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 127–140.
689. S. Hada and H. Tanaka, "An Improvement Scheme of DES against Differential Cryptanalysis," *Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94)*, Lake Biwa, Japan, 27–29 Jan 1994, pp 14A.1–11. (In Japanese.)
690. B.C.W. Hagelin, "The Story of the Hagelin Cryptos," *Cryptologia*, v. 18, n. 3, Jul 1994, pp. 204–242.
691. T. Hansen and G.L. Mullen, "Primitive Polynomials over Finite Fields," *Mathematics of Computation*, v. 59, n. 200, Oct 1992, pp. 639–643.
692. S. Harada and S. Kasahara, "An ID-Based Key Sharing Scheme Without Preliminary Communication," IEICE Japan, Technical Report, ISEC89-38, 1989. (In Japanese.)
693. S. Harari, "A Correlation Cryptographic Scheme," *EUROCODE '90—International Symposium on Coding Theory*, Springer-Verlag, 1991, pp. 180–192.
694. T. Hardjono and J. Seberry, "Authentication via Multi-Service Tickets in the Kuperee Server," *Computer Security—ESORICS 94*, Springer-Verlag, 1994, pp. 144–160.
695. L. Harn and T. Kiesler, "New Scheme for Digital Multisignatures," *Electronics Letters*, v. 25, n. 15, 20 Jul 1989, pp. 1002–1003.
696. L. Harn and T. Kiesler, "Improved Rahin's Scheme with High Efficiency," *Electronics Letters*, v. 25, n. 15, 20 Jul 1989, p. 1016.
697. L. Harn and T. Kiesler, "Two New Efficient Cryptosystems Based on Rabin's Scheme," *Fifth Annual Computer Security Applications Conference*, IEEE Computer Society Press, 1990, pp. 263–270.
698. L. Harn and D.-C. Wang, "Cryptanalysis and Modification of Digital Signature Scheme Based on Error-Correcting Codes," *Electronics Letters*, v. 28, n. 2, 10 Jan 1992, p. 157–159.

699. L. Harn and Y. Xu, "Design of Generalized ElGamal Type Digital Signature Schemes Based on Discrete Logarithm," *Electronics Letters*, v. 30, n. 24, 24 Nov 1994, p. 2025–2026.

700. L. Harn and S. Yang, "Group-Oriented Undeniable Signature Schemes without the Assistance of a Mutually Trusted Party," *Advances in Cryptology—AUSCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 133–142.

701. G. Harper, A. Menezes, and S. Vanstone, "Public-Key Cryptosystems with Very Small Key Lengths," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 163–173.

702. C. Harpes, "Notes on High Order Differential Cryptanalysis of DES," internal report, Signal and Information Processing Laboratory, Swiss Federal Institute of Technology, Aug 1993.

703. G.W. Hart, "To Decode Short Cryptograms," *Communications by the ACM*, v. 37, n. 9, Sep 1994, pp. 102–108.

704. J. Hastad, "On Using RSA with Low Exponent in a Public Key Network," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 403–408.

705. J. Hastad and A. Shamir, "The Cryptographic Security of Truncated Linearly Related Variables," *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing*, 1985, pp. 356–362.

706. R.C. Hauser and E.S. Lee, "Verification and Modelling of Authentication Protocols," *ESORICS 92, Proceedings of the Second European Symposium on Research in Computer Security*, Springer-Verlag, 1992, pp. 131–154.

707. B. Hayes, "Anonymous One-Time Signatures and Flexible Untraceable Electronic Cash," *Advances in Cryptology—AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 294–305.

708. D.K. He, "LUC Public Key Cryptosystem and its Properties," *CHINACRYPT '94*, Xidian, China, 11–15 Nov 1994, pp. 60–69. (In Chinese.)

709. J. He and T. Kiesler, "Enhancing the Security of ElGamal's Signature Scheme," *IEE Proceedings on Computers and Digital Techniques*, v. 141, n. 3, 1994, pp. 193–195.

710. E.H. Hebern, "Electronic Coding Machine," U.S. Patent #1,510,441, 30 Sep 1924.

711. N. Heintze and J.D. Tygar, "A Model for Secure Protocols and their Compositions," *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, 1994, pp. 2–13.

712. M.E. Hellman, "An Extension of the Shannon Theory Approach to Cryptography," *IEEE Transactions on Information Theory*, v. IT-23, n. 3, May 1977, pp. 289–294.

713. M.E. Hellman, "The Mathematics of Public-Key Cryptography," *Scientific American*, v. 241, n. 8, Aug 1979, pp. 146–157.

714. M.E. Hellman, "DES Will Be Totally Insecure within Ten Years," *IEEE Spectrum*, v. 16, n. 7, Jul 1979, pp. 32–39.

715. M.E. Hellman, "On DES-Based Synchronous Encryption," Dept. of Electrical Engineering, Stanford University, 1980.

716. M.E. Hellman, "A Cryptanalytic Time-Memory Trade Off," *IEEE Transactions on Information Theory*, v. 26, n. 4, Jul 1980, pp. 401–406.

717. M.E. Hellman, "Another Cryptanalytic Attack on 'A Cryptosystem for Multiple Communications'," *Information Processing Letters*, v. 12, 1981, pp. 182–183.

718. M.E. Hellman, W. Diffie, and R.C. Merkle, "Cryptographic Apparatus and Method," U.S. Patent #4,200,770, 29 Apr 1980.

719. M.E. Hellman, W. Diffie, and R.C. Merkle, "Cryptographic Apparatus and Method," Canada Patent #1,121,480, 6 Apr 1982.

720. M.E. Hellman and R.C. Merkle, "Public Key Cryptographic Apparatus and Method," U.S. Patent #4,218,582, 19 Aug 1980.

721. M.E. Hellman, R. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig, and P. Schweitzer, "Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard," Technical Report SEL 76-042, Information Systems Lab, Department of Electrical Engineering, Stanford University, 1976.

722. M.E. Hellman and S.C. Pohlig, "Exponentiation Cryptographic Apparatus and Method," U.S. Patent #4,424,414, 3 Jan 1984.

723. M.E. Hellman and J.M. Reyneri, "Distribution of Drainage in the DES," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 129–131.

724. F. Hendessi and M.R. Aref, "A Successful Attack Against the DES," *Third Canadian*

Workshop on Information Theory and Applications, Springer-Verlag, 1994, pp. 78–90.

725. T. Herlestam, "Critical Remarks on Some Public-Key Cryptosystems," *BIT*, v. 18, 1978, pp. 493–496.
726. T. Herlestam, "On Functions of Linear Shift Register Sequences", *Advances in Cryptology—EUROCRYPT '85*, Springer-Verlag, 1986, pp. 119–129.
727. T. Herlestam and R. Johannesson, "On Computing Logarithms over GF(2_p)," *BIT*, v. 21, 1981, pp. 326–334.
728. H.M. Heys and S.E. Tavares, "On the Security of the CAST Encryption Algorithm," *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, Halifax, Nova Scotia, Sep 1994, pp. 332–335.
729. H.M. Heys and S.E. Tavares, "The Design of Substitution-Permutation Networks Resistant to Differential and Linear Cryptanalysis," *Proceedings of the 2nd Annual ACM Conference on Computer and Communications Security*, ACM Press, 1994, pp. 148–155.
730. E. Heyst and T.P. Pederson, "How to Make Fail-Stop Signatures," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 366–377.
731. E. Heyst, T.P. Pederson, and B. Pfitzmann, "New Construction of Fail-Stop Signatures and Lower Bounds," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 15–30.
732. L.S. Hill, "Cryptography in an Algebraic Alphabet," *American Mathematical Monthly*, v. 36, Jun–Jul 1929, pp. 306–312.
733. P.J.M. Hin, "Channel-Error-Correcting Privacy Cryptosystems," Ph.D. dissertation, Delft University of Technology, 1986. (In Dutch.)
734. R. Hirschfeld, "Making Electronic Refunds Safer," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 106–112.
735. A. Hodges, *Alan Turing: The Enigma of Intelligence*, Simon and Schuster, 1983.
736. W. Hohl, X. Lai, T. Meier, and C. Waldvogel, "Security of Iterated Hash Functions Based on Block Ciphers," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 379–390.
737. F. Hoornaert, M. Decroos, J. Vandewalle, and R. Govaerts, "Fast RSA-Hardware: Dream or Reality?" *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 257–264.
738. F. Hoornaert, J. Goubert, and Y. Desmedt, "Efficient Hardware Implementation of the DES," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 147–173.
739. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Rockville, MD: Computer Science Press, 1978.
740. P. Horster, H. Petersen, and M. Michels, "Meta-ElGamal Signature Schemes," *Proceedings of the 2nd Annual ACM Conference on Computer and Communications Security*, ACM Press, 1994, pp. 96–107.
741. P. Horster, H. Petersen, and M. Michels, "Meta Message Recovery and Meta Blind Signature Schemes Based on the Discrete Logarithm Problem and their Applications," *Advances in Cryptology—ASIACRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 224–237.
742. L.K. Hua, *Introduction to Number Theory*, Springer-Verlag, 1982.
743. K. Hubr, "Specialized Attack on Chor-Rivest Public Key Cryptosystem," *Electronics Letters*, v. 27, n. 23, 7 Nov 1991, pp. 2130–2131.
744. E. Hughes, "A Cypherpunk's Manifesto," 9 Mar 1993.
745. E. Hughes, "An Encrypted Key Transmission Protocol," presented at the rump session of CRYPTO '94, Aug 1994.
746. H. Hule and W.B. Müller, "On the RSA-Cryptosystem with Wrong Keys," *Contributions to General Algebra 6*, Vienna: Verlag Hölder-Pichler-Tempsky, 1988, pp. 103–109.
747. H.A. Hussain, J.W.A. Sada, and S.M. Kalipha, "New Multistage Knapsack Public-Key Cryptosystem," *International Journal of Systems Science*, v. 22, n. 11, Nov 1991, pp. 2313–2320.
748. T. Hwang, "Attacks on Okamoto and Tanaka's One-Way ID-Based Key Distribution System," *Information Processing Letters*, v. 43, n. 2, Aug 1992, pp. 83–86.
749. T. Hwang and T.R.N. Rao, "Secret Error-Correcting Codes (SECC)," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 540–563.
750. C. I'Anson and C. Mitchell, "Security Defects in CCITT Recommendation

X.509—the Directory Authentication Framework," *Computer Communications Review*, v. 20, n. 2, Apr 1990, pp. 30–34.

751. IBM, "Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference," SC40-1675-1, IBM Corp., Nov 1990.
752. IBM, "Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference—Public Key Algorithm," IBM Corp., Mar 1993.
753. R. Impagliazzo and M. Yung, "Direct Minimum-Knowledge Computations," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 40–51.
754. I. Ingemarsson, "A New Algorithm for the Solution of the Knapsack Problem," *Lecture Notes in Computer Science 149; Cryptography: Proceedings of the Workshop on Cryptography*, Springer-Verlag, 1983, pp. 309–315.
755. I. Ingemarsson, "Delay Estimation for Truly Random Binary Sequences or How to Measure the Length of Rip van Winkle's Sleep," *Communications and Cryptography: Two Sides of One Tapestry*, R.E. Blahut et al., eds., Kluwer Adademic Publishers, 1994, pp. 179–186.
756. I. Ingemarsson and G.J. Simmons, "A Protocol to Set Up Shared Secret Schemes without the Assistance of a Mutually Trusted Party," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 266–282.
757. I. Ingemarsson, D.T. Tang, and C.K. Wong, "A Conference Key Distribution System," *IEEE Transactions on Information Theory*, v. IT-28, n. 5, Sep 1982, pp. 714–720.
758. ISO DIS 8730, "Banking—Requirements for Message Authentication (Wholesale)," Association for Payment Clearing Services, London, Jul 1987.
759. ISO DIS 8731-1, "Banking—Approved Algorithms for Message Authentication—Part 1: DEA," Association for Payment Clearing Services, London, 1987.
760. ISO DIS 8731-2, "Banking—Approved Algorithms for Message Authentication—Part 2: Message Authenticator Algorithm," Association for Payment Clearing Services, London, 1987.
761. ISO DIS 8732, "Banking—Key Management (Wholesale)," Association for Payment Clearing Services, London, Dec 1987.
762. ISO/IEC 9796, "Information Technology—Security Techniques—Digital Signature Scheme Giving Message Recovery," International Organization for Standardization, Jul 1991.
763. ISO/IEC 9797, "Data Cryptographic Techniques—Data Integrity Mechanism Using a Cryptographic Check Function Employing a Block Cipher Algorithm," International Organization for Standardization, 1989.
764. ISO DIS 10118 DRAFT, "Information Technology—Security Techniques—Hash Functions," International Organization for Standardization, 1989.
765. ISO DIS 10118 DRAFT, "Information Technology—Security Techniques—Hash Functions," International Organization for Standardization, April 1991.
766. ISO N98, "Hash Functions Using a Pseudo Random Algorithm," working document, ISO-IEC/JTC1/SC27/WG2, International Organization for Standardization, 1992.
767. ISO N179, "AR Fingerprint Function," working document, ISO-IEC/JTC1/SC27/WG2, International Organization for Standardization, 1992.
768. ISO/IEC 10118, "Information Technology—Security Techniques—Hash Functions—Part 1: General and Part 2: Hash-Functions Using an n-Bit Block Cipher Algorithm," International Organization for Standardization, 1993.
769. K. Ito, S. Kondo, and Y. Mitsuoka, "SXAL8/MBAL Algorithm," Technical Report, ISEC93-68, IEICE Japan, 1993. (In Japanese.)
770. K.R. Iversen, "The Application of Cryptographic Zero-Knowledge Techniques in Computerized Secret Ballot Election Schemes," Ph.D. dissertation, IDT-report 1991:3, Norwegian Institute of Technology, Feb 1991.
771. K.R. Iversen, "A Cryptographic Scheme for Computerized General Elections," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 405–419.
772. K. Iwamura, T. Matsumoto, and H. Imai, "An Implementation Method for RSA Cryptosystem with Parallel Processing," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. J75-A, n. 8, Aug 1992, pp. 1301–1311.

773. W.J. Jaburek, "A Generalization of ElGamal's Public Key Cryptosystem," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, 1990, Springer-Verlag, pp. 23–28.

774. N.S. James, R. Lidi, and H. Niederreiter, "Breaking the Cade Cipher," *Advances in Cryptology—CRYPTO '86 Proceedings*, 1987, Springer-Verlag, pp. 60–63.

775. C.J.A. Jansen, "On the Key Storage Requirements for Secure Terminals," *Computers and Security*, v. 5, n. 2, Jun 1986, pp. 145–149.

776. C.J.A. Jansen, "Investigations on Nonlinear Streamcipher Systems: Construction and Evaluation Methods," Ph.D. dissertation, Technical University of Delft, 1989.

777. C.J.A. Jansen and D.E. Boekee, "Modes of Blockcipher Algorithms and their Protection against Active Eavesdropping," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 281–286.

778. S.M. Jennings, "A Special Class of Binary Sequences," Ph.D. dissertation, University of London, 1980.

779. S.M. Jennings, "Multiplexed Sequences: Some Properties of the Minimum Polynomial," *Lecture Notes in Computer Science 149; Cryptography: Proceedings of the Workshop on Cryptography*, Springer-Verlag, 1983, pp. 189–206.

780. S.M. Jennings, "Autocorrelation Function of the Multiplexed Sequence," *IEE Proceedings*, v. 131, n. 2, Apr 1984, pp. 169–172.

781. T. Jin, "Care and Feeding of Your Three-Headed Dog," Document Number IAG-90-011, Hewlett-Packard, May 1990.

782. T. Jin, "Living with Your Three-Headed Dog," Document Number IAG-90-012, Hewlett-Packard, May 1990.

783. A. Jiwa, J. Seberry, and Y. Zheng, "Beacon Based Authentication," *Computer Security—ESORICS 94*, Springer-Verlag, 1994, pp. 125–141.

784. D.B. Johnson, G.M. Dolan, M.J. Kelly, A.V. Le, and S.M. Matyas, "Common Cryptographic Architecture Cryptographic Application Programming Interface," *IBM Systems Journal*, v. 30, n. 2, 1991, pp. 130–150.

785. D.B. Johnson, S.M. Matyas, A.V. Le, and J.D. Wilkins, "Design of the Commercial Data Masking Facility Data Privacy Algorithm," *1st ACM Conference on Computer and Communications Security*, ACM Press, 1993, pp. 93–96.

786. J.P. Jordan, "A Variant of a Public-Key Cryptosystem Based on Goppa Codes," *Sigact News*, v. 15, n. 1, 1983, pp. 61–66.

787. A. Joux and L. Granboulan, "A Practical Attack Against Knapsack Based Hash Functions," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.

788. A. Joux and J. Stern, "Cryptanalysis of Another Knapsack Cryptosystem," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1993, pp. 470–476.

789. R.R. Jueneman, "Analysis of Certain Aspects of Output-Feedback Mode," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 99–127.

790. R.R. Jueneman, "Electronic Document Authentication," *IEEE Network Magazine*, v. 1, n. 2, Apr 1978, pp. 17–23.

791. R.R. Jueneman, "A High Speed Manipulation Detection Code," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 327–346.

792. R.R. Jueneman, S.M. Matyas, and C.H. Meyer, "Message Authentication with Manipulation Detection Codes," *Proceedings of the 1983 IEEE Computer Society Symposium on Research in Security and Privacy*, 1983, pp. 733–54.

793. R.R. Jueneman, S.M. Matyas, and C.H. Meyer, "Message Authentication," *IEEE Communications Magazine*, v. 23, n. 9, Sep 1985, pp. 29–40.

794. D. Kahn, *The Codebreakers: The Story of Secret Writing*, New York: Macmillan Publishing Co., 1967.

795. D. Kahn, *Kahn on Codes*, New York: Macmillan Publishing Co., 1983.

796. D. Kahn, *Seizing the Enigma*, Boston: Houghton Mifflin Co., 1991.

797. P. Kaijser, T. Parker, and D. Pinkas, "SESAME: The Solution to Security for Open Distributed Systems," *Journal of Computer Communications*, v. 17, n. 4, Jul 1994, pp. 501–518.

798. R. Kailar and V.D. Gilgor, "On Belief Evolution in Authentication Protocols," *Proceedings of the Computer Security Foundations Workshop IV*, IEEE Computer Society Press, 1991, pp. 102–116.

799. B.S. Kaliski, "A Pseudo Random Bit Generator Based on Elliptic Logarithms," Master's thesis, Massachusetts Institute of Technology, 1987.

800. B.S. Kaliski, letter to NIST regarding DSS, 4 Nov 1991.

801. B.S. Kaliski, "The MD2 Message Digest Algorithm," RFC 1319, Apr 1992.

802. B.S. Kaliski, "Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certificates and Related Services," RFC 1424, Feb 1993.

803. B.S. Kaliski, "An Overview of the PKCS Standards," RSA Laboratories, Nov 1993.

804. B.S. Kaliski, "A Survey of Encryption Standards, *IEEE Micro*, v. 13, n. 6, Dec 1993, pp. 74–81.

805. B.S. Kaliski, personal communication, 1993.

806. B.S. Kaliski, "On the Security and Performance of Several Triple-DES Modes," RSA Laboratories, draft manuscript, Jan 1994.

807. B.S. Kaliski, R.L. Rivest, and A.T. Sherman, "Is the Data Encryption Standard a Group?", *Advances in Cryptology—EUROCRYPT '85*, Springer-Verlag, 1986, pp. 81–95.

808. B.S. Kaliski, R.L. Rivest, and A.T. Sherman, "Is the Data Encryption Standard a Pure Cipher? (Results of More Cycling Experiments in DES)," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 212–226.

809. B.S. Kaliski, R.L. Rivest, and A.T. Sherman, "Is the Data Encryption Standard a Group? (Results of Cycling Experiments on DES)," *Journal of Cryptology*, v. 1, n. 1, 1988, pp. 3–36.

810. B.S. Kaliski and M.J.B. Robshaw, "Fast Block Cipher Proposal," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 33–40.

811. B.S. Kaliski and M.J.B. Robshaw, "Linear Cryptanalysis Using Multiple Approximations," *Advances in Cryptology—CRYPTO '94 Proceedings*, Springer-Verlag, 1994, pp. 26–39.

812. B.S. Kaliski and M.J.B. Robshaw, "Linear Cryptanalysis Using Multiple Approximations and FEAL," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.

813. R.G. Kammer, statement before the U.S. government Subcommittee on Telecommunications and Finance, Committee on Energy and Commerce, 29 Apr 1993.

814. T. Kaneko, K. Koyama, and R. Terada, "Dynamic Swapping Schemes and Differential Cryptanalysis, *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 292–301.

815. T. Kaneko, K. Koyama, and R. Terada, "Dynamic Swapping Schemes and Differential Cryptanalysis," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. E77-A, n. 8, Aug 1994, pp. 1328–1336.

816. T. Kaneko and H. Miyano, "A Study on the Strength Evaluation of Randomized DES-Like Cryptosystems against Chosen Plaintext Attacks," *Proceedings of the 1993 Symposium on Cryptography and Information Security (SCIS 93)*, Shuzenji, Japan, 28–30 Jan 1993, pp. 15C.1–10.

817. J. Kari, "A Cryptosystem Based on Propositional Logic," *Machines, Languages, and Complexity: 5th International Meeting of Young Computer Scientists, Selected Contributions*, Springer-Verlag, 1989, pp. 210–219.

818. E.D. Karnin, J.W. Greene, and M.E. Hellman, "On Sharing Secret Systems," *IEEE Transactions on Information Theory*, v. IT-29, 1983, pp. 35–41.

819. F.W. Kasiski, *Die Geheimschriften und die Dechiffrir-kunst*, E.S. Miller und Sohn, 1863. (In German.)

820. A. Kehne, J. Schonwalder, and H. Langendorfer, "A Nonce-Based Protocol for Multiple Authentications," *Operating Systems Review*, v. 26, n. 4, Oct 1992, pp. 84–89.

821. J. Kelsey, personal communication, 1994.

822. R. Kemmerer, "Analyzing Encryption Protocols Using Formal Verification Techniques," *IEEE Journal on Selected Areas in Communications*, v. 7, n. 4, May 1989, pp. 448–457.

823. R. Kemmerer, C.A. Meadows, and J. Millen, "Three Systems for Cryptographic Protocol Analysis," *Journal of Cryptology*, v. 7, n. 2, 1994, pp. 79–130.

824. S.T. Kent, "Encryption-Based Protection Protocols for Interactive User-Computer Communications," MIT/LCS/TR-162,

MIT Laboratory for Computer Science, May 1976.

825. S.T. Kent, "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management," RFC 1422, Feb 1993.

826. S.T. Kent, "Understanding the Internet Certification System," *Proceedings of INET '93*, The Internet Society, 1993, pp. BAB1-BAB10.

827. S.T. Kent and J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management," RFC 1114, Aug 1989.

828. V. Kessler and G. Wedel, "AUTOLOG—An Advanced Logic of Authentication," *Proceedings of the Computer Security Foundations Workshop VII*, IEEE Computer Society Press, 1994, pp. 90–99.

829. E.L. Key, "An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators," *IEEE Transactions on Information Theory*, v. IT-22, n. 6, Nov 1976, pp. 732–736.

830. T. Kiesler and L. Harn, "RSA Blocking and Multisignature Schemes with No Bit Expansion," *Electronics Letters*, v. 26, n. 18, 30 Aug 1990, pp. 1490–1491.

831. J. Kilian, *Uses of Randomness in Algorithms and Protocols*, MIT Press, 1990.

832. J. Kilian, "Achieving Zero-Knowledge Robustly," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 313–325.

833. J. Kilian and T. Leighton, "Failsafe Key Escrow," MIT/LCS/TR-636, MIT Laboratory for Computer Science, Aug 1994.

834. K. Kim, "Construction of DES-Like S-Boxes Based on Boolean Functions Satisfying the SAC," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1993, pp. 59–72.

835. K. Kim, S. Lee, and S. Park, "Necessary Conditions to Strengthen DES S-Boxes Against Linear Cryptanalysis," *Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94)*, Lake Biwa, Japan, 27–29 Jan 1994, pp. 15D.1–9.

836. K. Kim, S. Lee, and S. Park, "How to Strengthen DES against Differential Attack," unpublished manuscript, 1994.

837. K. Kim, S. Lee, S. Park, and D. Lee, "DES Can Be Immune to Differential Cryptanalysis," *Workshop on Selected Areas in Cryptography—Workshop Record*, Kingston, Ontario, 5–6 May 1994, pp. 70–81.

838. K. Kim, S. Park, and S. Lee, "How to Strengthen DES against Two Robust Attacks," *Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography*, Inuyama, Japan, 24–27 Jan 1995, 173–182.

839. K. Kim, S. Park, and S. Lee, "Reconstruction of s^2DES S-Boxes and their Immunity to Differential Cryptanalysis," *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 282–291.

840. S. Kim and B.S. Um, "A Multipurpose Membership Proof System Based on Discrete Logarithm," *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 177–183.

841. P. Kinnucan, "Data Encryption Gurus: Tuchman and Meyer," *Cryptologia*, v. 2, n. 4, Oct 1978.

842. A. Klapper, "The Vulnerability of Geometric Sequences Based on Fields of Odd Characteristic," *Journal of Cryptology*, v. 7, n. 1, 1994, pp. 33–52.

843. A. Klapper, "Feedback with Carry Shift Registers over Finite Fields," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.

844. A. Klapper and M. Goresky, "2-adic Shift Registers," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 174–178.

845. A. Klapper and M. Goresky, "2-adic Shift Registers," Technical Report #239-93, Department of Computer Science, University of Kentucky, 19 Apr 1994.

846. A. Klapper and M. Goresky, "Large Period Nearly de Bruijn FCSR Sequences," *Advances in Cryptology—EUROCRYPT '95 Proceedings*, Springer-Verlag, 1995, pp. 263–273.

847. D.V. Klein, " 'Foiling the Cracker': A Survey of, and Implications to, Password Security," *Proceedings of the USENIX UNIX Security Workshop*, Aug 1990, pp. 5–14.

848. D.V. Klein, personal communication, 1994.

849. C.S. Kline and G.J. Popek, "Public Key vs. Conventional Key Cryptosystems," *Pro-*

ceedings of AFIPS National Computer Conference, pp. 831–837.

850. H.-J. Knobloch, "A Smart Card Implementation of the Fiat-Shamir Identification Scheme," *Advances in Cryptology—EUROCRPYT '88 Proceedings*, Springer-Verlag, 1988, pp. 87–95.
851. T. Knoph, J. Fröβl, W. Beller, and T. Giesler, "A Hardware Implementation of a Modified DES Algorithm," *Microprocessing and Microprogramming*, v. 30, 1990, pp. 59–66.
852. L.R. Knudsen, "Cryptanalysis of LOKI," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1993, pp. 22–35.
853. L.R. Knudsen, "Cryptanalysis of LOKI," *Cryptography and Coding III*, M.J. Ganley, ed., Oxford: Clarendon Press, 1993, pp. 223–236.
854. L.R. Knudsen, "Cryptanalysis of LOKI91," *Advances in Cryptology—AUSCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 196–208.
855. L.R. Knudsen, "Iterative Characteristics of DES and s^2DES," *Advances in Cryptology—CRYPTO '92*, Springer-Verlag, 1993, pp. 497–511.
856. L.R. Knudsen, "An Analysis of Kim, Park, and Lee's DES-Like S-Boxes," unpublished manuscript, 1993.
857. L.R. Knudsen, "Practically Secure Feistel Ciphers," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 211–221.
858. L.R. Knudsen, "Block Ciphers—Analysis, Design, Applications," Ph.D. dissertation, Aarhus University, Nov 1994.
859. L.R. Knudsen, personal communication, 1994.
860. L.R. Knudsen, "Applications of Higher Order Differentials and Partial Differentials," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
861. L.R. Knudsen and X. Lai, "New Attacks on All Double Block Length Hash Functions of Hash Rate 1, Including the Parallel-DM," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
862. L.R. Knudsen, "A Weakness in SAFER K-64," *Advances in Cryptology–CRYPTO '95 Proceedings*, Springer-Verlag, 1995, to appear.
863. D. Knuth, *The Art of Computer Programming: Volume 2, Seminumerical Algorithms*, 2nd edition, Addison-Wesley, 1981.
864. D. Knuth, "Deciphering a Linear Congruential Encryption," *IEEE Transactions on Information Theory*, v. IT-31, n. 1, Jan 1985, pp. 49–52.
865. K. Kobayashi and L. Aoki, "On Linear Cryptanalysis of MBAL," *Proceedings of the 1995 Symposium on Cryptography and Information Security {SCIS 95}*, Inuyama, Japan, 24–27 Jan 1995, pp. A4.2.1–9.
866. K. Kobayashi, K. Tamura, and Y. Nemoto, "Two-dimensional Modified Rabin Cryptosystem," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. J72-D, n. 5, May 1989, pp. 850–851. {In Japanese.}
867. N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, v. 48, n. 177, 1987, pp. 203–209.
868. N. Koblitz, "A Family of Jacobians Suitable for Discrete Log Cryptosystems," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 94–99.
869. N. Koblitz, "Constructing Elliptic Curve Cryptosystems in Characteristic 2," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 156–167.
870. N. Koblitz, "Hyperelliptic Cryptosystems," *Journal of Cryptology*, v. 1, n. 3, 1989, pp. 129–150.
871. N. Koblitz, "CM-Curves with Good Cryptographic Properties," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 279–287.
872. Ç.K. Koç, "High-Speed RSA Implementation," Version 2.0, RSA Laboratories, Nov 1994.
873. M.J. Kochanski, "Remarks on Lu and Lee's Proposals," *Cryptologia*, v. 4, n. 4, 1980, pp. 204–207.
874. M.J. Kochanski, "Developing an RSA Chip," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 350–357.
875. J.T. Kohl, "The Use of Encryption in Kerberos for Network Authentication," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 35–43.

