Schneier on Security

Security Pitfalls in Cryptography

Bruce Schneier
Information Management & Computer Security
1998

French translation

Magazine articles like to describe cryptography products in terms of algorithms and key length. Algorithms make good sound bites: they can be explained in a few words and they're easy to compare with one another. "128-bit keys mean good security." "Triple-DES means good security." "40-bit keys mean weak security." "2048-bit RSA is better than 1024-bit RSA."

But reality isn't that simple. Longer keys don't always mean more security. Compare the cryptographic algorithm to the lock on your front door. Most door locks have four metal pins, each of which can be in one of ten positions. A key sets the pins in a particular configuration. If the key aligns them all correctly, then the lock opens. So there are only 10,000 possible keys, and a burglar willing to try all 10,000 is guaranteed to break into your house. But an improved lock with ten pins, making 10 billion possible keys, probably won't make your house more secure. Burglars don't try every possible key (a brute-force attack); most aren't even clever enough to pick the lock (a cryptographic attack against the algorithm). They smash windows, kick in doors, disguise themselves as policemen, or rob keyholders at gunpoint. One ring of art thieves in California defeated home security systems by taking a chainsaw to the house walls. Better locks don't help against these attacks.

Strong cryptography is very powerful when it is done right, but it is not a panacea. Focusing on the cryptographic algorithms while ignoring other aspects of security is like defending your house not by building a fence around it, but by putting an immense stake into the ground and hoping that the adversary runs right into it. Smart attackers will just go around the algorithms.

Counterpane has spent years designing, analyzing, and breaking cryptographic systems. While we do research on published algorithms and protocols, most of our work examines actual products. We've designed and analyzed systems that protect privacy, ensure confidentiality, provide fairness, and facilitate commerce. We've worked with software, stand-alone hardware, and everything in between. We've broken our share of algorithms, but we can almost always find attacks that bypass the algorithms altogether. We don't have to try every possible key, or even find flaws in the algorithms. We exploit errors in design, errors in implementation, and errors in installation. Sometimes we invent a new trick to break a system, but most of the time we exploit the same old mistakes that designers make over and over again.

Attacks Against Cryptographic Designs

Security First Innovations, LLC, Exhibit 2008
Page 2008 - 1
IPR2024-00215, Google LLC v. Security First Innovations, LLC

A cryptographic system can only be as strong as the encryption algorithms, digital signature algorithms, one-way hash functions, and message authentication codes it relies on. Break any of them, and you've broken the system. And just as it's possible to build a weak structure using strong materials, it's possible to build a weak cryptographic system using strong algorithms and protocols.

We often find systems that "void the warranty" of their cryptography by not using it properly: failing to check the size of values, reusing random parameters that should never be reused, and so on. Encryption algorithms don't necessarily provide data integrity. Key exchange protocols don't necessarily ensure that both parties receive the same key. In a recent research project, we found that some—not all—systems using related cryptographic keys could be broken, even though each individual key was secure. Security is a lot more than plugging in an algorithm and expecting the system to work. Even good engineers, well-known companies, and lots of effort are no guarantee of robust implementation; our work on the U.S. digital cellular encryption algorithm illustrated that.

Random-number generators are another place where cryptographic systems often break. Good random-number generators are hard to design, because their security often depends on the particulars of the hardware and software. Many products we examine use bad ones. The cryptography may be strong, but if the random-number generator produces weak keys, the system is much easier to break. Other products use secure random-number generators, but they don't use enough randomness to make the cryptography secure.

Recently Counterpane published new classes of attacks against random-number generators, based on our work with commercial designs. One of the most surprising things we've found is that specific random-number generators may be secure for one purpose but insecure for another; generalizing security analyses is dangerous.

In another research result, we looked at interactions between individually secure cryptographic protocols. Given a secure protocol, we show how to build another secure protocol that will break the first if both are used with the same keys on the same device.

Attacks Against Implementations

Many systems fail because of mistakes in implementation. Some systems don't ensure that plaintext is destroyed after it's encrypted. Other systems use temporary files to protect against data loss during a system crash, or virtual memory to increase the available memory; these features can accidentally leave plaintext lying around on the hard drive. In extreme cases, the operating system can leave the keys on the hard drive. One product we've seen used a special window for password input. The password remained in the window's memory even after it was closed. It didn't matter how good that product's cryptography was; it was broken by the user interface.

Other systems fall to more subtle problems. Sometimes the same data is encrypted with two different keys, one strong and one weak. Other systems use master keys and then one-time session keys. We've broken systems using partial information about the different keys. We've also seen systems that use inadequate protection mechanisms for the master keys, mistakenly relying on the security of the session keys. It's vital to secure all possible ways to learn a key, not just the most obvious ones.

Security First Innovations, LLC, Exhibit 2008
Page 2008 - 2
IPR2024-00215, Google LLC v. Security First Innovations, LLC

Electronic commerce systems often make implementation trade-offs to enhance usability. We've found subtle vulnerabilities here, when designers don't think through the security implications of their trade-offs. Doing account reconciliation only once per day might be easier, but what kind of damage can an attacker do in a few hours? Can audit mechanisms be flooded to hide the identity of an attacker? Some systems record compromised keys on "hotlists"; attacks against these hotlists can be very fruitful. Other systems can be broken through replay attacks: reusing old messages, or parts of old messages, to fool various parties.