876. J.T. Kohl, "The Evolution of the Kerberos Authentication Service," *EurOpen Conference Proceedings*, May 1991, pp. 295–313.
877. J.T. Kohl and B.C. Neuman, "The Kerberos Network Authentication Service," RFC 1510, Sep 1993.
878. J.T. Kohl, B.C. Neuman, and T. Ts'o, "The Evolution of the Kerberos Authentication System," *Distributed Open Systems*, IEEE Computer Society Press, 1994, pp. 78–94.
879. Kohnfelder, "Toward a Practical Public Key Cryptosystem," Bachelor's thesis, MIT Department of Electrical Engineering, May 1978.
880. A.G. Konheim, *Cryptography: A Primer*, New York: John Wiley & Sons, 1981.
881. A.G. Konheim, M.H. Mack, R.K. McNeill, B. Tuckerman, and G. Waldbaum, "The IPS Cryptographic Programs," *IBM Systems Journal*, v. 19, n. 2, 1980, pp. 253–283.
882. V.I. Korzhik and A.I. Turkin, "Cryptanalysis of McEliece's Public-Key Cryptosystem," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 68–70.
883. S.C. Kothari, "Generalized Linear Threshold Scheme," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 231–241.
884. J. Kowalchuk, B.P. Schanning, and S. Powers, "Communication Privacy: Integration of Public and Secret Key Cryptography," *Proceedings of the National Telecommunication Conference*, IEEE Press, 1980, pp. 49.1.1–49.1.5.
885. K. Koyama, "A Master Key for the RSA Public-Key Cryptosystem," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. J65-D, n. 2, Feb 1982, pp. 163–170.
886. K. Koyama, "A Cryptosystem Using the Master Key for Multi-Address Communications," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. J65-D, n. 9, Sep 1982, pp. 1151–1158.
887. K. Koyama, "Demonstrating Membership of a Group Using the Shizuya-Koyama-Itoh (SKI) Protocol," *Proceedings of the 1989 Symposium on Cryptography and Information Security (SCIS 89)*, Gotenba, Japan, 1989.
888. K. Koyama, "Direct Demonstration of the Power to Break Public-Key Cryptosystems," *Advances in Cryptology—AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 14–21.
889. K. Koyama, "Security and Unique Decipherability of Two-dimensional Public Key Cryptosystems," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. E73, n. 7, Jul 1990, pp. 1057–1067.
890. K. Koyama, U.M. Maurer, T. Okamoto, and S.A. Vanstone, "New Public-Key Schemes Based on Elliptic Curves over the Ring Z_n," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 252–266.
891. K. Koyama and K. Ohta, "Identity-based Conference Key Distribution System," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 175–184.
892. K. Koyama and T. Okamoto, "Elliptic Curve Cryptosystems and Their Applications," *IEICE Transactions on Information and Systems*, v. E75-D, n. 1, Jan 1992, pp. 50–57.
893. K. Koyama and R. Terada, "How to Strengthen DES-Like Cryptosystems against Differential Cryptanalysis," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. E76-A, n. 1, Jan 1993, pp. 63–69.
894. K. Koyama and R. Terada, "Probabilistic Swapping Schemes to Strengthen DES against Differential Cryptanalysis," *Proceedings of the 1993 Symposium on Cryptography and Information Security (SCIS 93)*, Shuzenji, Japan, 28–30 Jan 1993, pp. 15D.1–12.
895. K. Koyama and Y. Tsuruoka, "Speeding up Elliptic Cryptosystems Using a Singled Binary Window Method," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 345–357.
896. E. Kranakis, *Primality and Cryptography*, Wiler-Tuebner Series in Computer Science, 1986.
897. D. Kravitz, "Digital Signature Algorithm," U.S. Patent #5,231,668, 27 Jul 1993.
898. D. Kravitz and I. Reed, "Extension of RSA Cryptostructure: A Galois Approach," *Electronics Letters*, v. 18, n. 6, 18 Mar 1982, pp. 255–256.

899. H. Krawczyk, "How to Predict Congruential Generators," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 138–153.
900. H. Krawczyk, "How to Predict Congruential Generators," *Journal of Algorithms*, v. 13, n. 4, Dec 1992, pp. 527–545.
901. H. Krawczyk, "The Shrinking Generator: Some Practical Considerations," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 45–46.
902. G.J. Kühn, "Algorithms for Self-Synchronizing Ciphers," *Proceedings of COMSIG 88*, 1988.
903. G.J. Kühn, F. Bruwer, and W. Smit, "'n Vinnige Veeldoelige Enkripsievlokkie," *Proceedings of Infosec 90*, 1990. (In Afrikaans.)
904. S. Kullback, *Statistical Methods in Cryptanalysis*, U.S. Government Printing Office, 1935. Reprinted by Aegean Park Press, 1976.
905. P.V. Kumar, R.A. Scholtz, and L.R. Welch, "Generalized Bent Functions and their Properties," *Journal of Combinational Theory*, Series A, v. 40, n. 1, Sep 1985, pp. 90–107.
906. M. Kurosaki, T. Matsumoto, and H. Imai, "Simple Methods for Multipurpose Certification," *Proceedings of the 1989 Symposium on Cryptography and Information Security (SCIS 89)*, Gotenba, Japan, 1989.
907. M. Kurosaki, T. Matsumoto, and H. Imai, "Proving that You Belong to at Least One of the Specified Groups," *Proceedings of the 1990 Symposium on Cryptography and Information Security (SCIS 90)*, Hihondaira, Japan, 1990.
908. K. Kurosawa, "Key Changeable ID-Based Cryptosystem," *Electronics Letters*, v. 25, n. 9, 27 Apr 1989, pp. 577–578.
909. K. Kurosawa, T. Ito, and M. Takeuchi, "Puhlic Key Cryptosystem Using a Reciprocal Number with the Same Intractability as Factoring a Large Number," *Cryptologia*, v. 12, n. 4, Oct 1988, pp. 225–233.
910. K. Kurosawa, C. Park, and K. Sakano, "Group Signer/Verifier Separation Scheme," *Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography*, Inuyama, Japan, 24–27 Jan 1995, 134–143.
911. G.C. Kurtz, D. Shanks, and H.C. Williams, "Fast Primality Tests for Numbers Less than $50*10^9$," *Mathematics of Computation*, v. 46, n. 174, Apr 1986, pp. 691–701.
912. K. Kusuda and T. Matsumoto, "Optimization of the Time-Memory Trade-Off Cryptanalysis and Its Application to Block Ciphers," *Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS 95)*, Inuyama, Japan, 24–27 Jan 1995, pp. A3.2.1–11. (In Japanese.)
913. H. Kuwakado and K. Koyama, "Security of RSA-Type Cryptosystems Over Elliptic Curves against Hastad Attack," *Electronics Letters*, v. 30, n. 22, 27 Oct 1994, pp. 1843–1844.
914. H. Kuwakado and K. Koyama, "A New RSA-Type Cryptosystem over Singular Elliptic Curves," *IMA Conference on Applications of Finite Fields*, Oxford University Press, to appear.
915. H. Kuwakado and K. Koyama, "A New RSA-Type Scheme Based on Singular Cubic Curves," *Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography*, Inuyama, Japan, 24–27 Jan 1995, pp. 144–151.
916. M. Kwan, "An Eight Bit Weakness in the LOKI Cryptosystem," technical report, Australian Defense Force Academy, Apr 1991.
917. M. Kwan and J. Pieprzyk, "A General Purpose Technique for Locating Key Scheduling Weakness in DES-Like Cryptosystems," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 237–246.
918. J.B. Lacy, D.P. Mitchell, and W.M. Schell, "CryptoLib: Cryptography in Software," *UNIX Security Symposium IV Proceedings*, USENIX Association, 1993, pp. 1–17.
919. J.C. Lagarias, "Knapsack Public Key Cryptosystems and Diophantine Approximations," *Advances in Cryptology: Proceedings of Crypto 83*, Plenum Press, 1984, pp. 3–23.
920. J.C. Lagarias, "Performance Analysis of Shamir's Attack on the Basic Merkle-Hellman Knapsack Cryptosystem," *Lecture Notes in Computer Science 172; Proceedings of the 11th International Colloquium on Automata, Languages, and Programming (ICALP)*, Springer-Verlag, 1984, pp. 312–323.
921. J.C. Lagarias and A.M. Odlyzko, "Solving Low-Density Subset Sum Problems," *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, 1983, pp. 1–10.

922. J.C. Lagarias and A.M. Odlyzko, "Solving Low-Density Subset Sum Problems," *Journal of the ACM*, v. 32, n. 1, Jan 1985, pp. 229–246.

923. J.C. Lagarias and J. Reeds, "Unique Extrapolation of Polynomial Recurrences," *SIAM Journal on Computing*, v. 17, n. 2, Apr 1988, pp. 342–362.

924. X. Lai, *Detailed Description and a Software Implementation of the IPES Cipher*, unpublished manuscript, 8 Nov 1991.

925. X. Lai, *On the Design and Security of Block Ciphers*, ETH Series in Information Processing, v. 1, Konstanz: Hartung-Gorre Verlag, 1992.

926. X. Lai, personal communication, 1993.

927. X. Lai, "Higher Order Derivatives and Differential Cryptanalysis," *Communications and Cryptography: Two Sides of One Tapestry*, R.E. Blahut et al., eds., Kluwer Adademic Publishers, 1994, pp. 227–233.

928. X. Lai and L. Knudsen, "Attacks on Double Block Length Hash Functions," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 157–165.

929. X. Lai and J. Massey, "A Proposal for a New Block Encryption Standard," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 389–404.

930. X. Lai and J. Massey, "Hash Functions Based on Block Ciphers," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1992, pp. 55–70.

931. X. Lai, J. Massey, and S. Murphy, "Markov Ciphers and Differential Cryptanalysis," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 17–38.

932. X. Lai, R.A. Rueppel, and J. Woollven, "A Fast Cryptographic Checksum Algorithm Based on Stream Ciphers," *Advances in Cryptology—AUSCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 339–348.

933. C.S. Laih, J.Y. Lee, C.H. Chen, and L. Harn, "A New Scheme for ID-based Cryptosystems and Signatures," *Journal of the Chinese Institute of Engineers*, v. 15, n. 2, Sep 1992, pp. 605–610.

934. B.A. LaMacchia and A.M. Odlyzko, "Computation of Discrete Logarithms in Prime Fields," *Designs, Codes, and Cryptography*, v. 1, 1991, pp. 46–62.

935. L. Lamport, "Password Identification with Insecure Communications," *Communications of the ACM*, v. 24, n. 11, Nov 1981, pp. 770–772.

936. S. Landau, "Zero-Knowledge and the Department of Defense," *Notices of the American Mathematical Society*, v. 35, n. 1, Jan 1988, pp. 5–12.

937. S. Landau, S. Kent, C. Brooks, S. Charney, D. Denning, W. Diffie, A. Lauck, D. Mikker, P. Neumann, and D. Sobel, "Codes, Keys, and Conflicts: Issues in U.S. Crypto Policy," Report of a Special Panel of the ACM U.S. Public Policy Committee (USACM), Association for Computing Machinery, Jun 1994.

938. S.K. Langford and M.E. Hellman, "Cryptanalysis of DES," presented at 1994 RSA Data Security conference, Redwood Shores, CA, 12–14 Jan 1994.

939. D. Lapidot and A. Shamir, "Publicly Verifiable Non-Interactive Zero-Knowledge Proofs," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 353–365.

940. A.V. Le, S.M. Matyas, D.B. Johnson, and J.D. Wilkins, "A Public-Key Extension to the Common Cryptographic Architecture," *IBM Systems Journal*, v. 32, n. 3, 1993, pp. 461–485.

941. P. L'Ecuyer, "Efficient and Portable Combined Random Number Generators," *Communications of the ACM*, v. 31, n. 6, Jun 1988, pp. 742–749, 774.

942. P. L'Ecuyer, "Random Numbers for Simulation," *Communications of the ACM*, v. 33, n. 10, Oct 1990, pp. 85–97.

943. P.J. Lee and E.F. Brickell, "An Observation on the Security of McEliece's Public-Key Cryptosystem," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 275–280.

944. S. Lee, S. Sung, and K. Kim, "An Efficient Method to Find the Linear Expressions for Linear Cryptanalysis," *Proceedings of the 1995 Korea-Japan Workshop on Information Security and Cryptography*, Inuyama, Japan, 24–26 Jan 1995, pp. 183–190.

945. D.J. Lehmann, "On Primality Tests," *SIAM Journal on Computing*, v. 11, n. 2, May 1982, pp. 374–375.

946. T. Leighton, "Failsafe Key Escrow Systems," Technical Memo 483, MIT Laboratory for Computer Science, Aug 1994.

947. A. Lempel and M. Cohn, "Maximal Families of Bent Sequences," *IEEE Transactions on Information Theory*, v. IT-28, n. 6, Nov 1982, pp. 865–868.
948. A.K. Lenstra, "Factoring Multivariate Polynomials Over Finite Fields," *Journal of Computer System Science*, v. 30, n. 2, Apr 1985, pp. 235–248.
949. A.K. Lenstra, personal communication, 1995.
950. A.K. Lenstra and S. Haber, letter to NIST Regarding DSS, 26 Nov 1991.
951. A.K. Lenstra, H.W. Lenstra Jr., and L. Lovácz, "Factoring Polynomials with Rational Coefficients," *Mathematische Annalen*, v. 261, n. 4, 1982, pp. 515–534.
952. A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, and J.M. Pollard, "The Number Field Sieve," *Proceedings of the 22nd ACM Symposium on the Theory of Computing*, 1990, pp. 574–572.
953. A.K. Lenstra and H.W. Lenstra, Jr., eds., *Lecture Notes in Mathematics 1554: The Development of the Number Field Sieve*, Springer-Verlag, 1993.
954. A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, and J.M. Pollard, "The Factorization of the Ninth Fermat Number," *Mathematics of Computation*, v. 61, n. 203, 1993, pp. 319–349.
955. A.K. Lenstra and M.S. Manasse, "Factoring by Electronic Mail," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 355–371.
956. A.K. Lenstra and M.S. Manasse, "Factoring with Two Large Primes," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 72–82.
957. H.W. Lenstra Jr. "Elliptic Curves and Number-Theoretic Algorithms," Report 86-19, Mathematisch Instituut, Universiteit van Amsterdam, 1986.
958. H.W. Lenstra Jr. "On the Chor-Rivest Knapsack Cryptosystem," *Journal of Cryptology*, v. 3, n. 3, 1991, pp. 149–155.
959. W.J. LeVeque, *Fundamentals of Number Theory*, Addison-Wesley, 1977.
960. L.A. Levin, "One-Way Functions and Pseudo-Random Generators," *Proceedings of the 17th ACM Symposium on Theory of Computing*, 1985, pp. 363–365.
961. Lexar Corporation, "An Evaluation of the DES," Sep 1976.
962. D.-X. Li, "Cryptanalysis of Public-Key Distribution Systems Based on Dickson Polynomials," *Electronics Letters*, v. 27, n. 3, 1991, pp. 228–229.
963. F.-X. Li, "How to Break Okamoto's Cryptosystems by Continued Fraction Algorithm," *ASIACRYPT '91 Abstracts*, 1991, pp. 285–289.
964. Y.X. Li and X.M. Wang, "A Joint Authentication and Encryption Scheme Based on Algebraic Coding Theory," *Applied Algebra, Algebraic Algorithms and Error Correcting Codes 9*, Springer-Verlag, 1991, pp. 241–245.
965. R. Lidl, G.L. Mullen, and G. Turwald, *Pitman Monographs and Surveys in Pure and Applied Mathematics 65: Dickson Polynomials*, London: Longman Scientific and Technical, 1993.
966. R. Lidl and W.B. Müller, "Permutation Polynomials in RSA-Cryptosystems," *Advances in Cryptology: Proceedings of Crypto 83*, Plenum Press, 1984, pp. 293–301.
967. R. Lidl and W.B. Müller, "Generalizations of the Fibonacci Pseudoprimes Test," *Discrete Mathematics*, v. 92, 1991, pp. 211–220.
968. R. Lidl and W.B. Müller, "Primality Testing with Lucas Functions," *Advances in Cryptology—AUSCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 539–542.
969. R. Lidl, W.B. Müller, and A. Oswald, "Some Remarks on Strong Fibonacci Pseudoprimes," *Applicable Algebra in Engineering, Communication and Computing*, v. 1, n. 1, 1990, pp. 59–65.
970. R. Lidl and H. Niederreiter, "Finite Fields," *Encyclopedia of Mathematics and its Applications*, v. 20, Addison-Wesley, 1983.
971. R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, London: Cambridge University Press, 1986.
972. K. Lieberherr, "Uniform Complexity and Digital Signatures," *Theoretical Computer Science*, v. 16, n. 1, Oct 1981, pp. 99–110.
973. C.H. Lim and P.J. Lee, "A Practical Electronic Cash System for Smart Cards," *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 34–47.
974. C.H. Lim and P.J. Lee, "Security of Interactive DSA Batch Verification," *Electronics Letters*, v. 30, n. 19, 15 Sep 1994, pp. 1592–1593.

975. H.-Y. Lin and L. Harn, "A Generalized Secret Sharing Scheme with Cheater Detection," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1993, pp. 149–158.

976. M.-C. Lin, T.-C. Chang, and H.-L. Fu, "Information Rate of McEliece's Public-key Cryptosystem," *Electronics Letters*, v. 26, n. 1, 4 Jan 1990, pp. 16–18.

977. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I—Message Encipherment and Authentication Procedures," RFC 989, Feb 1987.

978. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I—Message Encipherment and Authentication Procedures," RFC 1040, Jan 1988.

979. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I—Message Encipherment and Authentication Procedures," RFC 1113, Aug 1989.

980. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part III—Algorithms, Modes, and Identifiers," RFC 1115, Aug 1989.

981. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I—Message Encipherment and Authentication Procedures," RFC 1421, Feb 1993.

982. S. Lloyd, "Counting Binary Functions with Certain Cryptographic Properties," *Journal of Cryptology*, v. 5, n. 2, 1992, pp. 107–131.

983. T.M.A. Lomas, "Collision-Freedom, Considered Harmful, or How to Boot a Computer," *Proceedings of the 1995 Korea-Japan Workshop on Information Security and Cryptography*, Inuyama, Japan, 24–26 Jan 1995, pp. 35–42.

984. T.M.A. Lomas and M. Roe, "Forging a Clipper Message," *Communications of the ACM*, v. 37, n. 12, 1994, p. 12.

985. D.L. Long, "The Security of Bits in the Discrete Logarithm," Ph.D. dissertation, Princeton University, Jan 1984.

986. D.L. Long and A. Wigderson, "How Discrete Is the Discrete Log," *Proceedings of the 15th Annual ACM Syposium on the Theory of Computing*, Apr 1983.

987. D. Longley and S. Rigby, "An Automatic Search for Security Flaws in Key Management Schemes," *Computers and Security*, v. 11, n. 1, Jan 1992. pp. 75–89.

988. S.H. Low, N.F. Maxemchuk, and S. Paul, "Anonymous Credit Cards," *Proceedings of the 2nd Annual ACM Conference on Computer and Communications Security*, ACM Press, 1994, pp. 108–117.

989. J.H. Loxton, D.S.P. Khoo, G.J. Bird, and J. Seberry, "A Cubic RSA Code Equivalent to Factorization," *Journal of Cryptology*, v. 5, n. 2, 1992, pp. 139–150.

990. S.C. Lu and L.N. Lee, "A Simple and Effective Public-Key Cryptosystem," *COMSAT Technical Review*, 1979, pp. 15–24.

991. M. Luby, S. Micali, and C. Rackoff, "How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin," *Proceedings of the 24nd Annual Symposium on the Foundations of Computer Science*, 1983, pp. 11–22.

992. M. Luby and C. Rackoff, "How to Construct Pseudo-Random Permutations from Pseudorandom Functions," *SIAM Journal on Computing*, Apr 1988, pp. 373–386.

993. F. Luccio and S. Mazzone, "A Cryptosystem for Multiple Communications," *Information Processing Letters*, v. 10, 1980, pp. 180–183.

994. V. Luchangco and K. Koyama, "An Attack on an ID-Based Key Sharing System, *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 262–271.

995. D.J.C. MacKay, "A Free Energy Minimization Framework for Inferring the State of a Shift Register Given the Noisy Output Sequence," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.

996. M.D. MacLaren and G. Marsaglia, "Uniform Random Number Generators," *Journal of the ACM* v. 12, n. 1, Jan 1965, pp. 83–89.

997. D. MacMillan, "Single Chip Encrypts Data at 14Mb/s," *Electronics*, v. 54, n. 12, 16 June 1981, pp. 161–165.

998. R. Madhavan and L.E. Peppard, "A Multiprocessor GaAs RSA Cryptosystem," *Proceedings CCVLSI-89: Canadian Conference on Very Large Scale Integration*, Vancouver, BC, Canada, 22–24 Oct 1989, pp. 115–122.

999. W.E. Madryga, "A High Performance Encryption Algorithm," *Computer Security: A Global Challenge*, Elsevier Science Publishers, 1984, pp. 557–570.

1000. M. Mambo, A. Nishikawa, S. Tsujii, and E. Okamoto, "Efficient Secure Broadcast

Communication System," *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 23–33.

1001. M. Mambo, K. Usuda, and E. Okamoto, "Proxy Signatures," *Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS 95)*, Inuyama, Japan, 24–27 Jan 1995, pp. B1.1.1–17.
1002. W. Mao and C. Boyd, "Towards Formal Analysis of Security Protocols," *Proceedings of the Computer Security Foundations Workshop VI*, IEEE Computer Society Press, 1993, pp. 147–158.
1003. G. Marsaglia and T.A. Bray, "On-Line Random Number Generators and their Use in Combinations," *Communications of the ACM*, v. 11, n. 11, Nov 1968, p. 757–759.
1004. K.M. Martin, "Untrustworthy Participants in Perfect Secret Sharing Schemes," *Cryptography and Coding III*, M.J. Ganley, ed., Oxford: Clarendon Press, 1993, pp. 255–264.
1005. J.L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Transactions on Information Theory*, v. IT-15, n. 1, Jan 1969, pp. 122–127.
1006. J.L. Massey, "Cryptography and System Theory," *Proceedings of the 24th Allerton Conference on Communication, Control, and Computers*, 1–3 Oct 1986, pp. 1–8.
1007. J.L. Massey, "An Introduction to Contemporary Cryptology," *Proceedings of the IEEE*, v. 76, n. 5., May 1988, pp. 533–549.
1008. J.L. Massey, "Contemporary Cryptology: An Introduction," in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 1–39.
1009. J.L. Massey, "SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 1–17.
1010. J.L. Massey, "SAFER K-64: One Year Later," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
1011. J.L. Massey and I. Ingemarsson, "The Rip Van Winkle Cipher—A Simple and Provably Computationally Secure Cipher with a Finite Key," *IEEE International Symposium on Information Theory*, Brighton, UK, May 1985.
1012. J.L. Massey and X. Lai, "Device for Converting a Digital Block and the Use Thereof," International Patent PCT/CH91/00117, 28 Nov 1991.
1013. J.L. Massey and X. Lai, "Device for the Conversion of a Digital Block and Use of Same," U.S. Patent #5,214,703, 25 May 1993.
1014. J.L. Massey and R.A. Rueppel, "Linear Ciphers and Random Sequence Generators with Multiple Clocks," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 74–87.
1015. M. Matsui, "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 386–397.
1016. M. Matsui, "Linear Cryptanalysis of DES Cipher (I)," *Proceedings of the 1993 Symposium on Cryptography and Information Security (SCIS 93)*, Shuzenji, Japan, 28–30 Jan 1993, pp. 3C.1–14. (In Japanese.)
1017. M. Matsui, "Linear Cryptanalysis Method for DES Cipher (III)," *Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94)*, Lake Biwa, Japan, 27–29 Jan 1994, pp. 4A.1–11. (In Japanese.)
1018. M. Matsui, "On Correlation Between the Order of the S-Boxes and the Strength of DES," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
1019. M. Matsui, "The First Experimental Cryptanalysis of the Data Encryption Standard," *Advances in Cryptology—CRYPTO '94 Proceedings*, Springer-Verlag, 1994, pp. 1–11.
1020. M. Matsui and A. Yamagishi, "A New Method for Known Plaintext Attack of FEAL Cipher," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 81–91.
1021. T. Matsumoto and H. Imai, "A Class of Asymmetric Crypto-Systems Based on Polynomials Over Finite Rings," *IEEE International Symposium on Information Theory*, 1983, pp. 131–132.
1022. T. Matsumoto and H. Imai, "On the Key Production System: A Practical Solution to the Key Distribution Problem," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 185–193.
1023. T. Matsumoto and H. Imai, "On the Security of Some Key Sharing Schemes (Part

2}," IEICE Japan, Technical Report, ISEC90-28, 1990.

1024. S.M. Matyas, "Digital Signatures—An Overview," *Computer Networks*, v. 3, n. 2, Apr 1979, pp. 87–94.

1025. S.M. Matyas, "Key Handling with Control Vectors," *IBM Systems Journal*, v. 30, n. 2, 1991, pp. 151–174.

1026. S.M. Matyas, A.V. Le, and D.G. Abraham, "A Key Management Scheme Based on Control Vectors," *IBM Systems Journal*, v. 30, n. 2, 1991, pp. 175–191.

1027. S.M. Matyas and C.H. Meyer, "Generation, Distribution, and Installation of Cryptographic Keys," *IBM Systems Journal*, v. 17, n. 2, 1978, pp. 126–137.

1028. S.M. Matyas, C.H. Meyer, and J. Oseas, "Generating Strong One-Way Functions with Cryptographic Algorithm," *IBM Technical Disclosure Bulletin*, v. 27, n. 10A, Mar 1985, pp. 5658–5659.

1029. U.M. Maurer, "Provable Security in Cryptography," Ph.D. dissertation, ETH No. 9260, Swiss Federal Institute of Technology, Zürich, 1990.

1030. U.M. Maurer, "A Provable-Secure Strongly-Randomized Cipher," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 361–373.

1031. U.M. Maurer, "A Universal Statistical Test for Random Bit Generators," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 409–420.

1032. U.M. Maurer, "A Universal Statistical Test for Random Bit Generators," *Journal of Cryptology*, v. 5, n. 2, 1992, pp. 89–106.

1033. U.M. Maurer and J.L. Massey, "Cascade Ciphers: The Importance of Being First," *Journal of Cryptology*, v. 6, n. 1, 1993, pp. 55–61.

1034. U.M. Maurer and J.L. Massey, "Perfect Local Randomness in Pseudo-Random Sequences," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 110–112.

1035. U.M. Maurer and Y. Yacobi, "Non-interactive Public Key Cryptography," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 498–507.

1036. G. Mayhew, "A Low Cost, High Speed Encryption System and Method," *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, 1994, pp. 147–154.

1037. G. Mayhew, R. Frazee, and M. Bianco, "The Kinetic Protection Device," *Proceedings of the 15th National Computer Security Conference*, NIST, 1994, pp. 147–154.

1038. K.S. McCurley, "A Key Distribution System Equivalent to Factoring," *Journal of Cryptology*, v. 1, n. 2, 1988, pp. 95–106.

1039. K.S. McCurley, "The Discrete Logarithm Problem," *Cryptography and Computational Number Theory (Proceedings of the Symposium on Applied Mathematics)*, American Mathematics Society, 1990, pp. 49–74.

1040. K.S. McCurley, open letter from the Sandia National Laboratories on the DSA of the NIST, 7 Nov 1991.

1041. R.J. McEliece, "A Public-Key Cryptosystem Based on Algebraic Coding Theory," Deep Space Network Progress Report 42–44, Jet Propulsion Laboratory, California Institute of Technology, 1978, pp. 42–44.

1042. R.J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Boston: Kluwer Academic Publishers, 1987.

1043. P. McMahon, "SESAME V2 Public Key and Authorization Extensions to Kerberos," *Proceedings of the Internet Society 1995 Symposium on Network and Distributed Systems Security*, IEEE Computer Society Press, 1995, pp. 114–131.

1044. C.A. Meadows, "A System for the Specification and Analysis of Key Management Protocols," *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, 1991, pp. 182–195.

1045. C.A. Meadows, "Applying Formal Methods to the Analysis of a Key Management Protocol," *Journal of Computer Security*, v. 1, n. 1, 1992, pp. 5–35.

1046. C.A. Meadows, "A Model of Computation for the NRL Protocol Analyzer," *Proceedings of the Computer Security Foundations Workshop VII*, IEEE Computer Society Press, 1994, pp. 84–89.

1047. C.A. Meadows, "Formal Verification of Cryptographic Protocols: A Survey," *Advances in Cryptology—ASIACRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 133–150.

1048. G. Medvinsky and B.C. Neuman, "NetCash: A Design for Practical Electronic Currency on the Internet," *Proceedings of the 1st Annual ACM Conference on Computer and Communications Security*, ACM Press, 1993, pp. 102–106.

1049. G. Medvinsky and B.C. Neuman, "Electronic Currency for the Internet," *Electronic Markets*, v. 3, n. 9/10, Oct 1993, pp. 23–24.

1050. W. Meier, "On the Security of the IDEA Block Cipher," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 371–385.

1051. W. Meier and O. Staffelbach, "Fast Correlation Attacks on Stream Ciphers," *Journal of Cryptology*, v. 1, n. 3, 1989, pp. 159–176.

1052. W. Meier and O. Staffelbach, "Analysis of Pseudo Random Sequences Generated by Cellular Automata," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 186–199.

1053. W. Meier and O. Staffelbach, "Correlation Properties of Combiners with Memory in Stream Ciphers," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 204–213.

1054. W. Meier and O. Staffelbach, "Correlation Properties of Combiners with Memory in Stream Ciphers," *Journal of Cryptology*, v. 5, n. 1, 1992, pp. 67–86.

1055. W. Meier and O. Staffelbach, "The Self-Shrinking Generator," *Communications and Cryptography: Two Sides of One Tapestry*, R.E. Blahut et al., eds., Kluwer Adademic Publishers, 1994, pp. 287–295.

1056. J. Meijers, "Algebraic-Coded Cryptosystems," Master's thesis, Technical University Eindhoven, 1990.

1057. J. Meijers and J. van Tilburg, "On the Rao-Nam Private-Key Cryptosystem Using Linear Codes," *International Symposium on Information Theory*, Budapest, Hungary, 1991.

1058. J. Meijers and J. van Tilburg, "An Improved ST-Attack on the Rao-Nam Private-Key Cryptosystem," *International Conference on Finite Fields, Coding Theory, and Advances in Communications and Computing*, Las Vegas, NV, 1991.

1059. A. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

1060. A. Menezes, ed., *Applications of Finite Fields*, Kluwer Academic Publishers, 1993.

1061. A. Menezes and S.A. Vanstone, "Elliptic Curve Cryptosystems and Their Implementations," *Journal of Cryptology*, v. 6, n. 4, 1993, pp. 209–224.

1062. A. Menezes and S.A. Vanstone, "The Implementation of Elliptic Curve Cryptosystems," *Advances in Cryptology—AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 2–13.

1063. R. Menicocci, "Short Gollmann Cascade Generators May Be Insecure," *Codes and Ciphers*, Institute of Mathematics and its Applications, 1995, pp. 281–297.

1064. R.C. Merkle, "Secure Communication Over Insecure Channels," *Communications of the ACM*, v. 21, n. 4, 1978, pp. 294–299.

1065. R.C. Merkle, "Secrecy, Authentication, and Public Key Systems," Ph.D. dissertation, Stanford University, 1979.

1066. R.C. Merkle, "Method of Providing Digital Signatures," U.S. Patent #4,309,569, 5 Jan 1982.

1067. R.C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1998, pp. 369–378.

1068. R.C. Merkle, "A Certified Digital Signature," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 218–238.

1069. R.C. Merkle, "One Way Hash Functions and DES," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 428–446.

1070. R.C. Merkle, "A Fast Software One-Way Hash Function," *Journal of Cryptology*, v. 3, n. 1, 1990, pp. 43–58.

1071. R.C. Merkle, "Fast Software Encryption Functions," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 476–501.

1072. R.C. Merkle, "Method and Apparatus for Data Encryption," U.S. Patent #5,003,597, 26 Mar 1991.

1073. R.C. Merkle, personal communication, 1993.

1074. R.C. Merkle and M. Hellman, "Hiding Information and Signatures in Trapdoor Knapsacks," *IEEE Transactions on Infor-*

mation Theory, v. 24, n. 5, Sep 1978, pp. 525–530.

1075. R.C. Merkle and M. Hellman, "On the Security of Multiple Encryption," *Communications of the ACM*, v. 24, n. 7, 1981, pp. 465–467.

1076. M. Merritt, "Cryptographic Protocols," Ph.D. dissertation, Georgia Institute of Technology, GIT-ICS-83/6, Feb 1983.

1077. M. Merritt, "Towards a Theory of Cryptographic Systems: A Critique of Crypto-Complexity," *Distributed Computing and Cryptography*, J. Feigenbaum and M. Merritt, eds., American Mathematical Society, 1991, pp. 203–212.

1078. C.H. Meyer, "Ciphertext/Plaintext and Ciphertext/Key Dependencies vs. Number of Rounds for Data Encryption Standard," *AFIPS Conference Proceedings, 47*, 1978, pp. 1119–1126.

1079. C.H. Meyer, "Cryptography—A State of the Art Review," *Proceedings of Compeuro '89, VLSI and Computer Peripherals, 3rd Annual European Computer Conference*, IEEE Press, 1989, pp. 150–154.

1080. C.H. Meyer and S.M. Matyas, *Cryptography: A New Dimension in Computer Data Security*, New York: John Wiley & Sons, 1982.

1081. C.H. Meyer and M. Schilling, "Secure Program Load with Manipulation Detection Code," *Proceedings of Securicom '88*, 1988, pp. 111–130.

1082. C.H. Meyer and W.L. Tuchman, "Pseudo-Random Codes Can Be Cracked," *Electronic Design*, v. 23, Nov 1972.

1083. C.H. Meyer and W.L. Tuchman, "Design Considerations for Cryptography," *Proceedings of the NCC*, v. 42, Montvale, NJ: AFIPS Press, Nov 1979, pp. 594–597.

1084. S. Micali, "Fair Public-Key Cryptosystems," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 113–138.

1085. S. Micali, "Fair Cryptosystems," MIT/LCS/TR-579.b, MIT Laboratory for Computer Science, Nov 1993.

1086. S. Micali, "Fair Cryptosystems and Methods for Use," U.S. Patent #5,276,737, 4 Jan 1994.

1087. S. Micali, "Fair Cryptosystems and Methods for Use," U.S. Patent #5,315,658, 24 May 1994.

1088. S. Micali and A. Shamir, "An Improvement on the Fiat-Shamir Identification and Signature Scheme," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 244–247.

1089. M.J. Mihajlevič, "A Correlation Attack on the Binary Sequence Generators with Time-Varying Output Function," *Advances in Cryptology—ASIACRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 67–79.

1090. M.J. Mihajlevič and J.D. Golič, "A Fast Iterative Algorithm for a Shift Register Internal State Reconstruction Given the Noisy Output Sequence," *Advances in Cryptology—AUSCRYPT '90 Proceedings*, Springer-Verlag, 1990, pp. 165–175.

1091. M.J. Mihajlevič and J.D. Golič, "Convergence of a Bayesian Iterative Error-Correction Procedure to a Noisy Shift Register Sequence," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 124–137.

1092. J.K. Millen, S.C. Clark, and S.B. Freedman, "The Interrogator: Protocol Security Analysis," *IEEE Transactions on Software Engineering*, v. SE-13, n. 2, Feb 1987, pp. 274–288.

1093. G.L. Miller, "Riemann's Hypothesis and Tests for Primality," *Journal of Computer Systems Science*, v. 13, n. 3, Dec 1976, pp. 300–317.

1094. S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer, "Section E.2.1: Kerberos Authentication and Authorization System," MIT Project Athena, Dec 1987.

1095. V.S. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 417–426.

1096. M. Minsky, *Computation: Finite and Infinite Machines*, Englewood Cliffs, NJ: Prentice-Hall, 1967.

1097. C.J. Mitchell, "Authenticating Multi-Cast Internet Electronic Mail Messages Using a Bidirectional MAC Is Insecure," draft manuscript, 1990.

1098. C.J. Mitchell, "Enumerating Boolean Functions of Cryptographic Significance," *Journal of Cryptology*, v. 2, n. 3, 1990, pp. 155–170.

1099. C.J. Mitchell, F. Piper, and P. Wild, "Digital Signatures," *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1991, pp. 325–378.

1100. C.J. Mitchell, M. Walker, and D. Rush, "CCITT/ISO Standards for Secure Message

Handling," *IEEE Journal on Selected Areas in Communications*, v. 7, n. 4, May 1989, pp. 517–524.

1101. S. Miyaguchi, "Fast Encryption Algorithm for the RSA Cryptographic System," *Proceedings of Compcon 82*, IEEE Press, pp. 672–678.

1102. S. Miyaguchi, "The FEAL-8 Cryptosystem and Call for Attack," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 624–627.

1103. S. Miyaguchi, "Expansion of the FEAL Cipher," *NTT Review*, v. 2, n. 6, Nov 1990.

1104. S. Miyaguchi, "The FEAL Cipher Family," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 627–638.

1105. S. Miyaguchi, K. Ohta, and M. Iwata, "128-bit Hash Function (*N*-Hash)," *Proceedings of SECURICOM '90*, 1990, pp. 127–137.

1106. S. Miyaguchi, K. Ohta, and M. Iwata, "128-bit Hash Function (*N*-Hash)," *NTT Review*, v. 2, n. 6, Nov 1990, pp. 128–132.

1107. S. Miyaguchi, K. Ohta, and M. Iwata, "Confirmation that Some Hash Functions Are Not Collision Free," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 326–343.

1108. S. Miyaguchi, A. Shiraishi, and A. Shimizu, "Fast Data Encipherment Algorithm FEAL-8," *Review of the Electrical Communication Laboratories*, v. 36, n. 4, 1988.

1109. H. Miyano, "Differential Cryptanalysis on CALC and Its Evaluation," *Proceedings of the 1992 Symposium on Cryptography and Information Security (SCIS 92)*, Tateshina, Japan, 2–4 Apr 1992, pp. 7B.1–8.

1110. R. Molva, G. Tsudik, E. van Herreweghen, and S. Zatti, "KryptoKnight Authentication and Key Distribution System," *Proceedings of European Symposium on Research in Computer Security*, Toulouse, France, Nov 1992.

1111. P.L. Montgomery, "Modular Multiplication without Trial Division," *Mathematics of Computation*, v. 44, n. 170, 1985, pp. 519–521.

1112. P.L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of Computation*, v. 48, n. 177, Jan 1987, pp. 243–264.

1113. P.L. Montgomery and R. Silverman, "An FFT Extension to the *p*-1 Factoring Algorithm," *Mathematics of Computation*, v. 54, n. 190, 1990, pp. 839–854.

1114. J.H. Moore, "Protocol Failures in Cryptosystems," *Proceedings of the IEEE*, v. 76, n. 5, May 1988.

1115. J.H. Moore, "Protocol Failures in Cryptosystems," in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 541–558.

1116. J.H. Moore and G.J. Simmons, "Cycle Structure of the DES with Weak and Semi-Weak Keys," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 3–32.

1117. T. Moriyasu, M. Morii, and M. Kasahara, "Nonlinear Pseudorandom Number Generator with Dynamic Structure and Its Properties," *Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94)*, Biwako, Japan, 27–29 Jan 1994, pp. 8A.1–11.

1118. R. Morris, "The Data Encryption Standard—Retrospective and Prospects," *IEEE Communications Magazine*, v. 16, n. 6, Nov 1978, pp. 11–14.

1119. R. Morris, remarks at the 1993 Cambridge Protocols Workshop, 1993.

1120. R. Morris, N.J.A. Sloane, and A.D. Wyner, "Assessment of the NBS Proposed Data Encryption Standard," *Cryptologia*, v. 1, n. 3, Jul 1977, pp. 281–291.

1121. R. Morris and K. Thompson, "Password Security: A Case History," *Communications of the ACM*, v. 22, n. 11, Nov 1979, pp. 594–597.

1122. S.B. Morris, "Escrow Encryption," lecture at MIT Laboratory for Computer Science, 2 Jun 1994.

1123. M.N. Morrison and J. Brillhart, "A Method of Factoring and the Factorization of F_7," *Mathematics of Computation*, v. 29, n. 129, Jan 1975, pp. 183–205.

1124. L.E. Moser, "A Logic of Knowledge and Belief for Reasoning About Computer Security," *Proceedings of the Computer Security Foundations Workshop II*, IEEE Computer Society Press, 1989, pp. 57–63.

1125. Motorola Government Electronics Division, *Advanced Techniques in Network Security*, Scottsdale, AZ, 1977.

1126. W.B. Müller, "Polynomial Functions in Modern Cryptology," *Contributions to General Algebra 3: Proceedings of the*

Vienna Conference, Vienna: Verlag Hölder-Pichler-Tempsky, 1985, pp. 7–32.

1127. W.B. Müller and W. Nöbauer, "Some Remarks on Public-Key Cryptography," *Studia Scientiarum Mathematicarum Hungarica*, v. 16, 1981, pp. 71–76.

1128. W.B. Müller and W. Nöbauer, "Cryptanalysis of the Dickson Scheme," *Advances in Cryptology—EUROCRYPT '85 Proceedings*, Springer-Verlag, 1986, pp. 50–61.

1129. C. Muller-Scholer, "A Microprocessor-Based Cryptoprocessor," *IEEE Micro*, Oct 1983, pp. 5–15.

1130. R.C. Mullin, E. Nemeth, and N. Weidenhofer, "Will Public Key Cryptosystems Live Up to Their Expectations?—HEP Implementation of the Discrete Log Codebreaker," *ICPP 85*, pp. 193–196.

1131. Y. Murakami and S. Kasahara, "An ID-Based Key Distribution Scheme," IEICE Japan, Technical Report, ISEC90-26, 1990.

1132. S. Murphy, "The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts," *Journal of Cryptology*, v. 2, n. 3, 1990, pp. 145–154.

1133. E.D. Myers, "STU-III—Multilevel Secure Computer Interface," *Proceedings of the Tenth Annual Computer Security Applications Conference*, IEEE Computer Society Press, 1994, pp. 170–179.

1134. D. Naccache, "Can O.S.S. be Repaired? Proposal for a New Practical Signature Scheme," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 233–239.

1135. D. Naccache, D. M'Raïhi, D. Raphaeli, and S. Vaudenay, "Can D.S.A. be Improved? Complexity Trade-Offs with the Digital Signature Standard," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.

1136. Y. Nakao, T. Kaneko, K. Koyama, and R. Terada, "A Study on the Security of RDES-1 Cryptosystem against Linear Cryptanalysis," *Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography*, Inuyama, Japan, 24–27 Jan 1995, pp. 163–172.

1137. M. Naor, "Bit Commitment Using Pseudo-Randomness," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 128–136.

1138. M. Naor and M. Yung, "Universal One-Way Hash Functions and Their Cryptographic Application," *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*, 1989, pp. 33–43.

1139. National Bureau of Standards, "Report of the Workshop on Estimation of Significant Advances in Computer Technology," NBSIR76-1189, National Bureau of Standards, U.S. Department of Commerce, 21–22 Sep 1976, Dec 1977.

1140. National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

1141. National Bureau of Standards, NBS FIPS PUB 46-1, "Data Encryption Standard," U.S. Department of Commerce, Jan 1988.

1142. National Bureau of Standards, NBS FIPS PUB 74, "Guidelines for Implementing and Using the NBS Data Encryption Standard," U.S. Department of Commerce, Apr 1981.

1143. National Bureau of Standards, NBS FIPS PUB 81, "DES Modes of Operation," U.S. Department of Commerce, Dec 1980.

1144. National Bureau of Standards, NBS FIPS PUB 112, "Password Usage," U.S. Department of Commerce, May 1985.

1145. National Bureau of Standards, NBS FIPS PUB 113, "Computer Data Authentication," U.S. Department of Commerce, May 1985.

1146. National Computer Security Center, "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria," NCSC-TG-005 Version 1, Jul 1987.

1147. National Computer Security Center, "Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria," NCSC-TG-021 Version 1, Apr 1991.

1148. National Computer Security Center, "A Guide to Understanding Data Remembrance in Automated Information Systems," NCSC-TG-025 Version 2, Sep 1991.

1149. National Institute of Standards and Technology, NIST FIPS PUB XX, "Digital Signature Standard," U.S. Department of Commerce, DRAFT, 19 Aug 1991.

1150. National Institute of Standards and Technology, NIST FIPS PUB 46-2, "Data Encryption Standard," U.S. Department of Commerce, Dec 93.

1151. National Institute of Standards and Technology, NIST FIPS PUB 171, "Key Management Using X9.17," U.S. Department of Commerce, Apr 92.

1152. National Institute of Standards and Technology, NIST FIPS PUB 180, "Secure Hash Standard," U.S. Department of Commerce, May 93.

1153. National Institute of Standards and Technology, NIST FIPS PUB 185, "Escrowed Encryption Standard," U.S. Department of Commerce, Feb 94.

1154. National Institute of Standards and Technology, NIST FIPS PUB 186, "Digital Signature Standard," U.S. Department of Commerce, May 1994.

1155. National Institute of Standards and Technology, "Clipper Chip Technology," 30 Apr 1993.

1156. National Institute of Standards and Technology, "Capstone Chip Technology," 30 Apr 1993.

1157. J. Nechvatal, "Public Key Cryptography," NIST Special Publication 800-2, National Institute of Standards and Technology, U.S. Department of Commerce, Apr 1991.

1158. J. Nechvatal, "Public Key Cryptography," *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 177–288.

1159. R.M. Needham and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, v. 21, n. 12, Dec 1978, pp. 993–999.

1160. R.M. Needham and M.D. Schroeder, "Authentication Revisited," *Operating Systems Review*, v. 21, n. 1, 1987, p. 7.

1161. D.M. Nessett, "A Critique of the Burrows, Abadi, and Needham Logic," *Operating System Review*, v. 20, n. 2, Apr 1990, pp. 35–38.

1162. B.C. Neuman and S. Stubblebine, "A Note on the Use of Timestamps as Nonces," *Operating Systems Review*, v. 27, n. 2, Apr 1993, pp. 10–14.

1163. B.C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications Magazine*, v. 32, n. 9, Sep 1994, pp. 33–38.

1164. L. Neuwirth, "Statement of Lee Neuwirth of Cylink on HR145," submitted to congressional committees considering HR145, Feb 1987.

1165. D.B. Newman, Jr. and R.L. Pickholtz, "Cryptography in the Private Sector," *IEEE Communications Magazine*, v. 24, n. 8, Aug 1986, pp. 7–10.

1166. H. Niederreiter, "A Public-Key Cryptosystem Based on Shift Register Sequences," *Advances in Cryptology—EUROCRYPT '85 Proceedings*, Springer-Verlag, 1986, pp. 35–39.

1167. H. Niederreiter, "Knapsack-Type Cryptosystems and Algebraic Coding Theory," *Problems of Control and Information Theory*, v. 15, n. 2, 1986, pp. 159–166.

1168. H. Niederreiter, "The Linear Complexity Profile and the Jump Complexity of Keystream Sequences," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 174–188.

1169. V. Niemi, "A New Trapdoor in Knapsacks," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 405–411.

1170. V. Niemi and A. Renvall, "How to Prevent Buying of Voters in Computer Elections," *Advances in Cryptology—ASIACRYPT '94 Proceedings*, Springer-Verlag, 1995, pp. 164–170.

1171. I. Niven and H.A. Zuckerman, *An Introduction to the Theory of Numbers*, New York: John Wiley & Sons, 1972.

1172. R. Nöbauer, "Cryptanalysis of the Rédei Scheme," *Contributions to General Algebra 3: Proceedings of the Vienna Conference*, Verlag Hölder-Pichler-Tempsky, Vienna, 1985, pp. 255–264.

1173. R. Nöbauer, "Cryptanalysis of a Public-Key Cryptosystem Based on Dickson-Polynomials," *Mathematica Slovaca*, v. 38, n. 4, 1988, pp. 309–323.

1174. K. Noguchi, H. Ashiya, Y. Sano, and T. Kaneko, "A Study on Differential Attack of MBAL Cryptosystem," *Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94)*, Lake Biwa, Japan, 27–29 Jan 1994, pp. 14B.1–7. (In Japanese.)

1175. H. Nurmi, A. Salomaa, and L. Santean, "Secret Ballot Elections in Computer Networks," *Computers & Security*, v. 10, 1991, pp. 553–560.

1176. K. Nyberg, "Construction of Bent Functions and Difference Sets," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 151–160.

1177. K. Nyberg, "Perfect Nonlinear S-Boxes," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 378–386.

1178. K. Nyberg, "On the Construction of Highly Nonlinear Permutations," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1991, pp. 92–98.

1179. K. Nyberg, "Differentially Uniform Mappings for Cryptography," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 55–64.

1180. K. Nyberg, "Provable Security against Differential Cryptanalysis," presented at the rump session of Eurocrypt '94, May 1994.

1181. K. Nyberg and L.R. Knudsen, "Provable Security against Differential Cryptanalysis," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 566–574.

1182. K. Nyberg and L.R. Knudsen, "Provable Security against Differential Cryptanalysis," *Journal of Cryptology*, v. 8, n. 1, 1995, pp. 27–37.

1183. K. Nyberg and R.A. Rueppel, "A New Signature Scheme Based on the DSA Giving Message Recovery," *1st ACM Conference on Computer and Communications Security*, ACM Press, 1993, pp. 58–61.

1184. K. Nyberg and R.A. Rueppel, "Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.

1185. L. O'Connor, "Enumerating Nondegenerate Permutations," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 368–377.

1186. L. O'Connor, "On the Distribution of Characteristics in Bijective Mappings," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 360–370.

1187. L. O'Connor, "On the Distribution of Characteristics in Composite Permutations," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 403–412.

1188. L. O'Connor and A. Klapper, "Algebraic Nonlinearity and Its Application to Cryptography," *Journal of Cryptology*, v. 7, n. 3, 1994, pp. 133–151.

1189. A. Odlyzko, "Discrete Logarithms in Finite Fields and Their Cryptographic Significance," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 224–314.

1190. A. Odlyzko, "Progress in Integer Factorization and Discrete Logarithms," unpublished manuscript, Feb 1995.

1191. Office of Technology Assessment, U.S. Congress, "Defending Secrets, Sharing Data: New Locks and Keys for Electronic Communication," OTA-CIT-310, Washington, D.C.: U.S. Government Printing Office, Oct 1987.

1192. B. O'Higgins, W. Diffie, L. Strawczynski, and R. de Hoog, "Encryption and ISDN—a Natural Fit," *Proceedings of the 1987 International Switching Symposium*, 1987, pp. 863–869.

1193. Y. Ohnishi, "A Study on Data Security," Master's thesis, Tohuku University, Japan, 1988. (In Japanese.)

1194. K. Ohta, "A Secure and Efficient Encrypted Broadcast Communication System Using a Public Master Key," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. J70-D, n. 8, Aug 1987, pp. 1616–1624.

1195. K. Ohta, "An Electrical Voting Scheme Using a Single Administrator," *IEICE Spring National Convention*, A-294, 1988, v. 1, p. 296. (In Japanese.)

1196. K. Ohta, "Identity-based Authentication Schemes Using the RSA Cryptosystem," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. J72D-II, n. 8, Aug 1989, pp. 612–620.

1197. K. Ohta and M. Matsui, "Differential Attack on Message Authentication Codes," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 200–223.

1198. K. Ohta and T. Okamoto, "Practical Extension of Fiat-Shamir Scheme," *Electronics Letters*, v. 24, n. 15, 1988, pp. 955–956.

1199. K. Ohta and T. Okamoto, "A Modification of the Fiat-Shamir Scheme," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 232–243.

1200. K. Ohta and T. Okamoto, "A Digital Multisignature Scheme Based on the Fiat-Shamir Scheme," *Advances in Cryptology—ASIACRYPT '91 Proceedings*, Springer-Verlag, 1993, pp. 139–148.

1201. K. Ohta, T. Okamoto and K. Koyama, "Membership Authentication for Hierarchy Multigroups Using the Extended Fiat-Shamir Scheme," *Advances in Cryptol-*

ogy—EUROCRYPT '90 Proceedings, Springer-Verlag, 1991, pp. 446–457.

1202. E. Okamoto and K. Tanaka, "Key Distribution Based on Identification Information," *IEEE Journal on Selected Areas in Communication*, v. 7, n. 4, May 1989, pp. 481–485.
1203. T. Okamoto, "Fast Public-Key Cryptosystems Using Congruent Polynomial Equations," *Electronics Letters*, v. 22, n. 11, 1986, pp. 581–582.
1204. T. Okamoto, "Modification of a Public-Key Cryptosystem," *Electronics Letters*, v. 23, n. 16, 1987, pp. 814–815.
1205. T. Okamoto, "A Fast Signature Scheme Based on Congruential Polynomial Operations," *IEEE Transactions on Information Theory*, v. 36, n. 1, 1990, pp. 47–53.
1206. T. Okamoto, "Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 31–53.
1207. T. Okamoto, A. Fujioka, and E. Fujisaki, "An Efficient Digital Signature Scheme Based on Elliptic Curve over the Ring Z_n," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, 1993, pp. 54–65.
1208. T. Okamoto, S. Miyaguchi, A. Shiraishi, and T. Kawoaka, "Signed Document Transmission System," U.S. Patent #4,625,076, 25 Nov 1986.
1209. T. Okamoto and K. Ohta, "Disposable Zero-Knowledge Authentication and Their Applications to Untraceable Electronic Cash," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 134–149.
1210. T. Okamoto and K. Ohta, "How to Utilize the Randomness of Zero-Knowledge Proofs," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 456–475.
1211. T. Okamoto and K. Ohta, "Universal Electronic Cash," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 324–337.
1212. T. Okamoto and K. Ohta, "Survey of Digital Signature Schemes," *Proceedings of the Third Symposium on State and Progress of Research in Cryptography*, Fondazone Ugo Bordoni, Rome, 1993, pp. 17–29.
1213. T. Okamoto and K. Ohta, "Designated Confirmer Signatures Using Trapdoor Functions," *Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 94)*, Lake Biwa, Japan, 27–29 Jan 1994, pp. 16B.1–11.
1214. T. Okamoto and K. Sakurai, "Efficient Algorithms for the Construction of Hyperelliptic Cryptosystems," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 267–278.
1215. T. Okamoto and A. Shiraishi, "A Fast Signature Scheme Based on Quadratic Inequalities," *Proceedings of the 1985 Symposium on Security and Privacy*, IEEE, Apr 1985, pp. 123–132.
1216. J.D. Olsen, R.A. Scholtz, and L.R. Welch, "Bent Function Sequences," *IEEE Transactions on Information Theory*, v. IT-28, n. 6, Nov 1982, pp. 858–864.
1217. H. Ong and C.P. Schnorr, "Signatures through Approximate Representations by Quadratic Forms," *Advances in Cryptology: Proceedings of Crypto 83*, Plenum Press, 1984.
1218. H. Ong and C.P. Schnorr, "Fast Signature Generation with a Fiat Shamir-Like Scheme," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 432–440.
1219. H. Ong, C.P. Schnorr, and A. Shamir, "An Efficient Signature Scheme Based on Polynomial Equations," *Proceedings of the 16th Annual Symposium on the Theory of Computing*, 1984, pp. 208–216.
1220. H. Ong, C.P. Schnorr, and A. Shamir, "Efficient Signature Schemes Based on Polynomial Equations," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 37–46.
1221. Open Shop Information Services, *OSIS Security Aspects*, OSIS European Working Group, WG1, final report, Oct 1985.
1222. G.A. Orton, M.P. Roy, P.A. Scott, L.E. Peppard, and S.E. Tavares, "VLSI Implementation of Public-Key Encryption Algorithms," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 277–301.
1223. H. Orup, E. Svendsen, and E. Andreasen, "VICTOR—An Efficient RSA Hardware Implementation," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 245–252.
1224. D. Otway and O. Rees, "Efficient and Timely Mutual Authentication," *Operating Systems Review*, v. 21, n. 1, 1987, pp. 8–10.

1225. G. Pagels-Fick, "Implementation Issues for Master Key Distribution and Protected Keyload Procedures," *Computers and Security: A Global Challenge, Proceedings of IFIP/SEC '83*, North Holland: Elsevier Science Publishers, 1984, pp. 381–390.

1226. C.M. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

1227. C.S. Park, "Improving Code Rate of McEliece's Public-key Cryptosystem," *Electronics Letters*, v. 25, n. 21, 12 Oct 1989, pp. 1466–1467.

1228. S. Park, Y. Kim, S. Lee, and K. Kim, "Attacks on Tanaka's Non-interactive Key Sharing Scheme," *Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS 95)*, Inuyama, Japan, 24–27 Jan 1995, pp. B3.4.1–4.

1229. S.J. Park, K.H. Lee, and D.H. Lon, "An Entrusted Undeniable Signature," *Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography*, Inuyama, Japan, 24–27 Jan 1995, pp. 120–126.

1230. S.J. Park, K.H. Lee, and D.H. Lon, "A Practical Group Signature," *Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography*, Inuyama, Japan, 24–27 Jan 1995, pp. 127–133.

1231. S.K. Park and K.W. Miller, "Random Number Generators: Good Ones Are Hard to Find," *Communications of the ACM*, v. 31, n. 10, Oct 1988, pp. 1192–1201.

1232. J. Patarin, "How to Find and Avoid Collisions for the Knapsack Hash Function," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 305–317.

1233. W. Patterson, *Mathematical Cryptology for Computer Scientists and Mathematicians*, Totowa, N.J.: Rowman & Littlefield, 1987.

1234. W.H. Payne, "Public Key Cryptography Is Easy to Break," William H. Payne, unpublished manuscript, 16 Oct 90.

1235. T.P. Pederson, "Distributed Provers with Applications to Undeniable Signatures," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 221–242.

1236. S. Peleg and A. Rosenfield, "Breaking Substitution Ciphers Using a Relaxation Algorithm," *Communications of the ACM*, v. 22, n. 11, Nov 1979, pp. 598–605.

1237. R. Peralta, "Simultaneous Security of Bits in the Discrete Log," *Advances in Cryptology—EUROCRYPT '85*, Springer-Verlag, 1986, pp. 62–72.

1238. I. Peterson, "Monte Carlo Physics: A Cautionary Lesson," *Science News*, v. 142, n. 25, 19 Dec 1992, p. 422.

1239. B. Pfitzmann, "Fail-Stop Signatures: Principles and Applications," *Proceedings of COMPUSEC '91, Eighth World Conference on Computer Security, Audit, and Control*, Elsevier Science Publishers, 1991, pp. 125–134.

1240. B. Pfitzmann and M. Waidner, "Formal Aspects of Fail-Stop Signatures," Fakultät für Informatik, University Karlsruhe, Report 22/90, 1990.

1241. B. Pfitzmann and M. Waidner, "Fail-Stop Signatures and Their Application," *Securicom '91*, 1991, pp. 145–160.

1242. B. Pfitzmann and M. Waidner, "Unconditional Concealment with Cryptographic Ruggedness," *VIS '91 Verlassliche Informationsysteme Proceedings*, Darmstadt, Germany, 13–15 March 1991, pp. 3-2-320. [In German.]

1243. B. Pfitzmann and M. Waidner, "How to Break and Repair a 'Provably Secure' Untraceable Payment System," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 338–350.

1244. C.P. Pfleeger, *Security in Computing*, Englewood Cliffs, N.J.: Prentice-Hall, 1989.

1245. S.J.D. Phoenix and P.D. Townsend, "Quantum Cryptography and Secure Optical Communication," *BT Technology Journal*, v. 11, n. 2, Apr 1993, pp. 65–75.

1246. J. Pieprzyk, "On Public-Key Cryptosystems Built Using Polynomial Rings," *Advances in Cryptology—EUROCRYPT '85*, Springer-Verlag, 1986, pp. 73–80.

1247. J. Pieprzyk, "Error Propagation Property and Applications in Cryptography," *IEE Proceedings-E, Computers and Digital Techniques*, v. 136, n. 4, Jul 1989, pp. 262–270.

1248. D. Pinkas, T. Parker, and P. Kaijser, "SESAME: An Introduction," Issue 1.2, Bull, ICL, and SNI, Sep 1993.

1249. F. Piper, "Stream Ciphers," *Elektrotechnic und Maschinenbau*, v. 104, n. 12, 1987, pp. 564–568.

1250. V.S. Pless, "Encryption Schemes for Computer Confidentiality," *IEEE Transactions on Computing*, v. C-26, n. 11, Nov 1977, pp. 1133–1136.

1251. J.B. Plumstead, "Inferring a Sequence Generated by a Linear Congruence," *Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science*, 1982, pp. 153–159.

1252. R. Poet, "The Design of Special Purpose Hardware to Factor Large Integers," *Computer Physics Communications*, v. 37, 1985, pp. 337–341.

1253. S.C. Pohlig and M.E. Hellman, "An Improved Algorithm for Computing Logarithms in GF(*p*) and Its Cryptographic Significance," *IEEE Transactions on Information Theory*, v. 24, n. 1, Jan 1978, pp. 106–111.

1254. J.M. Pollard, "A Monte Carlo Method for Factorization," *BIT*, v. 15, 1975, pp. 331–334.

1255. J.M. Pollard and C.P. Schnorr, "An Efficient Solution of the Congruence $x^2 + ky^2 = m \pmod{n}$," *IEEE Transactions on Information Theory*, v. IT-33, n. 5, Sep 1987, pp. 702–709.

1256. C. Pomerance, "Recent Developments in Primality Testing," *The Mathematical Intelligencer*, v. 3, n. 3, 1981, pp. 97–105.

1257. C. Pomerance, "The Quadratic Sieve Factoring Algorithm," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, 169–182.

1258. C. Pomerance, "Fast, Rigorous Factorization and Discrete Logarithm Algorithms," *Discrete Algorithms and Complexity*, New York: Academic Press, 1987, pp. 119–143.

1259. C. Pomerance, J.W. Smith, and R. Tuler, "A Pipe-Line Architecture for Factoring Large Integers with the Quadratic Sieve Algorithm," *SIAM Journal on Computing*, v. 17, n. 2, Apr 1988, pp. 387–403.

1260. G.J. Popek and C.S. Kline, "Encryption and Secure Computer Networks," *ACM Computing Surveys*, v. 11, n. 4, Dec 1979, pp. 331–356.

1261. F. Pratt, *Secret and Urgent*, Blue Ribbon Books, 1942.

1262. B. Preneel, "Analysis and Design of Cryptographic Hash Functions," Ph.D. dissertation, Katholieke Universiteit Leuven, Jan 1993.

1263. B. Preneel, "Differential Cryptanalysis of Hash Functions Based on Block Ciphers," *Proceedings of the 1st ACM Conference on Computer and Communications Security*, 1993, pp. 183–188.

1264. B. Preneel, "Cryptographic Hash Functions," *European Transactions on Telecommunications*, v 5, n. 4, Jul/Aug 1994, pp. 431–448.

1265. B. Preneel, personal communication, 1995.

1266. B. Preneel, A. Bosselaers, R. Govaerts, and J. Vanderwalle, "Collision-Free Hash Functions Based on Block Cipher Algorithms," *Proceedings of the 1989 Carnahan Conference on Security Technology*, 1989, pp. 203–210.

1267. B. Preneel, R. Govaerts, and J. Vandewalle, "An Attack on Two Hash Functions by Zheng-Matsumoto-Imai," *Advances in Cryptology—ASIACRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 535–538.

1268. B. Preneel, R. Govaerts, and J. Vandewalle, "Hash Functions Based on Block Ciphers: A Synthetic Approach," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 368–378.

1269. B. Preneel, M. Nuttin, V. Rijmen, and J. Buelens, "Cryptanalysis of the CFB mode of the DES with a Reduced Number of Rounds," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 212–223.

1270. B. Preneel and V. Rijmen, "On Using Maximum Likelihood to Optimize Recent Cryptanalytic Techniques," presented at the rump session of EUROCRYPT '94, May 1994.

1271. B. Preneel, W. Van Leekwijck, L. Van Linden, R. Govaerts, and J. Vandewalle, "Propagation Characteristics of Boolean Functions," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 161–173.

1272. W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 1988.

1273. W. Price, "Key Management for Data Encipherment," *Security: Proceedings of IFIP/SEC '83*, North Holland: Elsevier Science Publishers, 1983.

1274. G.P. Purdy, "A High-Security Log-in Procedure," *Communications of the ACM*, v. 17, n. 8, Aug 1974, pp. 442–445.

1275. J.-J. Quisquater, "Announcing the Smart Card with RSA Capability," *Proceedings of the Conference: IC Cards and Applications, Today and Tomorrow*, Amsterdam, 1989.

1276. J.-J. Quisquater and C. Couvreur, "Fast Decipherment Algorithm for RSA Public-Key Cryptosystem," *Electronic Letters*, v. 18, 1982, pp. 155–168.

1277. J.-J. Quisquater and J.-P. Delescaille, "Other Cycling Tests for DES," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 255–256.

1278. J.-J. Quisquater and Y.G. Desmedt, "Chinese Lotto as an Exhaustive Code-Breaking Machine," *Computer*, v. 24, n. 11, Nov 1991, pp. 14–22.

1279. J.-J. Quisquater and M. Girault, "2*n*-bit Hash Functions Using *n*-bit Symmetric Block Cipher Algorithms, *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 102–109.

1280. J.-J. Quisquater and L.C. Guillou, "Des Procédés d'Authentification Basés sur une Publication de Problèmes Complexes et Personnalisés dont les Solutions Maintenues Secrètes Constituent autant d'Accréditations," *Proceedings of SECURICOM '89: 7th Worldwide Congress on Computer and Communications Security and Protection*, Société d'Édition et d'Organisation d'Expositions Professionnelles, 1989, pp. 149–158. (In French.)

1281. J.-J., Myriam, Muriel, and Michaël Quisquater; L., Marie Annick, Gaïd, Anna, Gwenolé, and Soazig Guillou; and T. Berson, "How to Explain Zero-Knowledge Protocols to Your Children," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 628–631.

1282. M.O. Rabin, "Digital Signatures," *Foundations of Secure Communication*, New York: Academic Press, 1978, pp. 155–168.

1283. M.O. Rabin, "Digital Signatures and Public-Key Functions as Intractable as Factorization," MIT Laboratory for Computer Science, Technical Report, MIT/LCS/TR-212, Jan 1979.

1284. M.O. Rabin, "Probabilistic Algorithm for Testing Primality," *Journal of Number Theory*, v. 12, n. 1, Feb 1980, pp. 128–138.

1285. M.O. Rabin, "Probabilistic Algorithms in Finite Fields," *SIAM Journal on Computing*, v. 9, n. 2, May 1980, pp. 273–280.

1286. M.O. Rabin, "How to Exchange Secrets by Oblivious Transfer," Technical Memo TR-81, Aiken Computer Laboratory, Harvard University, 1981.

1287. M.O. Rabin, "Fingerprinting by Random Polynomials," Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

1288. T. Rabin and M. Ben-Or, "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority," *Proceedings of the 21st ACM Symposium on the Theory of Computing*, 1989, pp. 73–85.

1289. RAND Corporation, *A Million Random Digits with 100,000 Normal Deviates*, Glencoe, IL: Free Press Publishers, 1955.

1290. T.R.N. Rao, "Cryposystems Using Algebraic Codes," *International Conference on Computer Systems and Signal Processing*, Bangalore, India, Dec 1984.

1291. T.R.N. Rao, "On Struit-Tilburg Cryptanalysis of Rao-Nam Scheme," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 458–460.

1292. T.R.N. Rao and K.H. Nam, "Private-Key Algebraic-Coded Cryptosystems," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 35–48.

1293. T.R.N. Rao and K.H. Nam, "Private-Key Algebraic-Code Encryptions," *IEEE Transactions on Information Theory*, v. 35, n. 4, Jul 1989, pp. 829–833.

1294. J.A. Reeds, "Cracking Random Number Generator," *Cryptologia*, v. 1, n. 1, Jan 1977, pp. 20–26.

1295. J.A. Reeds, "Cracking a Multiplicative Congruential Encryption Algorithm," in *Information Linkage Between Applied Mathematics and Industry*, P.C.C. Wang, ed., Academic Press, 1979, pp. 467–472.

1296. J.A. Reeds, "Solution of Challenge Cipher," *Cryptologia*, v. 3, n. 2, Apr 1979, pp. 83–95.

1297. J.A. Reeds and J.L. Manferdelli, "DES Has No Per Round Linear Factors," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 377–389.

1298. J.A. Reeds and N.J.A. Sloane, "Shift Register Synthesis (Modulo *m*)," *SIAM Journal on Computing*, v. 14, n. 3, Aug 1985, pp. 505–513.

1299. J.A. Reeds and B.J. Weinberger, "File Security and the UNIX Crypt Command," *AT&T Technical Journal*, v. 63, n. 8, Oct 1984, pp. 1673–1683.

1300. T. Renji, "On Finite Automaton One-Key Cryptosystems," *Fast Software Encryption*,

Cambridge Security Workshop Proceedings, Springer-Verlag, 1994, pp. 135–148.

1301. T. Renji and C. Shihua, "A Finite Automaton Public Key Cryptosystems and Digital Signature," *Chinese Journal of Computers*, v. 8, 1985, pp. 401–409. (In Chinese.)
1302. T. Renji and C. Shihua, "Two Varieties of Finite Automaton Public Key Cryptosystems and Digital Signature," *Journal of Computer Science and Tecnology*, v. 1, 1986, pp. 9–18. (In Chinese.)
1303. T. Renji and C. Shihua, "An Implementation of Identity-based Cryptosystems and Signature Schemes by Finite Automaton Public Key Cryptosystems," *Advances in Cryptology—CHINACRYPT '92*, Bejing: Science Press, 1992, pp. 87–104. (In Chinese.)
1304. T. Renji and C. Shihua, "Note on Finite Automaton Public Key Cryptosystems," *CHINACRYPT '94*, Xidian, China, 11–15 Nov 1994, pp. 76–80.
1305. Research and Development in Advanced Communication Technologies in Europe, *RIPE Integrity Primitives: Final Report of RACE Integrity Primitives Evaluation (R1040)*, RACE, June 1992.
1306. J.M. Reyneri and E.D. Karnin, "Coin Flipping by Telephone," *IEEE Transactions on Information Theory*, v. IT-30, n. 5, Sep 1984, pp. 775–776.
1307. P. Ribenboim, *The Book of Prime Number Records*, Springer-Verlag, 1988.
1308. P. Ribenboim, *The Little Book of Big Primes*, Springer-Verlag, 1991.
1309. M. Richter, "Ein Rauschgenerator zur Gewinnung won quasi-idealen Zufallszahlen für die stochastische Simulation," Ph.D. dissertation, Aachen University of Technology, 1992. (In German.)
1310. R.F. Rieden, J.B. Snyder, R.J. Widman, and W.J. Barnard, "A Two-Chip Implementation of the RSA Public Encryption Algorithm," *Proceedings of GOMAC (Government Microcircuit Applications Conference)*, Nov 1982, pp. 24–27.
1311. H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Boston: Birkhaüser, 1985.
1312. K. Rihaczek, "Data Interchange and Legal Security—Signature Surrogates," *Computers & Security*, v. 13, n. 4, Sep 1994, pp. 287–293.
1313. V. Rijmen and B. Preneel, "Improved Characteristics for Differential Cryptanalysis of Hash Functions Based on Block Ciphers," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
1314. R.L. Rivest, "A Description of a Single-Chip Implementation of the RSA Cipher," *LAMBDA Magazine*, v. 1, n. 3, Fall 1980, pp. 14–18.
1315. R.L. Rivest, "Statistical Analysis of the Hagelin Cryptograph," *Cryptologia*, v. 5, n. 1, Jan 1981, pp. 27–32.
1316. R.L. Rivest, "A Short Report on the RSA Chip," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, p. 327.
1317. R.L. Rivest, "RSA Chips (Past/Present/Future)," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 159–168.
1318. R.L. Rivest, "The MD4 Message Digest Algorithm," RFC 1186, Oct 1990.
1319. R.L. Rivest, "The MD4 Message Digest Algorithm," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 303–311.
1320. R.L. Rivest, "The RC4 Encryption Algorithm," RSA Data Security, Inc., Mar 1992.
1321. R.L. Rivest, "The MD4 Message Digest Algorithm," RFC 1320, Apr 1992.
1322. R.L. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, Apr 1992.
1323. R.L. Rivest, "Dr. Ron Rivest on the Difficulty of Factoring," *Ciphertext: The RSA Newsletter*, v. 1, n. 1, Fall 1993, pp. 6, 8.
1324. R.L. Rivest, "The RC5 Encryption Algorithm," *Dr. Dobb's Journal*, v. 20, n. 1, Jan 95, pp. 146–148.
1325. R.L. Rivest, "The RC5 Encryption Algorithm," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.
1326. R.L. Rivest, M.E. Hellman, J.C. Anderson, and J.W. Lyons, "Responses to NIST's Proposal," *Communications of the ACM*, v. 35, n. 7, Jul 1992, pp. 41–54.
1327. R.L. Rivest and A. Shamir, "How to Expose an Eavesdropper," *Communications of the ACM*, v. 27, n. 4, Apr 1984, pp. 393–395.
1328. R.L. Rivest, A. Shamir, and L.M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, v. 21, n. 2, Feb 1978, pp. 120–126.
1329. R.L. Rivest, A. Shamir, and L.M. Adleman, "On Digital Signatures and Public Key

Cryptosystems," MIT Laboratory for Computer Science, Technical Report, MIT/LCS/TR-212, Jan 1979.

1330. R.L. Rivest, A. Shamir, and L.M. Adleman, "Cryptographic Communications System and Method," U.S. Patent #4,405,829, 20 Sep 1983.

1331. M.J.B. Robshaw, "Implementations of the Search for Pseudo-Collisions in MD5," Technical Report TR-103, Version 2.0, RSA Laboratories, Nov 1993.

1332. M.J.B. Robshaw, "The Final Report of RACE 1040: A Technical Summary," Technical Report TR-9001, Version 1.0, RSA Laboratories, Jul 1993.

1333. M.J.B. Robshaw, "On Evaluating the Linear Complexity of a Sequence of Least Period 2^n," *Designs, Codes and Cryptography*, v. 4, n. 3, 1994, pp. 263–269.

1334. M.J.B. Robshaw, "Block Ciphers," Technical Report TR-601, RSA Laboratories, Jul 1994.

1335. M.J.B. Robshaw, "MD2, MD4, MD5, SHA, and Other Hash Functions," Technical Report TR-101, Version 3.0, RSA Laboratories, Jul 1994.

1336. M.J.B. Robshaw, "On Pseudo-Collisions in MD5," Technical Report TR-102, Version 1.1, RSA Laboratories, Jul 1994.

1337. M.J.B. Robshaw, "Security of RC4," Technical Report TR-401, RSA Laboratories, Jul 1994.

1338. M.J.B. Robshaw, personal communication, 1995.

1339. M. Roe, "Reverse Engineering of an EES Device," *K.U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995, to appear.

1340. P. Rogaway and D. Coppersmith, "A Software-Oriented Encryption Algorithm," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 56–63.

1341. H.L. Rogers, "An Overview of the Candware Program," *Proceedings of the 3rd Annual Symposium on Physical/Electronic Security*, Armed Forces Communications and Electronics Association, paper 31, Aug 1987.

1342. J. Rompel, "One-Way Functions Are Necessary and Sufficient for Secure Signatures," *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, 1990, pp. 387–394.

1343. T. Rosati, "A High Speed Data Encryption Processor for Public Key Cryptography," *Proceedings of the IEEE Custom Integrated Circuits Conference*, 1989, pp. 12.3.1–12.3.5.

1344. O.S. Rothaus, "On 'Bent' Functions," *Journal of Combinational Theory*, Series A, v. 20, n. 3, 1976, pp. 300–305.

1345. RSA Laboratories, "PKCS #1: RSA Encryption Standard," version 1.5, Nov 1993.

1346. RSA Laboratories, "PKCS #3: Diffie-Hellman Key-Agreement Standard," version 1.4, Nov 1993.

1347. RSA Laboratories, "PKCS #5: Password-Based Encryption Standard," version 1.5, Nov 1993.

1348. RSA Laboratories, "PKCS #6: Extended-Certificate Syntax Standard," version 1.5, Nov 1993.

1349. RSA Laboratories, "PKCS #7: Cryptographic Message Syntax Standard," version 1.5, Nov 1993.

1350. RSA Laboratories, "PKCS #8: Private Key Information Syntax Standard," version 1.2, Nov 1993.

1351. RSA Laboratories, "PKCS #9: Selected Attribute Types," version 1.1, Nov 1993.

1352. RSA Laboratories, "PKCS #10: Certification Request Syntax Standard," version 1.0, Nov 1993.

1353. RSA Laboratories, "PKCS #11: Cryptographic Token Interface Standard," version 1.0, Apr 95.

1354. RSA Laboratories, "PKCS #12: Public Key User Information Syntax Standard," version 1.0, 1995.

1355. A.D. Rubin and P. Honeyman, "Formal Methods for the Analysis of Authentication Protocols," draft manuscript, 1994.

1356. F. Rubin, "Decrypting a Stream Cipher Based on J-K Flip-Flops," *IEEE Transactions on Computing*, v. C-28, n. 7, Jul 1979, pp. 483–487.

1357. R.A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, 1986.

1358. R.A. Rueppel, "Correlation Immunity and the Summation Combiner," *Advances in Cryptology—EUROCRYPT '85*, Springer-Verlag, 1986, pp. 260–272.

1359. R.A. Rueppel, "When Shift Registers Clock Themselves," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1987, pp. 53–64.

1360. R.A. Rueppel, "Security Models and Notions for Stream Ciphers," *Cryptogra-*

phy and Coding II, C. Mitchell, ed., Oxford: Clarendon Press, 1992, pp. 213–230.

1361. R.A. Rueppel, "On the Security of Schnorr's Pseudo-Random Sequence Generator," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 423–428.
1362. R.A. Rueppel, "Stream Ciphers," *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 65–134.
1363. R.A. Rueppel and J.L. Massey, "The Knapsack as a Nonlinear Function," *IEEE International Symposium on Information Theory*, Brighton, UK, May 1985.
1364. R.A. Rueppel and O.J. Staffelbach, "Products of Linear Recurring Sequences with Maximum Complexity," *IEEE Transactions on Information Theory*, v. IT-33, n. 1, Jan 1987, pp. 124–131.
1365. D. Russell and G.T. Gangemi, *Computer Security Basics*, O'Reilly and Associates, Inc., 1991.
1366. S. Russell and P. Craig, "Privacy Enhanced Mail Modules for ELM," *Proceedings of the Internet Society 1994 Workshop on Network and Distributed System Security*, The Internet Society, 1994, pp. 21–34.
1367. D.F.H. Sadok and J. Kelner, "Privacy Enhanced Mail Design and Implementation Perspectives," *Computer Communications Review*, v. 24, n. 3, Jul 1994, pp. 38–46.
1368. K. Sakano, "Digital Signatures with User-Flexible Reliability," *Proceedings of the 1993 Symposium on Cryptography and Information Security (SCIS 93)*, Shuzenji, Japan, 28–30 Jan 1993, pp. 5C.1–8.
1369. K. Sakano, C. Park, and K. Kurosawa, "{k,n} Threshold Undeniable Signature Scheme," *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 184–193.
1370. K. Sako, "Electronic Voting Schemes Allowing Open Objection to the Tally," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. E77-A, n. 1, 1994, pp. 24–30.
1371. K. Sako and J. Kilian, "Secure Voting Using Partially Compatible Homomorphisms," *Advances in Cryptology—CRYPTO '94 Proceedings*, Springer-Verlag, 1994, p. 411–424.
1372. K. Sako and J. Kilian, "Receipt-Free Mix-Type Voting Scheme—A Practical Solution to the Implementation of a Voting Booth," *Advances in Cryptology—EUROCRYPT '95 Proceedings*, Springer-Verlag, 1995, pp. 393–403.
1373. A. Salomaa, *Public-Key Cryptography*, Springer-Verlag, 1990.
1374. A. Salomaa and L. Santean, "Secret Selling of Secrets with Many Buyers," *ETACS Bulletin*, v. 42, 1990, pp. 178–186.
1375. M. Sántha and U.V. Vazirani, "Generating Quasi-Random Sequences from Slightly Random Sources," *Proceedings of the 25th Annual Symposium on the Foundations of Computer Science*, 1984, pp. 434–440.
1376. M. Sántha and U.V. Vazirani, "Generating Quasi-Random Sequences from Slightly Random Sources," *Journal of Computer and System Sciences*, v. 33, 1986, pp. 75–87.
1377. S. Saryazdi, "An Extension to ElGamal Public Key Cryptosystem with a New Signature Scheme," *Proceedings of the 1990 Bilkent International Conference on New Trends in Communication, Control, and Signal Processing*, North Holland: Elsevier Science Publishers, 1990, pp. 195–198.
1378. J.E. Savage, "Some Simple Self-Synchronizing Digital Data Scramblers," *Bell System Technical Journal*, v. 46, n. 2, Feb 1967, pp. 448–487.
1379. B.P. Schanning, "Applying Public Key Distribution to Local Area Networks," *Computers & Security*, v. 1, n. 3, Nov 1982, pp. 268–274.
1380. B.P. Schanning, S.A. Powers, and J. Kowalchuk, "MEMO: Privacy and Authentication for the Automated Office," *Proceedings of the 5th Conference on Local Computer Networks*, IEEE Press, 1980, pp. 21–30.
1381. Schaumuller-Bichl, "Zur Analyse des Data Encryption Standard und Synthese Verwandter Chiffriersysteme," Ph.D. dissertation, Linz University, May 1981. (In German.)
1382. Schaumuller-Bichl, "On the Design and Analysis of New Cipher Systems Related to the DES," Technical Report, Linz University, 1983.

1383. A. Scherbius, "Ciphering Machine," U.S. Patent #1,657,411, 24 Jan 1928.
1384. J.I. Schiller, "Secure Distributed Computing," *Scientific American*, v. 271, n. 5, Nov 1994, pp. 72–76.
1385. R. Schlafly, "Complaint Against Exclusive Federal Patent License," Civil Action File No. C-93 20450, United States District Court for the Northern District of California.
1386. B. Schneier, "One-Way Hash Functions," *Dr. Dobb's Journal*, v. 16, n. 9, Sep 1991, pp. 148–151.
1387. B. Schneier, "Data Guardians," *MacWorld*, v. 10, n. 2, Feb 1993, pp. 145–151.
1388. B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 191–204.
1389. B. Schneier, "The Blowfish Encryption Algorithm," *Dr. Dobb's Journal*, v. 19, n. 4, Apr 1994, pp. 38–40.
1390. B. Schneier, *Protect Your Macintosh*, Peachpit Press, 1994.
1391. B. Schneier, "Designing Encryption Algorithms for Real People," *Proceedings of the 1994 ACM SIGSAC New Security Paradigms Workshop*, IEEE Computer Society Press, 1994, pp. 63–71.
1392. B. Schneier, "A Primer on Authentication and Digital Signatures," *Computer Security Journal*, v. 10, n. 2, 1994, pp. 38–40.
1393. B. Schneier, "The GOST Encryption Algorithm," *Dr. Dobb's Journal*, v. 20, n. 1, Jan 95, pp. 123–124.
1394. B. Schneier, *E-Mail Security* (with PGP and PEM) New York: John Wiley & Sons, 1995.
1395. C.P. Schnorr, "On the Construction of Random Number Generators and Random Function Generators," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 225–232.
1396. C.P. Schnorr, "Efficient Signature Generation for Smart Cards," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 239–252.
1397. C.P. Schnorr, "Efficient Signature Generation for Smart Cards," *Journal of Cryptology*, v. 4, n. 3, 1991, pp. 161–174.
1398. C.P. Schnorr, "Method for Identifying Subscribers and for Generating and Verifying Electronic Signatures in a Data Exchange System," U.S. Patent #4,995,082, 19 Feb 1991.
1399. C.P. Schnorr, "An Efficient Cryptographic Hash Function," presented at the rump session of CRYPTO '91, Aug 1991.
1400. C.P. Schnorr, "FFT-Hash II, Efficient Cryptographic Hashing," *Advances in Cryptology—EUROCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 45–54.
1401. C.P. Schnorr and W. Alexi, "RSA-bits are 0.5 + ε Secure," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 113–126.
1402. C.P. Schnorr and S. Vaudenay, "Parallel FFT-Hashing," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 149–156.
1403. C.P. Schnorr and S. Vaudenay, "Black Box Cryptanalysis of Hash Networks Based on Multipermutations," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
1404. W. Schwartau, *Information Warfare: Chaos on the Electronic Superhighway*, New York: Thunders Mouth Press, 1994.
1405. R. Scott, "Wide Open Encryption Design Offers Flexible Implementations," *Cryptologia*, v. 9, n. 1, Jan 1985, pp. 75–90.
1406. J. Seberry, "A Subliminal Channel in Codes for Authentication without Secrecy," *Ars Combinatorica*, v. 19A, 1985, pp. 337–342.
1407. J. Seberry and J. Pieprzyk, *Cryptography: An Introduction to Computer Security*, Englewood Cliffs, N.J.: Prentice-Hall, 1989.
1408. J. Seberry, X.-M. Zhang, and Y. Zheng, "Nonlinearly Balanced Boolean Functions and Their Propagation Characteristics," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1994, pp. 49–60.
1409. H. Sedlack, "The RSA Cryptography Processor: The First High Speed One-Chip Solution," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 95–105.
1410. H. Sedlack and U. Golze, "An RSA Cryptography Processor," *Microprocessing and Microprogramming*, v. 18, 1986, pp. 583–590.
1411. E.S. Selmer, *Linear Recurrence over Finite Field*, University of Bergen, Norway, 1966.

1412. J.O. Shallit, "On the Worst Case of Three Algorithms for Computing the Jacobi Symbol," *Journal of Symbolic Computation*, v. 10, n. 6, Dec 1990, pp. 593–610.

1413. A. Shamir, "A Fast Signature Scheme," MIT Laboratory for Computer Science, Technical Memorandum, MIT/LCS/TM-107, Massachusetts Institute of Technology, Jul 1978.

1414. A. Shamir, "How to Share a Secret," *Communications of the ACM*, v. 24, n. 11, Nov 1979, pp. 612–613.

1415. A. Shamir, "On the Cryptocomplexity of Knapsack Systems," *Proceedings of the 11th ACM Symposium on the Theory of Computing*, 1979, pp. 118–129.

1416. A. Shamir, "The Cryptographic Security of Compact Knapsacks," MIT Library for Computer Science, Technical Memorandum, MIT/LCS/TM-164, Massachusetts Institute of Technology, 1980.

1417. A. Shamir, "On the Generation of Cryptographically Strong Pseudo-Random Sequences," *Lecture Notes in Computer Science 62: 8th International Colloquium on Automata, Languages, and Programming*, Springer-Verlag, 1981.

1418. A. Shamir, "A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem," *Advances in Cryptology: Proceedings of Crypto 82*, Plenum Press, 1983, pp. 279–288.

1419. A. Shamir, "A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem," *Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science*, 1982, pp. 145–152.

1420. A. Shamir, "On the Generation of Cryptographically Strong Pseudo-Random Sequences," *ACM Transactions on Computer Systems*, v. 1, n. 1, Feb 1983, pp. 38–44.

1421. A. Shamir, "A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem," *IEEE Transactions on Information Theory*, v. IT-30, n. 5, Sep 1984, pp. 699–704.

1422. A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, pp. 47–53.

1423. A. Shamir, "On the Security of DES," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 280–281.

1424. A. Shamir, lecture at SECURICOM '89.

1425. A. Shamir, "Efficient Signature Schemes Based on Birational Permutations," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 1–12.

1426. A. Shamir, personal communication, 1993.

1427. A. Shamir and A. Fiat, "Method, Apparatus and Article for Identification and Signature," U.S. Patent #4,748,668, 31 May 1988.

1428. A. Shamir and R. Zippel, "On the Security of the Merkle-Hellman Cryptographic Scheme," *IEEE Transactions on Information Theory*, v. 26, n. 3, May 1980, pp. 339–340.

1429. M. Shand, P. Bertin, and J. Vuillemin, "Hardware Speedups in Long Integer Multiplication," *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, 1990, pp. 138–145.

1430. D. Shanks, *Solved and Unsolved Problems in Number Theory*, Washington D.C.: Spartan, 1962.

1431. C.E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, v. 27, n. 4, 1948, pp. 379–423, 623–656.

1432. C.E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, v. 28, n. 4, 1949, pp. 656–715.

1433. C.E. Shannon, *Collected Papers: Claude Elmwood Shannon*, N.J.A. Sloane and A.D. Wyner, eds., New York: IEEE Press, 1993.

1434. C.E. Shannon, "Predication and Entropy in Printed English," *Bell System Technical Journal*, v. 30, n. 1, 1951, pp. 50–64.

1435. A. Shimizu and S. Miyaguchi, "Fast Data Encipherment Algorithm FEAL," *Transactions of IEICE of Japan*, v. J70-D, n. 7, Jul 87, pp. 1413–1423. (In Japanese.)

1436. A. Shimizu and S. Miyaguchi, "Fast Data Encipherment Algorithm FEAL," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 267–278.

1437. A. Shimizu and S. Miyaguchi, "FEAL—Fast Data Encipherment Algorithm," *Systems and Computers in Japan*, v. 19, n. 7, 1988, pp. 20–34, 104–106.

1438. A. Shimizu and S. Miyaguchi, "Data Randomization Equipment," U.S. Patent #4,850,019, 18 Jul 1989.

1439. M. Shimada, "Another Practical Public-key Cryptosystem," *Electronics Letters*, v. 28, n. 23, 5 Nov 1992, pp. 2146–2147.
1440. K. Shirriff, personal communication, 1993.
1441. H. Shizuya, T. Itoh, and K. Sakurai, "On the Complexity of Hyperelliptic Discrete Logarithm Problem," *Advances in Cryptology—EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 337–351.
1442. Z. Shmuley, "Composite Diffie-Hellman Public-Key Generating Systems Are Hard to Break," Computer Science Department, Technion, Haifa, Israel, Technical Report 356, Feb 1985.
1443. P.W. Shor, "Algorithms for Quantum Computation: Discrete Log and Factoring," *Proceedings of the 35th Symposium on Foundations of Computer Science*, 1994, pp. 124–134.
1444. L. Shroyer, letter to NIST regarding DSS, 17 Feb 1992.
1445. C. Shu, T. Matsumoto, and H. Imai, "A Multi-Purpose Proof System, *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. E75-A, n. 6, Jun 1992, pp. 735–743.
1446. E.H. Sibley, "Random Number Generators: Good Ones Are Hard to Find," *Communications of the ACM*, v. 31, n. 10, Oct 1988, pp. 1192–1201.
1447. V.M. Sidenikov and S.O. Shestakov, "On Encryption Based on Generalized Reed-Solomon Codes," *Diskretnaya Math*, v. 4, 1992, pp. 57–63. [In Russian.]
1448. V.M. Sidenikov and S.O. Shestakov, "On Insecurity of Cryptosystems Based on Generalized Reed-Solomon Codes," unpublished manuscript, 1992.
1449. D.P. Sidhu, "Authentication Protocols for Computer Networks," *Computer Networks and ISDN Systems*, v. 11, n. 4, Apr 1986, pp. 297–310.
1450. T. Siegenthaler, "Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications," *IEEE Transactions on Information Theory*, v. IT-30, n. 5, Sep 1984, pp. 776–780.
1451. T. Siegenthaler, "Decrypting a Class of Stream Ciphers Using Ciphertext Only," *IEEE Transactions on Computing*, v. C-34, Jan 1985, pp. 81–85.
1452. T. Siegenthaler, "Cryptanalyst's Representation of Nonlinearity Filtered *ml*-sequences," *Advances in Cryptology—EUROCRYPT '85*, Springer-Verlag, 1986, pp. 103–110.
1453. R.D. Silverman, "The Multiple Polynomial Quadratic Sieve," *Mathematics of Computation*, v. 48, n. 177, Jan 1987, pp. 329–339.
1454. G.J. Simmons, "Authentication without Secrecy: A Secure Communication Problem Uniquely Solvable by Asymmetric Encryption Techniques," *Proceedings of IEEE EASCON '79*, 1979, pp. 661–662.
1455. G.J. Simmons, "Some Number Theoretic Questions Arising in Asymmetric Encryption Techniques," *Annual Meeting of the American Mathematical Society*, AMS Abstract 763.94.1, 1979, pp. 136–151.
1456. G.J. Simmons, "High Speed Arithmetic Using Redundant Number Systems," *Proceedings of the National Telecommunications Conference*, 1980, pp. 49.3.1–49.3.2.
1457. G.J. Simmons, "A 'Weak' Privacy Protocol Using the RSA Cryptosystem," *Cryptologia*, v. 7, n. 2, Apr 1983, pp. 180–182.
1458. G.J. Simmons, "The Prisoner's Problem and the Subliminal Channel," *Advances in Cryptology: Proceedings of CRYPTO '83*, Plenum Press, 1984, pp. 51–67.
1459. G.J. Simmons, "The Subliminal Channel and Digital Signatures," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 364–378.
1460. G.J. Simmons, "A Secure Subliminal Channel (?)," *Advances in Cryptology—CRYPTO '85 Proceedings*, Springer-Verlag, 1986, pp. 33–41.
1461. G.J. Simmons, "Cryptology," *Encyclopedia Britannica*, 16th edition, 1986, pp. 913–924B.
1462. G.J. Simmons, "How to (Really) Share a Secret," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 390–448.
1463. G.J. Simmons, "Prepositioned Secret Sharing Schemes and/or Shared Control Schemes," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 436–467.
1464. G.J. Simmons, "Geometric Shares Secret and/or Shared Control Schemes," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 216–241.
1465. G.J. Simmons, ed., *Contemporary Cryptology: The Science of Information Integrity*, IEEE Press, 1992.

1466. G.J. Simmons, "An Introduction to Shared Secret and/or Shared Control Schemes and Their Application," in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 441–497.

1467. G.J. Simmons, "How to Insure that Data Acquired to Verify Treaty Compliance Are Trustworthy," in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 615–630.

1468. G.J. Simmons, "The Subliminal Channels of the U.S. Digital Signature Algorithm (DSA)," *Proceedings of the Third Symposium on: State and Progress of Research in Cryptography*, Rome: Fondazone Ugo Bordoni, 1993, pp. 35–54.

1469. G.J. Simmons, "Subliminal Communication is Easy Using the DSA," *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 218–232.

1470. G.J. Simmons, "An Introduction to the Mathematics of Trust in Security Protocols," *Proceedings: Computer Security Foundations Workshop VI*, IEEE Computer Society Press, 1993, pp. 121–127.

1471. G.J. Simmons, "Protocols that Ensure Fairness," *Codes and Ciphers*, Institute of Mathematics and its Applications, 1995, pp. 383–394.

1472. G.J. Simmons, "Cryptanalysis and Protocol Failures," *Communications of the ACM*, v. 37, n. 11, Nov 1994, pp. 56–65.

1473. G.J. Simmons, "Subliminal Channels: Past and Present," *European Transactions on Telecommuncations*, v. 4, n. 4, Jul/Aug 1994, pp. 459–473.

1474. G.J. Simmons and M.J. Norris, *How to Cipher Fast Using Redundant Number Systems*, SAND-80-1886, Sandia National Laboratories, Aug 1980.

1475. A. Sinkov, *Elementary Cryptanalysis*, Mathematical Association of America, 1966.

1476. R. Siromoney and L. Matthew, "A Public Key Cryptosystem Based on Lyndon Words," *Information Processing Letters*, v. 35, n. 1, 15 Jun 1990, pp. 33–36.

1477. B. Smeets, "A Note on Sequences Generated by Clock-Controlled Shift Registers," *Advances in Cryptology—EUROCRYPT '85*, Springer-Verlag, 1986, pp. 40–42.

1478. M.E. Smid, "A Key Notarization System for Computer Networks," NBS Special Report 500-54, U.S. Department of Commerce, Oct 1979.

1479. M.E. Smid, "The DSS and the SHS," *Federal Digital Signature Applications Symposium*, Rockville, MD, 17–18 Feb 1993.

1480. M.E. Smid and D.K. Branstad, "The Data Encryption Standard: Past and Future," *Proceedings of the IEEE*, v. 76, n. 5., May 1988, pp. 550–559.

1481. M.E. Smid and D.K. Branstad, "The Data Encryption Standard: Past and Future," in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 43–64.

1482. J.L. Smith, "The Design of Lucifer, A Cryptographic Device for Data Communications," IBM Research Report RC3326, 1971.

1483. J.L. Smith, "Recirculating Block Cipher Cryptographic System," U.S. Patent #3,796,830, 12 Mar 1974.

1484. J.L. Smith, W.A. Notz, and P.R. Osseck, "An Experimental Application of Cryptography to a Remotely Accessed Data System," *Proceedings of the ACM Annual Conference*, Aug 1972, pp. 282–290.

1485. K. Smith, "Watch Out Hackers, Public Encryption Chips Are Coming," *Electronics Week*, 20 May 1985, pp. 30–31.

1486. P. Smith, "LUC Public-Key Encryption," *Dr. Dobb's Journal*, v. 18, n. 1, Jan 1993, pp. 44–49.

1487. P. Smith and M. Lennon, "LUC: A New Public Key System," *Proceedings of the Ninth International Conference on Information Security, IFIP/Sec 1993*, North Holland: Elsevier Science Publishers, 1993, pp. 91–111.

1488. E. Snekkenes, "Exploring the BAN Approach to Protocol Analysis," *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, 1991, pp. 171–181.

1489. B. Snow, "Multiple Independent Binary Bit Stream Generator," U.S. Patent #5,237,615, 17 Aug 1993.

1490. R. Solovay and V. Strassen, "A Fast Monte-Carlo Test for Primality," *SIAM Journal on Computing*, v. 6, Mar 1977, pp. 84–85; erratum in ibid, v. 7, 1978, p. 118.

1491. T. Sorimachi, T. Tokita, and M. Matsui, "On a Cipher Evaluation Method Based on Differential Cryptanalysis," *Proceedings of the 1994 Symposium on Cryptography*

and Information Security (SCIS 94), Lake Biwa, Japan, 27–29 Jan 1994, pp. 4C.1–9. (In Japanese.)

1492. A. Sorkin, "Lucifer, a Cryptographic Algorithm," *Cryptologia*, v. 8, n. 1, Jan 1984, pp. 22–41.
1493. W. Stallings, "Kerberos Keeps the Ethernet Secure," *Data Communications*, Oct 1994, pp. 103–111.
1494. W. Stallings, *Network and Internetwork Security*, Englewood Cliffs, N.J.: Prentice-Hall, 1995.
1495. W. Stallings, *Protect Your Privacy: A Guide for PGP Users*, Englewood Cliffs, N.J.: Prentice-Hall, 1995.
1496. Standards Association of Australia, "Australian Standard 2805.4 1985: Electronic Funds Transfer—Requirements for Interfaces: Part 4—Message Authentication," SAA, North Sydney, NSW, 1985.
1497. Standards Association of Australia, "Australian Standard 2805.5 1985: Electronic Funds Transfer—Requirements for Interfaces: Part 5—Data Encipherment Algorithm," SAA, North Sydney, NSW, 1985.
1498. Standards Association of Australia, "Australian Standard 2805.5.3: Electronic Data Transfer—Requirements for Interfaces: Part 5.3—Data Encipherment Algorithm 2," SAA, North Sydney, NSW, 1992.
1499. J.G. Steiner, B.C. Neuman, and J.I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," *USENIX Conference Proceedings*, Feb 1988, pp. 191–202.
1500. J. Stern, "Secret Linear Congruential Generators Are Not Cryptographically Secure," *Proceedings of the 28th Symposium on Foundations of Computer Science*, 1987, pp. 421–426.
1501. J. Stern, "A New Identification Scheme Based on Syndrome Decoding," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 13–21.
1502. A. Stevens, "Hacks, Spooks, and Data Encryption," *Dr. Dobb's Journal*, v. 15, n. 9, Sep 1990, pp. 127–134, 147–149.
1503. R. Struik, "On the Rao-Nam Private-Key Cryptosystem Using Non-Linear Codes," *IEEE 1991 Symposium on Information Theory*, Budapest, Hungary, 1991.
1504. R. Struik and J. van Tilburg, "The Rao-Nam Scheme Is Insecure against a Chosen-Plaintext Attack," *Advances in Cryptology—CRYPTO '87 Proceedings*, Springer-Verlag, 1988, pp. 445–457.
1505. S.G. Stubblebine and V.G. Glinor, "Protecting the Integrity of Privacy-Enhanced Mail with DES-Based Authentication Codes," *Proceedings of the Privacy and Security Research Group 1993 Workshop on Network and Distributed System Security*, The Internet Society, 1993, pp. 75–80.
1506. R. Sugarman, "On Foiling Computer Crime," *IEEE Spectrum*, v. 16, n. 7, Jul 79, pp. 31–32.
1507. H.N. Sun and T. Hwang, "Public-key ID-Based Cryptosystem," *Proceedings of the 25th Annual 1991 IEEE International Carnahan Conference on Security Technology*, Taipei, Taiwan, 1–3 Oct 1991, pp. 142–144.
1508. P.F. Syverson, "Formal Semantics for Logics of Computer Protocols," *Proceedings of the Computer Security Foundations Workshop III*, IEEE Computer Society Press, 1990, pp. 32–41.
1509. P.F. Syverson, "The Use of Logic in the Analysis of Cryptographic Protocols," *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, 1991, pp. 156–170.
1510. P.F. Syverson, "Knowledge, Belief, and Semantics in the Analysis of Cryptographic Protocols," *Journal of Computer Security*, v. 1, n. 3, 1992, pp. 317–334.
1511. P.F. Syverson, "Adding Time to a Logic Authentication," *1st ACM Conference on Computer and Communications Security*, ACM Press, 1993, pp. 97–106.
1512. P.F. Syverson and C.A. Meadows, "A Logical Language for Specifying Cryptographic Protocol Requirements," *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, 1993, pp. 14–28.
1513. P.F. Syverson and C.A. Meadows, "Formal Requirements for Key Distribution Protocols," *Advances in Cryptology—EUROCRYPT '94 Proceedings*, Springer-Verlag, 1995, to appear.
1514. P.F. Syverson and P.C. van Oorschot, "On Unifying Some Cryptographic Protocol Logics," *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, 1994, pp. 165–177.
1515. H. Tanaka, "A Realization Scheme for the Identity-Based Cryptosystem," *Advances*

in Cryptology—CRYPTO '87 Proceedings, Springer-Verlag, 1988, pp. 340–349.

1516. H. Tanaka, "A Realization Scheme for the Identity-Based Cryptosystem," *Electronics and Communications in Japan, Part 3 (Fundamental Electronic Science)*, v. 73, n. 5, May 1990, pp. 1–7.

1517. H. Tanaka, "Identity-Based Noninteractive Common-Key Generation and Its Application to Cryptosystems," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, v. J75-A, n. 4, Apr 1992, pp. 796–800.

1518. J. Tardo and K. Alagappan, "SPX: Global Authentication Using Public Key Certificates," *Proceedings of the 1991 IEEE Computer Society Symposium on Security and Privacy*, 1991, pp. 232–244.

1519. J. Tardo, K. Alagappan, and R. Pitkin, "Public Key Based Authentication Using Internet Certificates," *USENIX Security II Workshop Proceedings*, 1990, pp. 121–123.

1520. A. Tardy-Corfdir and H. Gilbert, "A Known Plaintext Attack of FEAL-4 and FEAL-6," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 172–182.

1521. M. Tatebayashi, N. Matsuzaki, and D.B. Newman, "Key Distribution Protocol for Digital Mobile Communication System," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 324–333.

1522. M. Taylor, "Implementing Privacy Enhanced Mail on VMS," *Proceedings of the Privacy and Security Research Group 1993 Workshop on Network and Distributed System Security*, The Internet Society, 1993, pp. 63–68.

1523. R. Taylor, "An Integrity Check Value Algorithm for Stream Ciphers," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 40–48.

1524. T. Tedrick, "Fair Exchange of Secrets," *Advances in Cryptology: Proceedings of CRYPTO '84*, Springer-Verlag, 1985, pp. 434–438.

1525. R. Terada and P.G. Pinheiro, "How to Strengthen FEAL against Differential Cryptanalysis," *Proceedings of the 1995 Japan-Korea Workshop on Information Security and Cryptography*, Inuyama, Japan, 24–27 Jan 1995, pp. 153–162.

1526. J.-P. Tillich and G. Zémor, "Hashing with SI_2," *Advances in Cryptology—CRYPTO '94 Proceedings*, Springer-Verlag, 1994, pp. 40–49.

1527. T. Tokita, T. Sorimachi, and M. Matsui, "An Efficient Search Algorithm for the Best Expression on Linear Cryptanalysis," IEICE Japan, Technical Report, ISEC93-97, 1994.

1528. M. Tompa and H. Woll, "Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information," *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science*, 1987, pp. 472–482.

1529. M. Tompa and H. Woll, "How to Share a Secret with Cheaters," *Journal of Cryptology*, v. 1, n. 2, 1988, pp. 133–138.

1530. M.-J. Toussaint, "Verification of Cryptographic Protocols," Ph.D. dissertation, Université de Liège, 1991.

1531. M.-J. Toussaint, "Deriving the Complete Knowledge of Participants in Cryptographic Protocols," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 24–43.

1532. M.-J. Toussaint, "Separating the Specification and Implementation Phases in Cryptology," *ESORICS 92, Proceedings of the Second European Symposium on Research in Computer Security*, Springer-Verlag, 1992, pp. 77–101.

1533. P.D. Townsend, J.G. Rarity, and P.R. Tapster, "Enhanced Single Photon Fringe Visibility in a 10 km-Long Prototype Quantum Cryptography Channel," *Electronics Letters*, v. 28, n. 14, 8 Jul 1993, pp. 1291–1293.

1534. S.A. Tretter, "Properties of PN^2 Sequences," *IEEE Transactions on Information Theory*, v. IT-20, n. 2, Mar 1974, pp. 295–297.

1535. H. Truman, "Memorandum for: The Secretary of State, The Secretary of Defense," A 20707 5/4/54/OSO, NSA TS CONTL. NO 73-00405, 24 Oct 1952.

1536. Y.W. Tsai and T. Hwang, "ID Based Public Key Cryptosystem Based on Okamoto and Tanaka's ID Based One-Way Communications Scheme," *Electronics Letters*, v. 26, n. 10, 1 May 1990, pp. 666–668.

1537. G. Tsudik, "Message Authentication with One-Way Hash Functions," *ACM Computer Communications Review*, v. 22, n. 5, 1992, pp. 29–38.

1538. S. Tsujii and K. Araki, "A Rebuttal to Coppersmith's Attacking Method," memorandum presented at Crypto '94, Aug 1994.

1539. S. Tsujii, K. Araki, J. Chao, T. Sekine, and Y. Matsuzaki, "ID-Based Key Sharing Scheme—Cancellation of Random Numbers by Iterative Addition," IEICE Japan, Technical Report, ISEC 92-47, Oct 1992.

1540. S. Tsujii, K. Araki, and T. Sekine, "A New Scheme of Noninteractive ID-Based Key Sharing with Explosively High Degree of Separability," Technical Report, Department of Computer Science, Tokyo Institute of Technology, 93TR-0016, May 1993.

1541. S. Tsujii, K. Araki, and T. Sekine, "A New Scheme of Non Interactive ID-Based key Sharing with Explosively High Degree of Separability (Second Version)," Technical Report, Department of Computer Science, Tokyo Institute of Technology, 93TR-0020, Jul 1993.

1542. S. Tsujii, K. Araki, T. Sekine, and K. Tanada, "A New Scheme of Non Interactive ID-Based Key Sharing with Explosively High Degree of Separability," *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 49–58.

1543. S. Tsujii, K. Araki, H. Tanaki, J. Chao, T. Sekine, and Y. Matsuzaki, "ID-Based Key Sharing Scheme—Reply to Tanaka's Comment," IEICE Japan, Technical Report, ISEC 92-60, Dec 1992.

1544. S. Tsujii and J. Chao, "A New ID-based Key Sharing System," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 288–299.

1545. S. Tsujii, J. Chao, and K. Araki, "A Simple ID-Based Scheme for Key Sharing," IEICE Japan, Technical Report, ISEC 92-25, Aug 1992.

1546. S. Tsujii and T. Itoh, "An ID-Based Cryptosystem Based on the Discrete Logarithm Problem," *IEEE Journal on Selected Areas in Communication*, v. 7, n. 4, May 1989, pp. 467–473.

1547. S. Tsujii and T. Itoh, "An ID-Based Cryptosystem Based on the Discrete Logarithm Problem," *Electronics Letters*, v. 23, n. 24, Nov 1989, pp. 1318–1320.

1548. S. Tsujii, K. Kurosawa, T. Itoh, A. Fujioka, and T. Matsumoto, "A Public-Key Cryptosystem Based on the Difficulty of Solving a System of Non-Linear Equations," TSUJII Laboratory Technical Memorandum, n. 1, 1986.

1549. Y. Tsunoo, E. Okamoto, and H. Doi, "Analytical Known Plain-Text Attack for FEAL-4 and Its Improvement," *Proceedings of the 1994 Symposium on Cryptography and Information Security (SCIS 93)*, 1993.

1550. Y. Tsunoo, E. Okamoto, T. Uyematsu, and M. Mambo, "Analytical Known Plain-Text Attack for FEAL-6" *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 253–261.

1551. W. Tuchman, "Hellman Presents No Shortcut Solutions to DES," *IEEE Spectrum*, v. 16, n. 7, July 1979, pp. 40–41.

1552. U.S. Senate Select Committee on Intelligence, "Unclassified Summary: Involvement of NSA in the Development of the Data Encryption Standard," *IEEE Communications Magazine*, v. 16, n. 6, Nov 1978, pp. 53–55.

1553. B. Vallée, M. Girault, and P. Toffin, "How to Break Okamoto's Cryptosystem by Reducing Lattice Values," *Advances in Cryptology—EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, p. 281–291.

1554. H. Van Antwerpen, "Electronic Cash," Master's thesis, CWI, Netherlands, 1990.

1555. K. Van Espen and J. Van Mieghem, "Evaluatie en Implementatie van Authentiseringsalgoritmen," graduate thesis, ESAT Laboratorium, Katholieke Universiteit Leuven, 1989. (In Dutch.)

1556. P.C. van Oorschot, "Extending Cryptographic Logics of Belief to Key Agreement Protocols," *Proceedings of the 1st Annual ACM Conference on Computer and Communications Security*, 1993, pp. 232–243.

1557. P.C. van Oorschot, "An Alternate Explanation for Two BAN-logic 'Failures,'" *Advances in Cryptology—EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 443–447.

1558. P.C. van Oorschot and M.J. Wiener, "A Known-Plaintext Attack on Two-Key Triple Encryption," *Advances in Cryptology—EUROCRYPT '90 Proceedings*, Springer-Verlag, 1991, pp. 318–325.

1559. J. van Tilburg, "On the McEliece Cryptosystem," *Advances in Cryptology—*

CRYPTO '88 Proceedings, Springer-Verlag, 1990, pp. 119–131.

1560. J. van Tilburg, "Cryptanalysis of the Xin-mei Digital Signature Scheme," *Electronics Letters*, v. 28, n. 20, 24 Sep 1992, pp. 1935–1938.

1561. J. van Tilburg, "Two Chosen-Plaintext Attacks on the Li Wang Joing Authentication and Encryption Scheme," *Applied Algebra, Algebraic Algorithms and Error Correcting Codes 10*, Springer-Verlag, 1993, pp. 332–343.

1562. J. van Tilburg, "Security-Analysis of a Class of Cryptosystems Based on Linear Error-Correcting Codes," Ph.D. dissertation, Technical University Eindhoven, 1994.

1563. A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer, and P.G. Jespers, "A Single Chip 1024 Bits RSA Processor," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 219–236.

1564. J. Vanderwalle, D. Chaum, W. Fumy, C. Jansen, P. Landrock, and G. Roelofsen, "A European Call for Cryptographic Algorithms: RIPE; RACE Integrity Primitives Evaluation," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 267–271.

1565. V. Varadharajan, "Verification of Network Security Protocols," *Computers and Security*, v. 8, n. 8, Aug 1989, pp. 693–708.

1566. V. Varadharajan, "Use of a Formal Description Technique in the Specification of Authentication Protocols," *Computer Standards and Interfaces*, v. 9, 1990, pp. 203–215.

1567. S. Vaudenay, "FFT-Hash-II Is not Yet Collision-Free," *Advances in Cryptology—CRYPTO '92 Proceedings*, Springer-Verlag, pp. 587–593.

1568. S. Vaudenay, "Differential Cryptanalysis of Blowfish," unpublished manuscript, 1995.

1569. U.V. Vazirani and V.V. Vazirani, "Trapdoor Pseudo-Random Number Generators with Applications to Protocol Design," *Proceedings of the 24th IEEE Symposium on the Foundations of Computer Science*, 1983, pp. 23–30.

1570. U.V. Vazirani and V.V. Vazirani, "Efficient and Secure Pseudo-Random Number Generation," *Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science*, 1984, pp. 458–463.

1571. U.V. Vazirani and V.V. Vazirani, "Efficient and Secure Pseudo-Random Number Generation," *Advances in Cryptology: Proceedings of CRYPTO '84*, Springer-Verlag, 1985, pp. 193–202.

1572. I. Verbauwhede, F. Hoornaert, J. Vanderwalle, and H. De Man, "ASIC Cryptographical Processor Based on DES," *Euro ASIC '91 Proceedings*, 1991, pp. 292–295.

1573. I. Verbauwhede, F. Hoornaert, J. Vanderwalle, H. De Man, and R. Govaerts, "Security Considerations in the Design and Implementation of a New DES Chip," *Advances in Cryptology—EUROCRYPT '87 Proceedings*, Springer-Verlag, 1988, pp. 287–300.

1574. R. Vogel, "On the Linear Complexity of Cascaded Sequences," *Advances in Cryptology: Proceedings of EUROCRYPT 84*, Springer-Verlag, 1985, pp. 99–109.

1575. S. von Solms and D. Naccache, "On Blind Signatures and Perfect Crimes," *Computers & Security*, v. 11, 1992, pp. 581–583.

1576. V.L. Voydock and S.T. Kent, "Security Mechanisms in High-Level Networks," *ACM Computing Surveys*, v. 15, n. 2, Jun 1983, pp. 135–171.

1577. N.R. Wagner, P.S. Putter, and M.R. Cain, "Large-Scale Randomization Techniques," *Advances in Cryptology—CRYPTO '86 Proceedings*, Springer-Verlag, 1987, pp. 393–404.

1578. M. Waidner and B. Pfitzmann, "The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, p. 690.

1579. S.T. Walker, "Software Key Escrow—A Better Solution for Law Enforcement's Needs?" TIS Report #533, Trusted Information Systems, Aug 1994.

1580. S.T. Walker, "Thoughts on Key Escrow Acceptability," TIS Report #534D, Trusted Information Systems, Nov 1994.

1581. S.T. Walker, S.B. Lipner, C.M. Ellison, D.K. Branstad, and D.M. Balenson, "Commercial Key Escrow—Something for Everyone—Now and for the Future," TIS Report #541, Trusted Information Systems, Jan 1995.

1582. M.Z. Wang and J.L. Massey, "The Characteristics of All Binary Sequences with Perfect Linear Complexity Profiles,"

Abstracts of Papers, EUROCRYPT '86, 20–22 May 1986.

1583. E.J. Watson, "Primitive Polynomials (Mod 2)," *Mathematics of Computation,* v. 16, 1962, p. 368.

1584. P. Wayner, "Mimic Functions," *Cryptologia,* v. 16, n. 3, Jul 1992, pp. 193–214.

1585. P. Wayner, "Mimic Functions and Tractability," draft manuscript, 1993.

1586. A.F. Webster and S.E. Tavares, "On the Design of S-Boxes," *Advances in Cryptology—CRYPTO '85 Proceedings,* Springer-Verlag, 1986, pp. 523–534.

1587. G. Welchman, *The Hut Six Story: Breaking the Enigma Codes,* New York: McGraw-Hill, 1982.

1588. A.L. Wells Jr., "A Polynomial Form for Logarithms Modulo a Prime," *IEEE Transactions on Information Theory,* Nov 1984, pp. 845–846.

1589. D.J. Wheeler, "A Bulk Data Encryption Algorithm," *Fast Software Encryption, Cambridge Security Workshop Proceedings,* Springer-Verlag, 1994, pp. 127–134.

1590. D.J. Wheeler, personal communication, 1994.

1591. D.J. Wheeler and R. Needham, "A Large Block DES-Like Algorithm," Technical Report 355, "Two Cryptographic Notes," Computer Laboratory, University of Cambridge, Dec 1994, pp. 1–3.

1592. D.J. Wheeler and R. Needham, "TEA, A Tiny Encryption Algorithm," Technical Report 355, "Two Cryptographic Notes," Computer Laboratory, University of Cambridge, Dec 1994, pp. 1–3.

1593. S.R. White, "Covert Distributed Processing with Computer Viruses," *Advances in Cryptology—CRYPTO '89 Proceedings,* Springer-Verlag, 1990, pp. 616–619.

1594. White House, Office of the Press Secretary, "Statement by the Press Secretary," 16 Apr 1993.

1595. B.A. Wichman and I.D. Hill, "An Efficient and Portable Pseudo-Random Number Generator," *Applied Statistics,* v. 31, 1982, pp. 188–190.

1596. M.J. Wiener, "Cryptanalysis of Short RSA Secret Exponents," *IEEE Transactions on Information Theory,* v. 36, n. 3, May 1990, pp. 553–558.

1597. M.J. Wiener, "Efficient DES Key Search," presented at the rump session of CRYPTO '93, Aug 1993.

1598. M.J. Wiener, "Efficient DES Key Search," TR-244, School of Computer Science, Carleton University, May 1994.

1599. M.V. Wilkes, *Time-Sharing Computer Systems,* New York: American Elsevier, 1968.

1600. E.A. Williams, *An Invitation to Cryptograms,* New York: Simon and Schuster, 1959.

1601. H.C. Williams, "A Modification of the RSA Public-Key Encryption Procedure," *IEEE Transactions on Information Theory,* v. IT-26, n. 6, Nov 1980, pp. 726–729.

1602. H.C. Williams, "An Overview of Factoring," *Advances in Cryptology: Proceedings of Crypto 83,* Plenum Press, 1984, pp. 71–80.

1603. H.C. Williams, "Some Public-Key Crypto-Functions as Intractable as Factorization," *Advances in Cryptology: Proceedings of CRYPTO 84,* Springer-Verlag, 1985, pp. 66–70.

1604. H.C. Williams, "Some Public-Key Crypto-Functions as Intractable as Factorization," *Cryptologia,* v. 9, n. 3, Jul 1985, pp. 223–237.

1605. H.C. Williams, "An M^3 Public-Key Encryption Scheme," *Advances in Cryptology—CRYPTO '85,* Springer-Verlag, 1986, pp. 358–368.

1606. R.S. Winternitz, "Producing One-Way Hash Functions from DES," *Advances in Cryptology: Proceedings of Crypto 83,* Plenum Press, 1984, pp. 203–207.

1607. R.S. Winternitz, "A Secure One-Way Hash Function Built from DES," *Proceedings of the 1984 Symposium on Security and Privacy,* 1984, pp. 88–90.

1608. S. Wolfram, "Random Sequence Generation by Cellular Automata," *Advances in Applied Mathematics,* v. 7, 1986, pp. 123–169.

1609. S. Wolfram, "Cryptography with Cellular Automata," *Advances in Cryptology—CRYPTO '85 Proceedings,* Springer-Verlag, 1986, pp. 429–432.

1610. T.Y.C. Woo and S.S. Lam, "Authentication for Distributed Systems," *Computer,* v. 25, n. 1, Jan 1992, pp. 39–52.

1611. T.Y.C. Woo and S.S. Lam, " 'Authentication' Revisited," *Computer,* v. 25, n. 3, Mar 1992, p. 10.

1612. T.Y.C. Woo and S.S. Lam, "A Semantic Model for Authentication Protocols," *Proceedings of the 1993 IEEE Computer Soci-*

ety Symposium on Research in Security and Privacy, 1993, pp. 178–194.

1613. M.C. Wood, technical report, Cryptech, Inc., Jamestown, NY, Jul 1990.
1614. M.C. Wood, "Method of Cryptographically Transforming Electronic Digital Data from One Form to Another," U.S. Patent #5,003,596, 26 Mar 1991.
1615. M.C. Wood, personal communication, 1993.
1616. C.K. Wu and X.M. Wang, "Determination of the True Value of the Euler Totient Function in the RSA Cryptosystem from a Set of Possibilities," *Electronics Letters*, v. 29, n. 1, 7 Jan 1993, pp. 84–85.
1617. M.C. Wunderlich, "Recent Advances in the Design and Implementation of Large Integer Factorization Algorithms," *Proceedings of 1983 Symposium on Security and Privacy*, IEEE Computer Society Press, 1983, pp. 67–71.
1618. Xerox Network System (XNS) Authentication Protocol, XSIS 098404, Xerox Corporation, Apr 1984.
1619. Y.Y. Xian, "New Public Key Distribution System," *Electronics Letters*, v. 23, n. 11, 1987, pp. 560–561.
1620. L.D. Xing and L.G. Sheng, "Cryptanalysis of New Modified Lu-Lee Cryptosystems," *Electronics Letters*, v. 26, n. 19, 13 Sep 1990, p. 1601–1602.
1621. W. Xinmei, "Digital Signature Scheme Based on Error-Correcting Codes," *Electronics Letters*, v. 26, n. 13, 21 Jun 1990, p. 898–899.
1622. S.B. Xu, D.K. He, and X.M. Wang, "An Implementation of the GSM General Data Encryption Algorithm A5," *CHINACRYPT '94*, Xidian, China, 11–15 Nov 1994, pp. 287–291. (In Chinese.)
1623. M. Yagisawa, "A New Method for Realizing Public-Key Cryptosystem," *Cryptologia*, v. 9, n. 4, Oct 1985, pp. 360–380.
1624. C.H. Yang, "Modular Arithmetic Algorithms for Smart Cards," IEICE Japan, Technical Report, ISEC92-16, 1992.
1625. C.H. Yang and H. Morita, "An Efficient Modular-Multiplication Algorithm for Smart-Card Software Implementation," IEICE Japan, Technical Report, ISEC91-58, 1991.
1626. J.H. Yang, K.C. Zeng, and Q.B. Di, "On the Construction of Large S-Boxes," *CHINACRYPT '94*, Xidian, China, 11–15 Nov 1994, pp. 24–32. (In Chinese.)
1627. A.C.-C. Yao, "Protocols for Secure Computations," *Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science*, 1982, pp. 160–164.
1628. B. Yee, "Using Secure Coprocessors," Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, May 1994.
1629. S.-M. Yen, "Design and Computation of Public Key Cryptosystems," Ph.D. dissertation, National Cheng Hung University, Apr 1994.
1630. S.-M. Yen and C.-S. Lai, "New Digital Signature Scheme Based on the Discrete Logarithm," *Electronics Letters*, v. 29, n. 12, 1993, pp. 1120–1121.
1631. K. Yiu and K. Peterson, "A Single-Chip VLSI Implementation of the Discrete Exponential Public-Key Distribution System," *IBM Systems Journal*, v. 15, n. 1, 1982, pp. 102–116.
1632. K. Yiu and K. Peterson, "A Single-Chip VLSI Implementation of the Discrete Exponential Public-Key Distribution System," *Proceedings of Government Microcircuit Applications Conference*, 1982, pp. 18–23.
1633. H.Y. Youm, S.L. Lee, and M.Y. Rhee, "Practical Protocols for Electronic Cash," *Proceedings of the 1993 Korea-Japan Workshop on Information Security and Cryptography*, Seoul, Korea, 24–26 Oct 1993, pp. 10–22.
1634. M. Yung, "Cryptoprotocols: Subscriptions to a Public Key, the Secret Blocking, and the Multi-Player Mental Poker Game," *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 1985, 439–453.
1635. G. Yuval, "How to Swindle Rabin," *Cryptologia*, v. 3, n. 3, Jul 1979, pp. 187–190.
1636. K.C. Zeng and M. Huang, "On the Linear Syndrome Method in Cryptanalysis," *Advances in Cryptology—CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 469–478.
1637. K.C. Zeng, M. Huang, and T.R.N. Rao, "An Improved Linear Algorithm in Cryptanalysis with Applications," *Advances in Cryptology—CRYPTO '90 Proceedings*, Springer-Verlag, 1991, pp. 34–47.
1638. K.C. Zeng, C.-H. Yang, and T.R.N. Rao, "On the Linear Consistency Test (LCT) in Cryptanalysis with Applications," *Advances in Cryptology—CRYPTO '89*

Proceedings, Springer-Verlag, 1990, pp. 164–174.

1639. K.C. Zeng, C.-H. Yang, D.-Y. Wei, and T.R.N. Rao, "Pseudorandom Bit Generators in Stream-Cipher Cryptography," *IEEE Computer*, v. 24, n. 2, Feb 1991, pp. 8–17.

1640. M. Zhang, S.E. Tavares, and L.L. Campbell, "Information Leakage of Boolean Functions and Its Relationship to Other Cryptographic Criteria," *Proceedings of the 2nd Annual ACM Conference on Computer and Communications Security*, ACM Press, 1994, pp. 156–165.

1641. M. Zhang and G. Xiao, "A Modified Design Criterion for Stream Ciphers," *CHINACRYPT '94*, Xidian, China, 11–15 Nov 1994, pp. 201–209. (In Chinese.)

1642. Y. Zheng, T. Matsumoto, and H. Imai, "Duality between two Cryptographic Primitives," *Papers of Technical Group for Information Security*, IEICE of Japan, Mar 1989, pp. 47–57.

1643. Y. Zheng, T. Matsumoto, and H. Imai, "Impossibility and Optimality Results in Constructing Pseudorandom Permutations," *Advances in Cryptology—EUROCRYPT '89 Proceedings*, Springer-Verlag, 1990, pp. 412–422.

1644. Y. Zheng, T. Matsumoto, and H. Imai, "On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 461–480.

1645. Y. Zheng, T. Matsumoto, and H. Imai, "Duality between two Cryptographic Primitives," *Proceedings of the 8th International Conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, Springer-Verlag, 1991, pp. 379–390.

1646. Y. Zheng, J. Pieprzyk, and J. Seberry, "HAVAL—A One-Way Hashing Algorithm with Variable Length of Output," *Advances in Crytology—AUSCRYPT '92 Proceedings*, Springer-Verlag, 1993, pp. 83–104.

1647. N. Zierler, "Linear Recurring Sequences," *Journal Soc. Indust. Appl. Math.*, v. 7, n. 1, Mar 1959, pp. 31–48.

1648. N. Zierler, "Primitive Trinomials Whose Degree Is a Mersenne Exponent," *Information and Control*, v. 15, 1969, pp. 67–69.

1649. N. Zierler and J. Brillhart, "On Primitive Trinomials (mod 2)," *Information and Control*, v. 13, n. 6, Dec 1968, pp. 541–544.

1650. N. Zierler and W.H. Mills, "Products of Linear Recurring Sequences," *Journal of Algebra*, v. 27, n. 1, Oct 1973, pp. 147–157.

1651. C. Zimmer, "Perfect Gibberish," *Discover*, v. 13, n. 12, Dec 1992, pp. 92–99.

1652. P.R. Zimmermann, *The Official PGP User's Guide*, Boston: MIT Press, 1995.

1653. P.R. Zimmermann, *PGP Source Code and Internals*, Boston: MIT Press, 1995.

Index

A5, 389, 662–667
Abadi, Martin, 66
Absolute rate, of language, 234
Accreditation, 103
Active attacks, 27
Active cheaters, 27
Adams, Carlisle, 334
Adaptive-chosen-plaintext attack, 6
Addition chaining, 244
Additive generators, 390–392
Adjudicated protocol, 26, 71
Adjudicator, 26
Adleman, Leonard M., 163–164, 467
Adler, Roy, 266
Agnew, G. B., 423
Algebraic structure, DES, 282–283
Algorithm M, 393–394
Algorithms, 2–4, 17
  all-or-nothing disclosure of secrets, 543–546
  Asmuth-Bloom, 529–530
  Barrett's, 244
  Berlekamp-Massey algorithm, 380, 404
  block
    chain mode, 206–207
    choosing, 354–355
    replay, 191–193
  breaking, 8
  CAST, 334–335
  choosing, 214–216
cipher block chaining mode, 193–197, 208–210
cipher block chaining of plaintext difference mode, 208
cipher block chaining with checksum, 207–208
cipher-feedback mode, 200–202, 208–210
cipher mode
  choosing, 208–210
  summary, 209
classes, 217
coin flipping
  using Blum integers, 543
  using exponentiation modulo p, 542–543
  using square roots, 541–542
complexity, 237–239
constant, 238
convertible undeniable signatures, 538–539
counter mode, 205–206, 209
cubic, 238
data compression, 226
designated confirmer signatures, 539–540
Diffie-Hellman, fair, 546–547
digital signatures, 39
exponential, 238
for export, 215–216
extended Euclidean, 246–248
factoring, 256
ISO/IEC 9979 registered, 607
Karnin-Greene-Hellman, 530
Khafre, 317–318
Khufu, 317
linear, 238
linear syndrome, 381
modes, DES, 277–278
multiple block
  cascading, 367–368
  combining, 368
multiple-key public-key cryptography, 527–528
oblivious transfer, 550
one-way accumulators, 543
output-feedback mode, 203–205, 208–210
output feedback with a nonlinear function, 208
plaintext block chaining mode, 208
plaintext feedback mode, 208
polynomial, 238
polynomial-time, 238
probabilistic encryption, 552–554
propagating cipher block chaining mode, 207
public-key, 4–5, 33
quadratic, 238
quantum cryptography, 554–557
restricted, 3
running times, 238–239
secret-sharing algorithms, 528–531
secure multiparty computation, 551–552

Algorithms (*Cont.*)
security, 8–9
self-synchronizing stream cipher, 198–199
stream ciphers, 197–198
subliminal-channel signature, 79
superpolynomial, 238
symmetric, 4
synchronous stream cipher, 202–203
TEA, 346
types, 189
unconditionally secure, 8
undeniable digital signatures, 536–539
using, 213–229
vector scheme, 529
zero-knowledge proofs, 548–550
See also Block ciphers; Stream ciphers
All-or-nothing disclosure of secrets, 96, 543–546
voting with a single central facility, 128–130
Alternating stop-and-go generator, 383, 385, 410–411
American National Standards Institute, DES approval, 267–268
Anderson, Ross, 391
ANDOS, *see* All-or-nothing disclosure of secrets
Anonymous message broadcast, 137–139
ANSI X3.105, 267
ANSI X3.106, 267
ANSI X9.8, 267
ANSI X9.17, 268, 359
key generation, 175
ANSI X9.19, 267
ANSI X9.26, 268
Arbitrated protocol, 23–26
Arbitration, timestamping, 75–76
Arbitrator, 23
document signing with, 35–37
group signatures with, 84–85
AR hash function, 453
Arithmetic, modular, 242–245
Arms Export Control Act, 610
Asmuth-Bloom scheme, 529–530
Association for Computing Machinery, 608
Asymmetric algorithms, *see* Public-key algorithms
Atomic Energy Act, 610
Attack, 5
AT&T Model 3600 Telephone Security Device, 594–595
Authentication, 2, 52–56
DASS, 62
Denning-Sacco protocol, 63
dictionary attacks, 52
ISO framework, 574–577
Kerberos, 60
message, 56
Needham-Schroeder protocol, 58–59
Neuman-Stubblebine protocol, 60–62
Otway-Rees protocol, 59–60
protocols, formal analysis, 65–68
salt, 52–53
Schnorr, 511
SESAME, 572
SKEY, 53
SKID, 55–56
using interlock protocol, 54–55
using one-way functions, 52
using public-key cryptography, 53–54
Wide-Mouth Frog protocol, 56–57
Woo-Lam protocol, 63–64
Yahalom, 57–58
Authenticators, 568
Avalanche effect, 273

Backup keys, 181–182
BAN logic, 66–67
Barrett's algorithm, 244
BaseKing, 346
Basis, polarization measurement, 555
Battista, Leon, 11
BBS generator, 417
add to spelled out, 553–554
Beacons, 64
Bellovin, Steve, 518, 520–521, 571
Bennett, Charles, 555, 557
Berlekamp-Massey algorithm, 380, 404
Bernstein, Dan, 616
Berson, Tom, 441
Best affine approximation attack, 381
Beth-Piper stop-and-go generator, 383–384
Bias, 425
Bidirectional message authentication codes, 457
Biham, Eli, 284–285, 288, 296, 301, 303, 306, 308, 311–312, 314, 316, 319, 354, 361, 434
Bilateral stop-and-go generator, 384–385
Binary trees, 78
Biotechnology, as cryptanalysis tool, 156–157
Birthday attack, 165–166, 430
Bit commitment, 86–88
using one-way functions, 87–88
using pseudo-random-sequence generators, 88
using symmetric cryptography, 86–87
Blakley, George, 72, 529
Blaze, Matt, 346, 364
Blinding factor, 112
Blind signatures, 112–115, 549–550
patents, 115
voting with, 126–127
Blobs, 88
Block algorithms, 4
Block chain mode, 206–207
Block ciphers, 4, 189
Blowfish, 336–339
CA-1.1, 327–328
cascading algorithms, 367–368
CAST, 334–335
CDMF key shortening, 366
choosing algorithms, 354–355
combining algorithms, 368
counter mode, 205–206, 209
Crab, 342–344
CRYPTO-MECCANO, 346
designing, 351
design theory, 346–351
Feistel networks, 347
group structure, 348
S-box, 349–351
simple relations, 347–348
strength against differential and linear cryptanalysis, 348–349
weak keys, 348
double encryption, 357–358
double OFB/counter, 363–364
doubling length, 363

electronic codebook mode, 189–191, 208–210
encryption speeds, 355
FEAL, 308–312
feedback, 193
GOST, 331–334
IDEA, 319–325
iterated, 347
Li-Wang algorithm, 346
LOKI, 314–316
Lucifer, 303–304
Madryga, 304–306
McEliece algorithm, 346
MMB, 325–327
multiple encryption, 357
NewDES, 306–308
Rao-Nam algorithm, 346
RC2, 318–319
RC5, 344–346
REDOC II, 311–313
REDOC III, 313
SAFER K-64, 339–341
security, based on one-way hash functions, 353–354
Skipjack, 328–329
versus stream ciphers, 210–211
SXAL8/MBAL, 344
triple encryption, 358–363
3-Way, 341–342
using one-way hash functions, 351–354
whitening, 366–367
xDES[1], 365–366
Block length, doubling, 363
Block replay, 191–193
Blocks, 4
Blowfish, 336–339, 354, 647–654
Blum, Manuel, 89, 105, 108
Blum, Blum, and Shub generator, 417–418
Blum integers, 253
coin flipping, 543
zero-knowledge proofs, 549
Blum-Micali generator, 416–417
Boolean functions, in S-boxes, 350
Bosselaers, Antoon, 436, 441
Boyar, Joan, 369
Brassard, Gilles, 555, 557
Broadcasting:
anonymous, 137–139
secret, 523–524
Brute-force attack, 8, 151–152
software-based, 154–155
time and cost estimates, 152–154
Bureau of Export Administration, 610–611
Burrows, Michael, 66

CA-1.1, 327–328
Cade algorithm, 500–501
Caesar Cipher, 11
CAFE, 606–607
CALC, 346
Cantwell Bill, 615–616
Capstone, 593–594
Cascade generators, 405
Cascades, Gollmann, 387–388
Cascading:
multiple block algorithms, 367–368
multiple stream ciphers, 419–420
Cash, digital, *see* Digital cash
Cassells, Ian, 381
CAST, 334–335
S-boxes, 349
CBC, *see* Cipher block chaining mode
CCEP, 269, 598–599
CDMF, 366, 574
Cellhash, 446
Cellular automata, 500
Cellular automaton generator, 414
Certificates:
Privacy-Enhanced Mail, 579
public-key, 185–187
X.509, 574–575
Certification authority, 186
Certification path, 576
Certified mail, digital, 122–123
Chaining variables, 436
Chambers, Bill, 385–386
Characteristics, 286–288
Chaum, David, 84, 115, 133, 137, 536, 549
Cheater, 27
sharing secrets with, 531
Chess Grandmaster Problem, 109
Chinese Lottery, 156–157
Chinese remainder theorem, 249–250, 470
Chor-Rivest knapsack, 466
Chosen-ciphertext attack, 6–7, 471–472
Chosen-key attack, 7
Chosen-plaintext attack, 6–7, 359
Chosen-text attack, 7

Cipher:
substitution, 10–12
transposition, 12
Cipher block chaining mode, 193–197, 208–210
DES, 277–278
error extension, 196
error propagation, 195–196
initialization vector, 194
message authentication codes, 456
padding, 195
security, 196–197
self-recovering, 196
triple encryption, 360–361
Cipher block chaining of plaintext difference mode, 208
Cipher block chaining with checksum, 207–208
Cipher-feedback mode, 200–202, 208–210
DES, 277
error propagation, 201–202
initialization vector, 201
Cipher mode:
choosing, 208–210
summary, 208–210
Ciphertext, 1–2
auto key, 198
hiding in ciphertext, 227–228
pairs, differential cryptanalysis, 285
stealing, 191
Ciphertext-only attack, 5–6
Cleartext, *see* Plaintext
Clipper chip, 591–593
Clipper key-escrow, 328
Clipper phone, 594
Clock-controlled generators, 381
Clocking, 381
CoCom, 610
Code, 9
Coefficients, solving for, 248
Coin flipping, 89–92
fair, 541–543
into a well, 92
key generation, 92
using Blum integers, 543
using one-way functions, 90
using public-key cryptography, 90–91
using square roots, 541–542
Collision, 166
Collision-free, 30
Collision-resistance, 429
Combination generator, 381

Combining function, 381
Commercial COMSEC Endorsement Program, 269, 598–599
Commercial Data Masking Facility, 366, 574
Common Cryptographic Architecture, 573–574
Common modulus, dangers of, 493
Common modulus attack, RSA, 472
Communications:
  using public-key cryptography, 31–34
  using symmetric cryptography, 28–29
Communications channels, encryption, 216–220
Communications Setup, 517–518
Complementation property, 281
Complement keys, DES, 281–282
Completely blind signatures, 112–113
Complete set of residues, 242
Complexity-theoretic approach, stream ciphers, 415–418
Complexity theory, 237–242
  algorithms, 237–239
  complexity of problems, 239–241
Compression, 226
Compression function, 431
Compression permutation, 273–274
Compromise, 5
Compromised keys, 182–183
Computational complexity, 237
Computationally secure, 8
Computer algorithms, 17
Computer clock, as random-sequence generator, 424
Computer Security Act of 1987, 600–601
Computing, with encrypted data, 85–86, 540–541
COMSET, 517–518
Conditional Access for Europe, 606–607
Conference key distribution, 524
Confusion, 237, 346–347
Congruent, 242
Connection integer, 403
  feedback with carry shift registers, maximal-period, 406–407
Continued fraction algorithm, 256
Contract signing, simultaneous:
  with an arbitrator, 118
  without an arbitrator
    face-to-face, 118–119
    not face-to-face, 119–120
    using cryptography, 120–122
Control Vector, 180
Convertible undeniable signatures, 538–539
Coppersmith, Don, 94, 266, 280, 283, 293, 398, 457
Coppersmith's algorithm, 263
Correlation attack, 380
Correlation immunity, stream ciphers, 380
Correlations, random-sequence generators, 425
Counter mode, 205–206, 209
Counting coincidences, 14
Crab, 342–344
Credit cards, anonymous, 147
Crepeau, Claude, 555
Crypt{1}, 414
CRYPT{3}, 296
Cryptanalysis, 1, 5–8
  differential, *see* Differential cryptanalysis
  FEAL, 311–312
  GOST, 333–334
  IDEA, 323
  linear, 290–293
  LOKI91, 316
  Madryga, 306
  N-Hash, 434–435
  related-key, 290
  Snefru, 432
  types, 5–7
Cryptanalysts, 1
Crypt Breakers Workbench, 414
Cryptographers, 1
Cryptographic algorithm, *see* Cipher
Cryptographically secure pseudo-random, 45
Cryptographic facility, 562
Cryptographic mode, 189
Cryptographic protection, databases, 73–74
Cryptographic protocol, 22
Cryptography, 1
CRYPTO-LEGGO, 414
Cryptologists, 1
Cryptology, 1
CRYPTO-MECCANO, 346
Cryptosystems, 4
  fair, 97
  finite automaton public-key, 482
  hybrid, 32–34
  security, 234–235
  weak, 97
Cusick, Thomas, 312
Cut and choose, 103
Cypherpunks, 609

Daemen, Joan, 325, 341, 349, 414
Damgard, Ivan, 446
Damm, Arvid Gerhard, 13
Data, encrypted:
  computing with, 85–86, 540–541
  discrete logarithm problem, 540–541
  for storage, 220–222
Databases, cryptographic protection, 73–74
Data complexity, 9
Data Encryption Algorithm, *see* Data Encryption Standard
Data Encryption Standard, 17, 265–301
  adoption, 267–268
  algorithm, brute-force attack efficiency, 152–153
  characteristics, 286–288
  commercial chips, 279
  compared to GOST, 333–334
  compression permutation, 273–274
  CRYPT{3}, 296
  decryption, 277
  description, 270
  DESX, 295
  development, 265–267
  differential cryptanalysis, 284–290
    DES variants, 298
  expansion permutation, 273–275
  final permutation, 277
  generalized, 296–297
  hardware and software implementation, 278–279
  with independent subkeys, 295
  initial permutation, 271
  iterated block cipher, 347
  key transformation, 272–273
  linear cryptanalysis, 290–293
  modes, 277–278

multiple, 294–295
1987 review, 268–269
1993 review, 269–270
outline of algorithm, 270–272
P-boxes
design criteria, 294
permutation, 275, 277
RDES, 297–298
related-key cryptanalysis, 290
RIPE-MAC, 457–458
S-boxes, 349
alternate, 296–298
design criteria, 294
key-dependent, 298, 300, 354
substitution, 274–276
security, 278, 280–285
algebraic structure, 282–283
complement keys, 281–282
current, 300–301
key length, 283–284
number of rounds, 284
possibly weak keys, 281–282
S-box design, 284–285
semiweak keys, 280–281
weak keys, 280–281
s^5DES, 298–299
source code, 623–632
speeds on microprocessors and computers, 279
validation and certification of equipment, 268
Data Exchange Key, 581
Data Keys, 176
Davies, Donald, 562
Davies-Meyer, 448
abreast, 452
modified, 449–450
parallel, 451
tandem, 451–452
Davies-Price, 358
Decoherence, 165
Decryption, 1
DES, 277
key, 3
key-error detection, 179
knapsack algorithms, 465
with a public key, 39
with symmetric algorithm, 4
den Boer, Bert, 434, 436, 441
Denning-Sacco protocol, 63
Dense, 378
Dereferencing keys, 221–222
Derived sequence attack, 381
Designated confirmer signatures, 82–83, 539–540
Desmedt, Yvo, 81
DES, *see* Data Encryption Standard
Destruction:
information, 228–229
of keys, 184–185
DESX, 295
Dictionary attack, 52, 171–173
Differential cryptanalysis, 284–290
attacks against
DES, 288–290
DES variants, 298
Lucifer, 303
extending to higher-order differentials, 293
strength against, block cipher design theory, 348–349
Differential-linear cryptanalysis, 293
Diffie, Whitfield, 31, 37, 122, 216, 283, 419, 461, 501, 565
Diffie-Hellman:
EKE implementation, 519–520
extended, 515
failsafe, 547–548
fair, 546–547
Hughes variant, 515
key exchange without exchanging keys, 515
patents, 516
with three or more parties, 514
Diffie's randomized stream cipher, 419
Diffusion, 237, 346–347
Digital card, properties, 146
Digital cash, 139–147
anonymous, 139
credit cards, 147
money orders, 140
double spending problem, 140–141
off-line systems, 146
on-line systems, 145–146
other protocols, 145–147
perfect crime, 145
practical, 145
secret splitting, 142–145
Digital certified mail, 122–123
Digital Notary System, 78
Digital Signature Algorithm, 17, 483–494
attacks against k, 492
computation time comparison with RSA, 489
criticisms, 484–486
dangers of common modulus, 493
description, 486–488
ElGamal encryption with, 490–491
patents, 493–494
prime generation, 488–490
proposal for NIST standard, 483–486
RSA encryption with, 491
security, 491–492
speed precomputations, 487–488
subliminal channel, 493, 534–536
foiling, 536
variants, 494–495
Digital signatures, 34–41
algorithms, 39
applications, 41
blind, 112–115, 549–550
convertible undeniable signatures, 538–539
converting identification schemes to, 512
definition, 39
designated confirmer signatures, 82–83, 539–540
ElGamal, 476–478
with encryption, 41–44
entrusted undeniable, 82
fail-stop, 85
Fiat-Shamir signature scheme, 507–508
group signatures, 84–85
Guillou-Quisquater signature scheme, 509–510
improved arbitrated solution, 76
key exchange with, 50
multiple, 39–40
Guillou-Quisquater, 510
nonrepudiation, 40
oblivious, 117
protocol, 40
proxy, 83
public-key algorithms, 483–502
Cade algorithm, 500–501
cellular automata, 500
Digital Signature Algorithm, *see* Digital Signature Algorithm
discrete logarithm signature schemes, 496–498
ESIGN, 499–500
GOST digital signature algorithm, 495–496

Digital signatures (*Cont.*)
public-key algorithms (*Cont.*)
Matsumoto-Imai algorithm, 500
Ong-Schnorr-Shamir, 498–499
public-key cryptography, 37–38
attacks against, 43–44
one-way hash functions and, 38–39
resend attack, foiling, 43
RSA, 473–474
Schnorr signature scheme, 511–512
subliminal-free, 80
with symmetric cryptosystems and arbitrator, 35–37
terminology, 39
timestamps, 38
trees, 37
undeniable, 81–82, 536–539
Dining Cryptographers Problem, 137
Discrete logarithm, 245
in finite field, 261–263
zero-knowledge proofs, 548
Discrete Logarithm Problem, 501, 540–541
Discrete logarithm signature schemes, 496–498
Distributed Authentication Security Service, 62
Distributed convertible undeniable signatures, 539
Distributed key management, 187
DNA computing, 163–164
DNRSG, 387
DoD key generation, 175
Double encryption, 357–358
Double OFB/counter, 363–364
Double spending problem, 140–141
Driver-level encryption, 222–223
DSA, *see* Digital Signature Algorithm
Dynamic random-sequence generator, 387

E-box, 273
ECB, *see* Electronic codebook mode
Electronic checks, 146
Electronic codebook mode, 189–191, 208–210
combined with OFB, 364
DES, 277–278
padding, 190–191
triple encryption, 362–363
Electronic coins, 146
Electronic Frontier Foundation, 608
Electronic-funds transfer, DES adoption, 268
Electronic Privacy Information Center, 608
ElGamal, 532–533
EKE implementation, 519
encryption, 478
with DSA, 490–491
patents, 479
signatures, 476–478
speed, 478–479
ElGamal, Taher, 263
Elliptic curve cryptosystems, 480–481
Elliptic curve method, 256
Ellison, Carl, 362
Encoding, 226
Encrypt-decrypt-encrypt mode, 359
Encrypted Key Exchange:
applications, 521–522
augmented, 520–521
basic protocol, 518–519
implementation with
Diffie-Hellman, 519–520
ElGamal, 519
RSA, 519
strengthening, 520
Encryption, 1
communication channels, 216–220
combining link-by-link and end-to-end, 219–221
with compression and error control, 226
data, for storage, 220–222
detection, 226–227
digital signatures with, 41–44
driver-level versus file-level, 222–223
ElGamal, 478
with DSA, 490–491
end-to-end, 217–220
with interleaving, 210–211
key, 3
knapsack algorithms, 464
link-by-link, 216–218
multiple, 357
with a private key, 39
probabilistic, 552–554
RSA, 468
with DSA, 491
with symmetric algorithm, 4
using public key, 5
End-to-end encryption, 217–220
combined with link-by-link, 219–221
Enigma, 13, 414
Entropy, 233–234
Entrusted undeniable signature, 82
Error detection:
during decryption, 179
during transmission, 178
Error extension, cipher block chaining mode, 196
Error propagation:
cipher block chaining mode, 195–196
cipher-feedback mode, 201–202
output-feedback mode, 204
Escrow agencies, 592
Escrowed Encryption Standard, 97, 593
ESIGN, 499–500, 533–534
Euclid's algorithm, 245
Euler totient function, 248–249
Expansion permutation, 273–275, 315
Export:
of algorithms, 215–216, 610–616
foreign, 617
Exportable Protection Device, 389
Export Administration Act, 610
EXPTIME, 241
Extended Euclidean algorithm, 246–248

Factoring, 255–258
general number field sieve, 159–160
long-range predictions, 162
public-key encryption algorithms, 158–159
special number field sieve, 160–161
using quadratic sieve, 159
Factoring Problem, 501
Failsafe:
Diffie-Hellman, 547–548
key escrowing, 98
Fail-stop digital signatures, 85
Fair cryptosystems, 97
Fait-Shamir, 508
FAPKC0, 482
FAPKC1, 482
FAPKC2, 482

FEAL, 308–312
  cryptanalysis, 311–312
  description, 308–10
  patents, 311
Feedback:
  cipher block chaining mode, 193, 195
  internal, output-feedback mode, 203
Feedback function, 373
Feedback shift register, 373
Feedback with carry shift registers, 402–404
  combining generators, 405, 410
  maximal-length, tap sequences, 408–409
  maximal-period, connection integers, 406–407
Feedforward, cipher block chaining mode, 195
Feige, Uriel, 503–504
Feige-Fiat-Shamir, 503–508
  enhancements, 506–507
  identification scheme, 504–505
  simplified, 503–504
Feistel, Horst, 266, 303
Feistel network, 347
  Blowfish, 337
  practically secure, 349
Fermat's little theorem, 248
  Euler's generalization, 248
FFT-Hash, 446
Fiat, Amos, 503–504
Fiat-Shamir signature scheme, 507–508
Fibonacci configuration, 373, 379
Fibonacci shrinking generator, 391
File-level encryption, 222–223
Filter generator, 381
Finite field, 254
  discrete logarithms, 261–263
FIPS PUB 46, 267
FIPS PUB 74, 267
FIPS PUB 81, 267
FIPS PUB 112, 267
Fish, 391
Fixed bit index, 543
Flat keyspace, 176
Flipping coins, *see* Coin flipping
Fortified key negotiation, 522

Galois configuration, linear feedback shift registers, 378–379
Galois field, computing in, 254–255
Garey, Michael, 241
Gatekeeper, 278
Geffe generator, 382–383
General number field sieve, 159–160, 256
General Services Administration, DES adoption, 268
Generators, 253–254
Gifford, 392–393
Gifford, David, 392
Gill, J., 501
Global deduction, 8
Goldwasser, Shafi, 94, 552
Gollmann, Dieter, 386
Gollmann cascade, 387–388
Goodman-McAuley cryptosystem, 466
Goresky, Mark, 404
GOST, 331–334, 354
  source code, 643–647
GOST digital signature algorithm, 495–496
GOST hash function, 454
GOST R 34.10–94, 495
Gosudarstvennyi Standard Soyuza SSR, 331–334
Graham-Shamir knapsacks, 465
Graph isomorphism, 104–105
Greatest common divisor, 245–246
Grossman, Edna, 266
Group signatures, 84–85
Group Special Mobile, 389
Group structure, block ciphers design theory, 348
GSM, 389
Guillou, Louis, 102, 508
Guillou-Quisquater:
  identification scheme, 508–510
  signature scheme, 509–510
Gutmann, Peter, 353
Guy, Richard, 159

Haber, Stuart, 75, 485, 488
Hamiltonian cycles, 105–106
Hard drive, encrypted, providing random access to, 222
Hardware:
  DES implementation, 278–279
  encryption, 223–225
  RSA, 469
Hash functions, *see* One-way hash functions
Hash value, 30
HAVAL, 445–446
Hellman, Martin, 31–32, 37, 262, 283, 293, 358–359, 461–462
Hiding information from an oracle, 86
Historical terms, 9
Homophonic substitution cipher, 10–11
Hughes, 515
Hughes, Eric, 609
Hughes XPD/KPD, 389–390
Hybrid cryptosystems, 32–34, 461

IBC-Hash, 458
IBM Common Cryptographic Architecture, 573–574
IBM secret-key management protocol, 561–562
IDEA, 319–325, 354
  cryptanalysis, 323
  description, 320–322
  modes of operation, 323–325
  overview, 320–321
  patents, 325
  S-boxes, 349
  source code, 637–643
  speed, 322–323
  strength against differential cryptanalysis, 348
  variants, 325
Ideal secrecy, 236
Identification schemes:
  converting to signature schemes, 512
  Feige-Fiat-Shamir, 503–508
  Guillou-Quisquater, 508–510
  Ohta-Okamoto, 508
  Schnorr authentication and signature scheme, 510–512
Identity-based cryptosystems, 115
Ignition key, 564
Import, foreign, 617
Index of coincidence, 14
Information:
  amount, information theory definition, 233
  deduction, 8
  destruction, 228–229
Information-theoretic approach, 418
  stream ciphers, 415

Information theory, 233–237
  cryptosystem security, 234–235
  entropy and uncertainty, 233–234
  in practice, 236–237
  rate of the language, 234
  unicity distance, 235–236
Ingemarsson, Ingemar, 418
Initialization vector:
  cipher block chaining mode, 194
  cipher-feedback mode, 201
  output-feedback mode, 204
Inner-CBC, 360, 363
Insertion attack, synchronous stream ciphers, 203
Instance deduction, 8
Institute of Electrical and Electronics Engineers, 608
Integrated Services Digital Network, 563–565
Integrity, 2
Interactive protocol, 103
Interchange Key, 581
Interleave, 210–211
Interlock protocol, mutual authentication using, 54–55
Internal feedback, 203
International Association for Cryptologic Research, 605
International Standards Organization:
  authentication framework, 574–577
  DES adoption, 268
International Traffic in Arms Regulations, 610–614
Internet, Privacy-Enhanced Mail, 577–584
Introducers, 187
Inverses modulo a number, 246–248
IPES, 319
ISDN, 563–565
ISO 8732, 359
ISO 9796, 472, 474, 486
ISO/IEC 9979, 607
ISO X.509 protocols, 574–577
Iterated block cipher, 347

Jacobi symbol, 252–253
J-algebras, 501
Jam, 414
Jennings generator, 383–384
Johnson, David, 241
Jueneman's methods, 457

Kaliski, Burt, 342
Karn, 351–352
Karn, Phil, 351
Karnin-Greene-Hellman, 530
Kerberos, 60, 566–571
  abbreviations, 567
  authentication steps, 567
  credentials, 568
  getting initial ticket, 569
  getting server tickets, 569–570
  licenses, 571
  model, 566
  requesting services, 570
  security, 571
  Version 4, 570–571
  Version 5 messages, 568
Kerckhoffs, A., 5
Kerckhoffs's assumption, 7
Key, 3
  backup, 181–182
  CDMF shortening, 366
  complement, DES, 281–282
  compromised, 182–183
  controlling usage, 180
  dereferencing, 221–222
  destroying, 184–185
  distribution in large networks, 177
  generating, 170–175
    ANSI X9.17 standard, 175
    DoD, 175
    pass phrases, 174–175
    poor choices, 171–173
    random keys, 173–174
    reduced keyspaces, 170–171
  ISDN, 563–564
  lifetime, 183–184
  possibly weak, DES, 281–282
  semiweak, DES, 280–281
  session, 33, 180
  storing, 180–181
  transferring, 176–177
  transmission, error detection, 178
  updating, 180
  using, 179–180
  verification, 178–179
  weak
    block ciphers design theory, 348
    DES, 280–281
Key and message broadcast, 51–52
Key and message transmission, 51
Key Auto-Key, 202
Keyboard latency, as random-sequence generator, 424–425
Key Certification Authority, 43
Key control vectors, 562
Key distribution:
  anonymous, 94–95
  conference, 524
Key Distribution Center, 43–44
Key-Encryption Keys, 176, 184
Key escrow, 97–100, 181–182, 591
  politics, 98–100
Key exchange, 47–52
  DASS, 62
  Denning-Sacco protocol, 63
  with digital signatures, 50
  interlock protocol, 49–50
  Kerberos, 60
  key and message broadcast, 51–52
  key and message transmission, 51
  man-in-the-middle attack, 48–49
  Needham-Schroeder protocol, 58–59
  Neuman-Stubblebine protocol, 60–62
  Otway-Rees protocol, 59–60
  protocols, formal analysis, 65–68
  with public-key cryptography, 48
  with symmetric cryptography, 47–48
  Wide-Mouth Frog protocol, 56–57
  without exchanging keys, 515
  Woo-Lam protocol, 63–64
  Yahalom, 57–58
Key-exchange algorithms:
  COMSET, 517–518
  conference key distribution and secret broadcasting, 523–525
  Diffie-Hellman, 513–516
  Encrypted Key Exchange, 518–522
  fortified key negotiation, 522
  Shamir's three-pass protocol, 516–517
  station-to-station protocol, 516

Tatebayashi-Matsuzaki-Newman, 524–525
Key generation, using coin flipping, 92
Key length:
comparing symmetric and public-key, 165–166
deciding on, 166–167
DES, 283–284
public-key, 158–165
DNA computing, 163–164
quantum computing, 164–165
recommended lengths, 161–163
symmetric, 151–158
biotechnology as cryptanalysis tool, 156–157
brute-force attack, 151–154
Chinese Lottery, 156–157
neural networks, 155
software-based brute-force attacks, 154–155
thermodynamic limitations on brute-force attacks, 157–158
using viruses to spread cracking program, 155–156
Key management, 169–187
distributed, 187
public-key, 185–187
Key negotiation, fortified, 522
Key notarization, 562
Key revocation certificate, 585
Keyspace, 3
flat, 176
nonlinear, 175–176
reduced, 170–171
Keystream generator, 197–198
counter mode, 206
periodic, 202
Khafre, 317–318, 349
Khufu, 317, 349
Kilian, Joe, 116
Kim, Kwangjo, 298, 350
Kinetic Protection Device, 389–390
Klapper, Andy, 404
Klein, Daniel, 53, 171
Knapsack algorithms, 462–466
decryption, 465
encryption, 464
implementations, 465
patents, 466
public key created from private key, 464
security, 465
superincreasing, 463–464
variants, 465–466
Knapsack problem, 501
Known-plaintext attack, 6–7, 151, 359
Knudsen, Lars, 8, 293, 314, 316, 348–349
Knuth, 393, 501
Koblitz, Neal, 480
Konheim, Alan, 266, 280
Kravitz, David, 493
Kravitz-Reed, 481
KryptoKnight, 571–572

Lagged Fibonacci generators, 390
LaGrange interpolating polynomial scheme, 528–529
Lai, Xuejia, 319, 449
Langford, Susan, 293
Law Enforcement Access Field, 591
Legal issues, 618
Legendre symbol, 251
Lehmann, 259
Lehmann algorithm, 259
Length, shift register, 373
Lenstra, Arjen, 159, 162, 257, 485, 488
LFSR/FCSR summation/parity cascade, 410–411
Lidl, Rudolph, 481
Linear complexity:
profile, 380
stream ciphers, 380
Linear congruential generators, 369–372
combining, 371–372
constants, 370
Linear consistency test, 381
Linear cryptanalysis:
DES, 290–293
strength against, block cipher design theory, 348–349
Linear error-correcting codes, algorithms based on, 480
Linear feedback shift registers, 372–379
Galois, 378–379
primitive polynomials mod 2, 376–377
software, 378–379
stream ciphers using, *see* Stream ciphers
Linear syndrome algorithm, 381
Link-by-link encryption, 216–218
combined with end-to-end, 219–221
Linking protocol, timestamping, 76–77
Li-Wang algorithm, 346
Local deduction, 8
Lock-in, 388
Logarithms, discrete, *see* Discrete logarithm
LOKI, 314–316
S-boxes, 349
source code, 632–637
LOKI Double-Block, 451
Low decryption exponent attack, RSA, 473
Low encryption exponent attack, RSA, 472–473
Luby, Michael, 352
Luby-Rackoff, 352–353
xDES[1], 365
LUC, 481
Lucas number, 481
Luccio-Mazzone, 501
Lucifer, 266, 303–304
Lu-Lee cryptosystem, 466
Lyndon words, 501

MacGuffin, 346
Madryga, W. E., 304
Mafia Fraud, 110
Magic numbers, 423
Manasse, Mark, 159, 257
Man-in-the-middle attack, 48–49
Masks, REDOC II, 312
Massey, James, 319, 339, 386, 418, 449
Master Key, 561
Master Terminal Key, 561
Matsui, Mitsuru, 290–291
Matsumoto-Imai algorithm, 500
Mauborgne, Joseph, 15
Maurer, Ueli, 419
Maurer's randomized stream cipher, 419
Maximal period generator, 369
MBAL, 344
McEliece, Robert, 479
McEliece algorithm, 346, 479–480
MD2, 441
MD3, 446
MD4, 435–436
MD5, 436–441
MDC, 353–354

MDC-2, 452–453
MDC-4, 452–454
MD-strengthening, 431
Meet-in-the-middle attack, 358, 381
Mental poker, 92–95
Merkle, Ralph, 34, 316–318, 358–359, 432, 455, 461–462
Merkle's puzzles, 34
Merritt, Michael, 67, 518, 520–521, 571
Message:
  authentication, 56
  broadcasting, 69
  Privacy-Enhanced Mail, 579–582
  recovery, 497–498
  resending as receipt, 42–43
Message authentication codes, 31, 455–459
  bidirectional, 457
  CBC-MAC, 456
  IBC-Hash, 458
  Jueneman's methods, 457
  message authenticator algorithm, 456–457
  one-way hash functions as, 458–459
  RIPE-MAC, 457–458
  stream ciphers, 459
Message authenticator algorithm, 456–457
Message broadcast, anonymous, 137–139
Message Digest, 435–436
Message Digest Cipher, 353
Message Integrity Check, 578
Message-meaning rule, 66
Message Security Protocol, 584
Meyer, Carl, 266, 278
Meyer, Joseph A., 614
Meyer-Schilling, 452
Micali, Silvio, 94, 508, 546–547, 552
Miller, Gary, 259
Miller, V. S., 480
Mimic functions, 10
Minimum-disclosure proofs, 108
MITRENET, 562–563
Miyaguchi, Shoji, 308
MMB, 325–327
m*n-bit S box, 349
Modular arithmetic, 242–245
Modular Multiplication-based Block cipher, 325–327
Modular reduction, 242
Modulo, inverses, 246–248
Monoalphabetic cipher, 10
Montgomery's method, 244
Moore's Law, 153
m-sequence, 374
MSP, 584
Muller, Winfried, 481
Multiparty unconditionally secure protocols, 137
Multiple-bit generator, 421
Multiple encryption, 357
  quintuple, 366
Multiple Identity Fraud, 111
Multiple-key public-key cryptography, 527–528
Multiple signatures, 39–40
Multiplier, 369
Multispeed inner-product generator, 386–387
Mush, 392
Mutual shrinking generator, 392
MYK-80, 593–594
Mykotronx Clipper chip, 328
MYK-78T, 591–593

Nanoteq, 390
National Bureau of Standards, *see* National Institute of Standards and Technology
National Computer Security Center, 599–600
National Institute of Standards and Technology, 600–603
  DES development, 265–267
  Memorandum of Understanding, 601–603
National Security Agency, 597–599
  DES development, 266–267
  export of cryptography, 614–615
  Memorandum of Understanding, 601–603
  S-box development role, 278, 280
Navy Research Laboratory, protocol analyzer, 67–68
Needham, Roger, 58, 66, 216
Needham-Schroeder protocol, 58–59
Networks, large, key distribution, 177
Neuman-Stubblebine protocol, 60–62
Neural networks, breaking algorithms, 155
NewDES, 306–308
N-Hash, 433–435
Niederreiter, Harald, 501
Niederreiter algorithm, 480
Niemi cryptosystem, 466
Nobauer, Wilfried, 481
Noise, random, using as random-sequence generator, 423–424
Nonce-verification rule, 66
Non-Interactive Key Sharing systems, 115
Nonlinear-feedback shift registers, 412–413
Nonlinear keyspace, 175–176
Nonrepudiation, 2
Notz, Bill, 266
NP-complete problem, 240–242
  graph isomorphism, 104
  knapsack algorithms, 462
  McEliece algorithm, 479
  solving, 163–164
NRL Protocol Analyzer, 67–68
NSDD-145, 268
Nuclear Non-Proliferation Act, 610
Number field sieve, 256
Numbers:
  2-adic, 404
  large, 17–18
Number theory, 242–255
  Barrett's algorithm, 244
  Blum integers, 253
  Chinese remainder theorem, 249–250
  Euclid's algorithm, 245
  Euler totient function, 248–249
  extended Euclidean algorithm, 246–248
  Fermat's little theorem, 248
  Galois field, computing in, 254–255
  generators, 253–254
  greatest common divisor, 245–246
  inverses modulo a number, 246–248
  Jacobi symbol, 252–253
  Legendre symbol, 251
  modular arithmetic, 242–245
  Montgomery's method, 244
  prime numbers, 245
  quadratic residues, 250–251
  solving for coefficients, 248
Nyberg, Kaisa, 348

Oblivious transfer, 116–117, 550

Oblivous signatures, 117
OFB, *see* Output-feedback mode
Ohta, Kazuo, 146, 501
Ohta-Okamoto identification scheme, 508
Okamoto, Tatsuaki, 146, 501
1/p generator, 414
One-time pad, 15–17
  hiding ciphertext in ciphertext, 227–228
One-time tape, 418
One-way accumulators, 95–96, 543
One-way function, 29–30
  authentication using, 52
  bit commitment using, 87–88
  coin flipping using, 90
  trap-door, 158
One-way hash functions, 30–31, 351–354
  background, 429–431
  birthday attacks, 165–166, 430
  choosing, 455
  cipher security, 353–354
  compression function, 431
  encryption speeds, 456
  HAVAL, 445–446
  improved arbitrated solution, 76
  Karn, 351–352
  length, 430–431
  Luby-Rackoff, 352–353
  MD2, 441
  MD3, 446
  MD4, 435–436
  MD5, 436–441
  MD-strengthening, 431
  message authentication codes, 455–459
  Message Digest Cipher, 353–354
  multiple signatures, 40
  N-Hash, 433–435
  RIPE-MD, 445
  Secure Hash Algorithm, 442–445
  signing documents with, 38–39
  Snefru, 432
  as unbiased random-bit generator, 107
  using public-key algorithms, 455
  using symmetric block algorithms, 446–455
    AR hash function, 453
    GOST hash function, 454
    hash length equals block size, 447–449
    LOKI Double-Block, 451
    MDC-2 and MDC-4, 452–454
    modified Davies-Meyer, 449–450
    parallel Davies-Meyer, 451
    Preneel-Bosselaers-Govaerts-Vandewalle, 450
    Quisquater-Girault, 450
    tandem and abreast Davies-Meyer, 451–452
Ong-Schnorr-Shamir, 498–499, 531–532
Orange Book, 599–600
Otway-Rees protocol, 59–60
Outerbridge, Richard, 363
Outer-CBC, 360
Output-feedback mode, 203–205, 208–210
  combined with ECB, 364
  DES, 277
  with a nonlinear function, 208
Overtake, 598
Overwriting, 229

Padding:
  cipher block chaining mode, 195
  electronic codebook mode, 190–191
  MD5, 436
  Secure Hash Algorithm, 442
  triple encryption with, 362
Painvin, Georges, 12
Pass phrases, 174–175
Passive attack, 27
Passive cheaters, 27
Patents, 609–610; *See also* specific algorithms
P-boxes:
  design criteria, 294
  permutation, 275, 277, 316
PEM, *see* Privacy-Enhanced Mail
Perfect secrecy, 235
Period, 11
  shift register, 373
Permutation, 237
  key, DES, 272–273
PES, 319, 324
Pike, 391–392
PKZIP, 394–395
Plaintext, 1–2
Plaintext block chaining mode, 208
Plaintext feedback mode, 208
Plaintext pair, right and wrong pairs, 287
Pless generator, 413–414
p-NEW scheme, 498
Pohlig, Stephen, 262
Pohlig-Hellman encryption scheme, 474
Polarized photons, 555
Pollard's Monte Carlo algorithm, 256
Polyalphabetic substitution cipher, 10–11
Polygram substitution cipher, 10–11
Polynomials:
  degree, shift register length, 374
  dense, 378
  irreducible, 255, 481
  sparse, 378
Pomerance, Carl, 257
Powerline System, 466
Pre-image, 30
Preneel, Bart, 457
Preneel-Bosselaers-Govaerts-Vandewalle, 450
Pretty Good Privacy, 584–587
Price, William, 562
Prime numbers, 245
  generation, 258–261
    DSA, 488–490
    practical considerations, 260–260
  relatively prime, 245
  strong, 261
Primitive, 253
Principal square root, 251
Privacy-Enhanced Mail, 577–584
  certificates, 579
  documents, 578
  messages, 579–582
  RIPEM, 583–584
  security, 582–583
  TIS/PEM, 583
Private key, 5
  creating public key from, 464
  for public-key cryptography, lifetime, 184
Probabilistic encryption, 552–554
Problems:
  complexity, 239–241
  EXPTIME, 241
  hard, 239
  intractable, 239
  PSPACE, 241

Problems (*Cont.*)
tractable, 239
undecidable, 240
See also NP-complete problem
Processing complexity, 9
Product cipher, 347
Proofs of Membership, 111
Propagating cipher block chaining mode, 207
Proposed Encryption Standard, 319
Protocols, 21, 47
adjudicated, 26, 70–71
all-or-nothing disclosure of secrets, 96
analysis, approaches, 65–66
anonymous message broadcast, 137–139
arbitrated, 23–26
attacks against, 27
authentication, 576–577
authentication and key-exchange, formal analysis, 65–68
BAN logic, 66–67
basic zero-knowledge, 102–104
bit commitment, 86–88
blind signatures, 112–115
characteristics, 21
cryptographic, 22
DASS, 62
definition, 21
Denning-Sacco, 63
digital cash, *see* Digital cash
digital certified mail, 122–123
digital signatures, 40
distributed, timestamping, 77–78
fair coin flips, 89–92
IBM Common Cryptographic Architecture, 573–574
IBM secret-key management, 561–562
identity-based public-key cryptography, 115
interactive, 103
interlock, 49–50, 54–55
Kerberos, 60, 566–571
key escrow, 97–100
key exchange, 47–52
KryptoKnight, 571–572
lessons, 64–65
mental poker, 92–95
multiparty unconditionally secure, 137
Needham-Schroeder, 58
Neuman-Stubblebine, 60–62
oblivious signatures, 117
oblivious transfer, 116–117
one-way accumulators, 95–96
Otway-Rees, 59–60
purpose, 22–23
secret splitting, 70–71
secure circuit evaluation, 137
secure elections, *see* Secure elections
secure multiparty computation, 134–137
self-enforcing, 26–27
SESAME, 572
simultaneous contract signing, 118–122
simultaneous exchange of secrets, 123–124
subliminal channel, 79–80
timestamping, 75–79
types, 24
Wide-Mouth Frog, 56–57
Woo-Lam, 63–64
Yahalom, 57–58
See also Authentication; Zero-knowledge proofs
Pseudo-Hadamard Transform, 340
Pseudo-random function family, SEAL, 398–399
Pseudo-random-number generator, 78, 416
Pseudo-random sequence, 44–45
Pseudo-random-sequence generator, 44
bit commitment using, 88
generating multiple streams, 420–421
linear congruential generators, 369–372
linear feedback shift registers, 372–379
PSPACE, 241
Public key, 5
certificates, 185–187
creating from private key, 464
key length, 158–165
recommended lengths, 161–163
key management, 185–187
Public-key algorithms, 4–5, 33, 500–502
background, 461–462
based on linear error-correcting codes, 480
Diffie-Hellman, 513
ElGamal, 476–479
elliptic curve cryptosystems, 480–481
finite automaton cryptosystems, 482
knapsack algorithms, 462–466
LUC, 481
McEliece, 479–480
one-way hash functions using, 455
Pohlig-Hellman, 474
Rabin, 475–476
RSA, *see* RSA
security, 461–462
strength, 502
Public-key cryptography:
attacks against, 43–44
authentication using, 53–54
coin flipping using, 90–91
communications using, 31–34
identity-based, 115
key exchange with, 48
multiple-key, 68–69
private keys, lifetime, 184
signing documents with, 37–38
one-way hash functions, 38–39
versus symmetric cryptography, 216–217
Public-Key Cryptography Standards, 588–589
Public Key Partners, 604–605
Public-key ring, 585
Purchase-key attack, 7

Quadratic nonresidues, 251
Quadratic residues, 250–251
generator, 417
Quadratic sieve, 256
factoring, 159
Quantum computing, 164–165
Quantum cryptography, 554–557
Quintuple encryption, 366
Quisquater, Jean-Jacques, 102, 508
Quisquater-Girault, 450

Rabin, 475–476
Rabin, Michael, 103, 259, 518, 550
Rabin-Miller algorithm, 259–260
RACE Integrity Primitives Evaluation, 605–606
Rackoff, Charles, 352

Rainbow Books, 600
Rambutan, 390
Random keys, 173–174
Random noise, as random-sequence generator, 423–424
Random-number generation, 44
Random-sequence generators, 421–428
  biases and correlations, 425–426
  computer clock, 424
  distilling randomness, 426–428
  keyboard latency measurement, 424–425
  RAND tables, 422–423
  using random noise, 423–424
Random sequences, real, 45–46
Randomized approach, stream ciphers, 415
Randomized stream cipher, 419
Randomness, distilling, 426–428
RAND tables, 422–423
Rao-Nam algorithm, 346
Rate of the language, 234
RC2, 318–319
RC4, 319, 397–398
RC5, 344–346
  source code, 659–662
RDES, 297–298
Receipt, resending message as, 42–43
REDOC II, 311–313
REDOC III, 313
Redundancy, of language, 234
Reeds, Jim, 369
Related-key cryptanalysis, 290
Renji, Tao, 482
Renting Passports, 111
Replay attacks, 58–59
Research and Development in Advanced Communication Technologies, Integrity Primitives Evaluation, 605–606
Resend attack, foiling, 43
Residue, 242
  quadratic, 250–251
  reduced set, 248
Restricted algorithms, 3
RFC 1421, 578
RFC 1422, 578
RFC 1423, 578
RFC 1424, 578
Richter, Manfield, 423
Riordan, Mark, 583–584
RIPE, 605–606
RIPEM, 583–584
RIPE-MAC, 457–458
RIPE-MD, 445
Rip van Winkle cipher, 418–419
Rivest, Ron, 159, 163, 318–319, 344, 397, 435, 440–441, 444, 446, 467
Rivest Cipher, 318
Robshaw, Matt, 342
Rogaway, Phil, 398
ROM key, 181
ROT13, 11
Rotor machines, 12–13
RSA, 17, 466–474
  ability to break, zero-knowledge proofs, 548–549
  attack on encrypting and signing with, 473–474
  blind signatures, 548
  chosen ciphertext attack, 471–472
  common modulus attack, 472
  compared to DSA, 485
  computation time comparison with DSA, 489
  as *de facto* standard, 485–486
  EKE implementation, 519
  encryption, 468
    with DSA, 491
  in hardware, 469
  low decryption exponent attack, 473
  low encryption exponent attack, 472–473
  patents, 474
  restrictions on use, 473
  security, 470–471
  speed, 469
  standards, 474
RSA Data Security, Inc., 295, 603–604
RSA Factoring Challenge, 257
RSA generator, 417
Rubber-hose cryptanalysis, 7
Rueppel, Ranier, 385–386
Running-key cipher, 12

SAFER K-64, 339–341
SAFER K-128, 341
Salt, 52–53
S-boxes:
  alternate, DES, 296–298
  Blowfish, 336
  Boolean functions in, 350
  DES, key-dependent, 298, 300
  design
    criteria, 294
    security questions, 284
    theory, 349–351
  Lucifer, 303
  NSA role, 278, 280
  substitution, 274–276
Scherbius, Arthur, 13
Schlafly, Roger, 394
Schneier, Bruce, 336, 346
Schnorr, Claus, 418, 446, 510
Schnorr authentication and signature scheme, 510–512
Schroeder, Michael, 58, 216
Schwartau, Winn, 300
Sci.crypt, 608–609
Scott, Robert, 306
SEAL, 398–400
  source code, 667–673
Secrecy:
  ideal, 236
  perfect, 235
Secrets, simultaneous exchange, 123–124
Secret sharing, 71–73
  without adjudication, 72
  with cheaters, 72
  with disenrollment, 73
  without revealing shares, 73
  schemes with prevention, 73
  verifiable, 73
Secret-sharing algorithms, 528–531
  advanced threshold schemes, 530–531
  Asmuth-Bloom, 529–530
  cheater detection, 531
  Karnin-Greene-Hellman, 530
  LaGrange interpolating polynomial scheme, 528–529
  vector scheme, 529
Secret splitting, 70–71
  digital cash, 142–145
Secure and Fast Encryption Routine, 339
Secure circuit evaluation, 137
Secure elections, 125–134
  divided protocols, 133
  multiple-key ciphers, 133
  simplistic voting protocols, 125–126
  voting with
    blind signatures, 126–127
    single central facility, 128–130
    two central facilities, 127–128

Secure elections (*Cont.*)
voting without central tabulating facility, 130–133
Secure European System for Applications in a Multivendor Environment, 572
Secure Hash Algorithm, 442–445
Secure multiparty computation, 134–137, 551–552
Secure Telephone Unit, 565
Security:
of algorithms, 8–9
Blowfish, 339
cipher block chaining mode, 196–197
ciphers based on one-way hash functions, 353–354
cryptosystem, 234–235
DES, 278, 280–285
algebraic structure, 282–283
current, 300–301
key length, 283–284
weak keys, 280–281
DSA, 491–492
ESIGN, 500
Kerberos, 571
knapsack algorithms, 465
MD5, 440–441
MMB, 326–327
output-feedback mode, 205
PKZIP, 395
Privacy-Enhanced Mail, 582–583
requirements for different information, 167
RSA, 470–471
SEAL, 400
Secure Hash Algorithm, 444–445
self-synchronizing stream cipher, 199
Selector string, 143
Self-decimated generator, 385–387
Self-enforcing protocols, 26–27
Self-recovering, cipher block chaining mode, 196
Self-shrinking generator, 388
Self-synchronizing stream cipher, 198–199
Selmer, E. S., 381
Semiweak keys, DES, 280–281
SESAME, 572
Session keys, 33, 180
SHA, 442–445
Shadows, 71–72
Shamir, Adi, 72, 284–285, 288, 291, 296, 303, 311–312, 314, 319, 416, 434, 462, 467, 502–504, 508, 516, 528
Shamir's pseudo-random-number generator, 416
Shamir's three-pass protocol, 516–517
Shimizu, Akihiro, 308
Shor, Peter, 164
Shrinking generator, 388, 411–412
Signature equation, 496
Signatures, *see* Digital signatures
Silverman, Bob, 159
Simmons, Gustavus, 72, 79, 493, 501, 531
Simple columnar transposition cipher, 12
Simple relations, 347–348
Simple substitution cipher, 10–11
Simultaneous exchange of secrets, 123–124
Skew, 425
SKEY, 53
SKID, 55–56
Skipjack, 267, 328–329
Smart cards, 587
observer, 146
Universal Electronic Payment System, 589–591
Smith, Lynn, 266
s^nDES, 298–299
Snefru, 432
Software:
DES implementation, 278–279
encryption, 225
linear feedback shift registers, 378–379
RSA speedups, 469–470
Software-based brute-force attack, 154–155
Software Publishers Association, 608
Solovay, Robert, 259
Solovay-Strassen algorithm, 259
Space complexity, 237
Sparse, 378
Special number field sieve, 160–161
SP network, 347
Square roots:
coin flipping using, 541–542
modulo n, 258
Standards:
public-key cryptography, 588–589
RSA, 474
Station-to-station protocol, 516
Steganography, 9–10
StepRightUp, 414
Stereotyped beginnings, 190
Stereotyped endings, 190
Storage:
data encryption for, 220–222
keys, 180–181
requirements, 9
Stornetta, W. Scott, 75
Straight permutation, 275
Strassen, Volker, 259
Stream algorithms, 4
Stream ciphers, 4, 189, 197–198
A5, 389
additive generators, 390–392
Algorithm M, 393–394
versus block ciphers, 210–211
Blum, Blum, and Shub generator, 417–418
Blum-Micali generator, 416–417
cascading multiple, 419–420
cellular automaton generator, 414
choosing, 420
complexity-theoretic approach, 415–418
correlation immunity, 380
counter mode, 206
crypt(1), 414
design and analysis, 379–381
Diffie's randomized stream cipher, 419
encryption speeds, 420
feedback with carry shift registers, 402–404
Fish, 391
Gifford, 392–393
Hughes XPD/KPD, 389–390
information-theoretic approach, 418
linear complexity, 380
Maurer's randomized stream cipher, 419
message authentication codes, 459
multiple, generating from single pseudo-random-sequence generator, 420–421

Mush, 392
Nanoteq, 390
nonlinear-feedback shift registers, 412–413
1/p generator, 414
output-feedback mode, 205
Pike, 391–392
PKZIP, 394–395
Pless generator, 413–414
Rambutan, 390
random-sequence generators, 421–428
RC4, 397–398
Rip van Winkle cipher, 418–419
RSA generator, 417
SEAL, 398–400
self-synchronizing, 198–199
synchronous, 202–203
system-theoretic approach, 415–416
using feedback with carry shift registers, 405–412
alternating stop-and-go generators, 410–411
cascade generators, 405
FCSR combining generators, 405, 410
LFSR/FCSR summation/parity cascade, 410–411
shrinking generators, 411–412
using linear feedback shift registers, 381–388
alternating stop-and-go generator, 383, 385
Beth-Piper stop-and-go generator, 383–384
bilateral stop-and-go generator, 384–385
DNRSG, 387
Geffe generator, 382
generalized Geffe generator, 382–383
Gollmann cascade, 387–388
Jennings generator, 383–384
multispeed inner-product generator, 386–387
self-decimated generator, 385–387
self-shrinking generator, 388
shrinking generator, 388
summation generator, 386–387
threshold generator, 384–386
WAKE, 400–402
Strict avalanche criteria, 350
Strong primes, 261
STU-III, 565–566
Subkey, 272
Blowfish, 338–339
Crab, 342–343
IDEA, 322
independent, DES, 295
Subliminal channel, 79–80
applications, 80
DSA, 493, 534–536
ElGamal, 532–533
ESIGN, 533–534
foiling, 536
Ong-Schnorr-Shamir, 531–532
signature algorithm, 79
Subliminal-free signature schemes, 80
Subprotocols, 26
Substitution boxes, 274–276
Substitution ciphers, 10–12
Substitution-permutation network, 347
SubStream, 414
Summation generator, 386–387
Superincreasing knapsack, 463–464
Superincreasing sequence, 463–464
Suppress-replay, 61
Surety Technologies, 79
SXAL8, 344
Symmetric algorithms, 4
Symmetric block algorithms, one-way hash functions using, 446–455
Symmetric cryptography:
bit commitment using, 86–87
communication using, 28–29
key exchange with, 47–48
versus public-key cryptography, 216–217
Symmetric cryptosystems, document signing, 35–37
Symmetric key length, 151–158
Synchronous stream cipher, 202–203
System-theoretic approach, stream ciphers, 415–416

Tap sequence, 373
feedback with carry shift registers, maximal-length, 408–409
Tatebayashi-Matsuzaki-Newman, 524–525
Tavares, Stafford, 334
TEA, 346
TEMPEST, 224
Terminology, 1–9, 39
Terrorist Fraud, 110
Thermodynamics, limitations on brute-force attacks, 157–158
Three-pass protocol, Shamir's, 516–517
Three-Satisfiability, 242
3-Way, 341–342, 354
source code, 654–659
Three-Way Marriage Problem, 242
Threshold generator, 384–386
Threshold schemes, 71–72, 530–531
Ticket-Granting Service, 567
Ticket Granting Ticket, 569
Tickets, 568
Time complexity, 237
Timestamping, 75
arbitrated solution, 75–76
digital signatures, 38
distributed protocol, 77–78
improved arbitrated solution, 76
improvements, 78–79
linking protocol, 76–77
patented protocols, 78–79
protocols, 75–79
TIS/PEM, 583
Total break, 8
Traffic analysis, 219
Traffic-flow security, 217
Transfer, oblivious, 116–117
Transposition, 237
ciphers, 12
Trapdoor one-way function, 30
Traveling Salesman Problem, 241–242
Trees, digital signatures, 37
Trial division, 256
Triple encryption, 358–363
encrypt-decrypt-encrypt mode, 359
with minimum key, 360
modes, 360–362
with three keys, 360
with two keys, 358–359
variants, 362–363
TSD, 594–595
Tsujii-Kurosawa-Itoh-Fujioka-Matsumoto, 501
Tuchman, Walt, 266, 278, 280, 294, 303, 358
Tuckerman, Bryant, 266
Turing, Alan, 240

Turing machine, 239, 241
2-adic numbers, 404

UEPS, 589–591
Uncertainty, 234
Unconditional sender and recipient untraceability, 138
Undeniable digital signatures, 81–82, 536–539
Unicity distance, 235–236
Unit key, 591
United States, export rules, 610–616
Universal Electronic Payment System, 589–591
Unpredictable, to left and to right, 417
Updating, keys, 180
Utah Digital Signature Act, 618

van Oorschot, Paul, 359
Vector scheme, 529
Verification, keys, 178–179
Verification block, 179
Verification equation, 496
Vernam, Gilbert, 15
Vigenere cipher, 10–11, 14
Vino, 346
Viruses, to spread cracking program, 155–156
VLSI 6868, 278
Voting, *see* Secure elections

WAKE, 400–402
Wayner, Peter, 10
Weak keys:
  block ciphers design theory, 348
  DES, 280–281
Wheeler, David, 400
Whitening, 363, 366–367
Wide-Mouth Frog protocol, 56–57
Wiener, Michael, 153, 284, 359
Williams, 475–476
Wolfram, Steve, 414, 446
Wood, Michael, 311, 313
Woo-Lam protocol, 63–64
Word Auto Key Encryption, 400
Work factor, 9

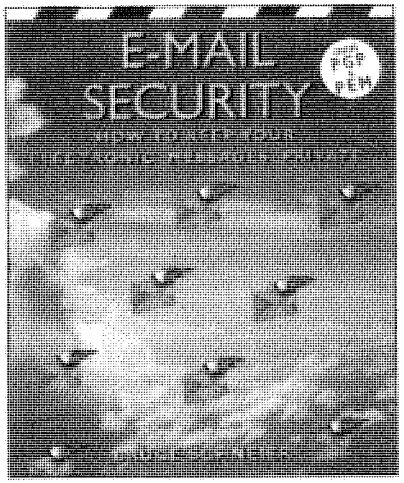
xDES[1], 365–366
XOR, 13–15
XPD, 389–390

Yagisawa algorithm, 501
Yahalom, 57–58
Yao's millionaire problem, 551
Yung, Moti, 81
Yuval, Gideon, 430

Zero-knowledge proofs, 101–109, 548–549
  ability to break RSA, 548–549
  Chess Grandmaster Problem, 109
  computational, 108
  discrete logarithm, 548
  generalities, 108–109
  identity, 109–111
  Mafia Fraud, 110
  minimum-disclosure, 108
  Multiple Identity Fraud, 111
  n is Blum integer, 549
  noninteractive, 106–107
  no-use, 108
  parallel, 106
  perfect, 108
  Proofs of Membership, 111
  Renting Passports, 111
  statistical, 108
  Terrorist Fraud, 110
Zero-knowledge protocol:
  basic, 102–104
  graph isomorphism, 104–105
  Hamiltonian cycles, 105–106
Zierler, Neal, 381
Zimmermann, Philip, 584

Information Security

BOOKS FROM JOHN WILEY AND SONS



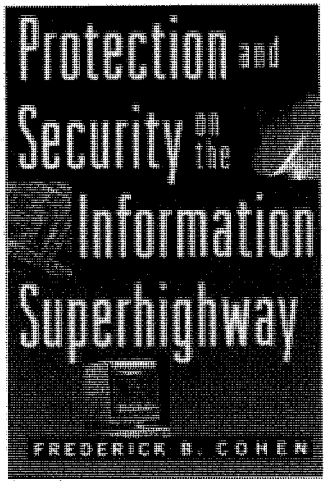
E-Mail Security

How to Keep Your Electronic Messages Private

BY BRUCE SCHNEIER

E-Mail Security is about protecting electronic mail from spies, interlopers, and spoofs—people who may want to destroy, alter, or just look at your private communications. This book shows how you can protect the financial information, contract negotiations or personal correspondence you entrust to public or private networks. Security expert Bruce Schneier shows how this protection is available right now, with free or inexpensive software. The book includes detailed information on PGP and PEM.

ISBN# 0-471-05318-X
Price $24.95 US/$32.50 CAN
Paper 384 pp. 1994

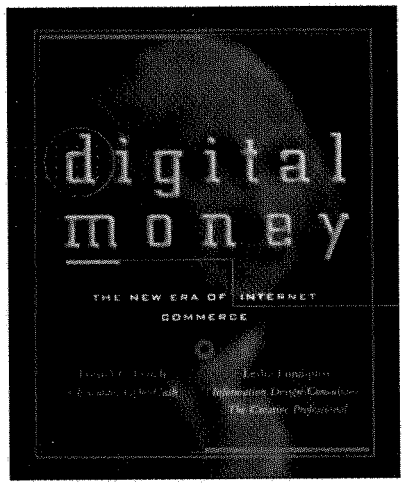


Protection and Security on the Information Superhighway

BY FREDERICK B. COHEN

The FBI estimates that each year as much as $5 billion is lost to computer crime. Just how serious is the problem of information security and how can it affect your life? How vulnerable is your organization's information system? Now get the answers to these and other critical questions in the most penetrating and broad-ranging investigation ever written on the problems of protection and security on the information superhighway. This book reveals the full magnitude of computer security, the impact of faulty security systems and practical steps you can take to protect your organization.

ISBN# 0-471-11389-1
Price $24.95 US/$32.50 CAN
Paper 304 pp. 1994



Digital Money

The New Era of Internet Commerce

DANIEL LYNCH AND LESLIE LUNDQUIST

Until now, commerce on the Internet has been shackled by the lack of secure transactions. *Digital Money* offers an executive briefing on this vital topic. Exploring the technical underpinnings of what can and will be done on the Net, it explains the processes, issues, and strategic considerations of a variety of approaches to secure transactions, including digital signatures. You'll learn about the pros and cons of each approach so you can decide which is best for your business.

ISBN# 0-471-14178-X
$24.95 US/$29.50 CAN
Paper 256 pp. 1995

WILEY
Publishers Since 1807

Available at Bookstores Everywhere • Prices subject to change
For more information **e-mail** - compbks@jwiley.com
or **Visit the Wiley Computer Book Web page at** http://www.wiley.com/CompBooks/CompBooks.html

The *Applied Cryptography* Source Code Disk Set

A source code disk set (three disks) associated with this book is available directly from the author. Included on these disks you will find:

Symmetric Algorithms:

Vigenère Cipher
Playfair Cipher
Hill Cipher
CRYPT (1)
CRYPT (3)
Enigma
DES - 10 versions
Lucifer - 2 versions
NewDES
FEAL-N
FEAL-NX
REDOC II
REDOC III
LOKI89
LOKI91
Khufu
IDEA - 3 versions
CA 1.1
MDC
GOST
BLOWFISH
3-Way
SAFER K-64
SAFER K-128
NewDE
NSEA
RC4
PKZIP
SEAL
WAKE

Public-Key Algorithms:

RSA
Diffie-Hellman
DSA

One-Way Hash Functions:

Snefru
N-Hash
MD4 - 3 versions
MD5 - 2 versions
MD2
SHA
HAVAL
RIPE-MD

Complete Systems:

RIPEM
PGP
TIS-PEM
RSAREF

Other:

LaGrange Threshold Scheme
Mimic Functions
Probabilistic Prime Number Generation
Random Number Generation using Oscillators
Random Number Generation using Keyboard Latency
Frequency Analysis
WordPerfect Password Cracker

Text:

Defense Trade Regulations
DoD Orange Book
European Computer Security Green Book
Various NIST FIPS
Various Internet RFCs

And more!

The disks also include a file containing corrections for all mistakes found in the book, as well as any updated information on any of the topics covered in the text: new algorithms, new protocols, new cryptanalytic results, and so on.

The MS-DOS disks are available from the author, and will be updated twice a year. Cost is $40 for a set, and $120 for a two-year subscription. Please send check or money order in U.S. funds, drawn on a U.S. bank, to:

Bruce Schneier
Counterpane Systems
7115 W. North Ave., Suite 16
Oak Park, IL 60302-1002

Please allow four weeks for delivery, and include your e-mail address if you have one. Due to the export restrictions on many of the algorithms on these disks, they will only be mailed to addresses within the United States and Canada. Apologies to the foreign readers of this book.

Programming/Security

$49.95 USA
$64.95 CAN

"...the best introduction to cryptography I've ever seen.... The book the National Security Agency wanted never to be published...."

—*Wired* Magazine

"... monumental ... fascinating ... comprehensive ... the definitive work on cryptography for computer programmers..."

—*Dr. Dobb's Journal*

"...easily ranks as one of the most authoritative in its field."

—*PC Magazine*

"...the bible of code hackers."

—*The Millennium Whole Earth Catalog*

This new edition of the cryptography classic provides you with a comprehensive survey of modern cryptography. The book details how programmers and electronic communications professionals can use cryptography—the technique of enciphering and deciphering messages—to maintain the privacy of computer data. It describes dozens of cryptography algorithms, gives practical advice on how to implement them into cryptographic software, and shows how they can be used to solve security problems. Covering the latest developments in practical cryptographic techniques, this new edition shows programmers who design computer applications, networks, and storage systems how they can build security into their software and systems.

What's new in the *Second Edition*?

- New information on the Clipper Chip, including ways to defeat the key escrow mechanism
- New encryption algorithms, including algorithms from the former Soviet Union and South Africa, and the RC4 stream cipher
- The latest protocols for digital signatures, authentication, secure elections, digital cash, and more
- More detailed information on key management and cryptographic implementations

BRUCE SCHNEIER is President of Counterpane Systems, a consulting firm specializing in cryptography and computer security. He is a contributing editor to *Dr. Dobb's Journal*, serves on the board of directors of the International Association of Cryptologic Research, and is a member of the Advisory Board for the Electronic Privacy Information Center. He is the author of *E-Mail Security* (Wiley) and is a frequent lecturer on cryptography, computer security, and privacy.



Cover Painting: Mona Mark

John Wiley & Sons, Inc.
Professional, Reference and Trade Group
605 Third Avenue, New York, N.Y. 10158-0012
New York • Chichester • Brisbane • Toronto • Singapore

ISBN 0-471-11709-9
54995
9 780471 117094