Systems that allow old keys to be recovered in an emergency provide another area to attack. Good cryptographic systems are designed so that the keys exist for as short a period of time as possible; key recovery often negates any security benefit by forcing keys to exist long after they are useful. Furthermore, key recovery databases become sources of vulnerability in themselves, and have to be designed and implemented securely. In one product we evaluated, flaws in the key recovery database allowed criminals to commit fraud and then frame legitimate users.

Attacks Against Passwords

Many systems break because they rely on user-generated passwords. Left to themselves, people don't choose strong passwords. If they're forced to use strong passwords, they can't remember them. If the password becomes a key, it's usually much easier—and faster—to guess the password than it is to brute-force the key; we've seen elaborate security systems fail in this way. Some user interfaces make the problem even worse: limiting the passwords to eight characters, converting everything to lower case, etc. Even passphrases can be weak: searching through 40-character phrases is often much easier than searching through 64-bit random keys. We've also seen key-recovery systems that circumvent strong session keys by using weak passwords for key-recovery.

Attacks Against Hardware

Some systems, particularly commerce systems, rely on tamper-resistant hardware for security: smart cards, electronic wallets, dongles, etc. These systems may assume public terminals never fall into the wrong hands, or that those "wrong hands" lack the expertise and equipment to attack the hardware. While hardware security is an important component in many secure systems, we distrust systems whose security rests solely on assumptions about tamper resistance. We've rarely seen tamper resistance techniques that work, and tools for defeating tamper resistance are getting better all the time. When we design systems that use tamper resistance, we always build in complementary security mechanisms just in case the tamper resistance fails.

The "timing attack" made a big press splash in 1995: RSA private keys could be recovered by measuring the relative times cryptographic operations took. The attack has been successfully implemented against smart cards and other security tokens, and against electronic commerce servers across the Internet. Counterpane and others have generalized these methods to include attacks on a system by measuring power consumption, radiation emissions, and other "side channels," and have implemented them against a variety of public-key and symmetric algorithms in "secure" tokens. We've yet to find a token that we can't pull the secret keys out of by looking at side channels. Related research has looked at fault analysis: deliberately introducing faults into cryptographic processors in order to determine the secret keys. The effects of this attack can be devastating.

Security First Innovations, LLC, Exhibit 2008
Page 2008 - 3
IPR2024-00215, Google LLC v. Security First Innovations, LLC

Attacks Against Trust Models

Many of our more interesting attacks are against the underlying trust model of the system: who or what in the system is trusted, in what way, and to what extent. Simple systems, like hard-drive encryption programs or telephone privacy products, have simple trust models. Complex systems, like electronic-commerce systems or multi-user e-mail security programs, have complex (and subtle) trust models. An e-mail program might use uncrackable cryptography for the messages, but unless the keys are certified by a trusted source (and unless that certification can be verified), the system is still vulnerable. Some commerce systems can be broken by a merchant and a customer colluding, or by two different customers colluding. Other systems make implicit assumptions about security infrastructures, but don't bother to check that those assumptions are actually true. If the trust model isn't documented, then an engineer can unknowingly change it in product development, and compromise security.

Many software systems make poor trust assumptions about the computers they run on; they assume the desktop is secure. These programs can often be broken by Trojan horse software that sniffs passwords, reads plaintext, or otherwise circumvents security measures. Systems working across computer networks have to worry about security flaws resulting from the network protocols. Computers that are attached to the Internet can also be vulnerable. Again, the cryptography may be irrelevant if it can be circumvented through network insecurity. And no software is secure against reverse-engineering.

Often, a system will be designed with one trust model in mind, and implemented with another. Decisions made in the design process might be completely ignored when it comes time to sell it to customers. A system that is secure when the operators are trusted and the computers are completely under the control of the company using the system may not be secure when the operators are temps hired at just over minimum wage and the computers are untrusted. Good trust models work even if some of the trust assumptions turn out to be wrong.

Attacks on the Users

Even when a system is secure if used properly, its users can subvert its security by accident—especially if the system isn't designed very well. The classic example of this is the user who gives his password to his co-workers so they can fix some problem when he's out of the office. Users may not report missing smart cards for a few days, in case they are just misplaced. They may not carefully check the name on a digital certificate. They may reuse their secure passwords on other, insecure systems. They may not change their software's default weak security settings. Good system design can't fix all these social problems, but it can help avoid many of them.

Attacks Against Failure Recovery

Strong systems are designed to keep small security breaks from becoming big ones. Recovering the key to one file should not allow the attacker to read every file on the hard drive. A hacker who reverse-engineers a smart card should only learn the secrets in that smart card, not information that will help him break other smart cards in the system. In a multi-user system, knowing one person's secrets shouldn't compromise everyone else's.

Many systems have a "default to insecure mode." If the security feature doesn't work, most people just turn it off and finish their business. If the on-line credit card verification system is down, merchants will

Security First Innovations, LLC, Exhibit 2008
Page 2008 - 4
IPR2024-00215, Google LLC v. Security First Innovations, LLC

default to the less-secure paper system. Similarly, it is sometimes possible to mount a “version rollback attack” against a system after it has been revised to fix a security problem: the need for backwards compatibility allows an attacker to force the protocol into an older, insecure, version.

Other systems have no ability to recover from disaster. If the security breaks, there’s no way to fix it. For electronic commerce systems, which could have millions of users, this can be particularly damaging. Such systems should plan to respond to attacks, and to upgrade security without having to shut the system down. The phrase “and then the company is screwed” is never something you want to put in your business plan. Good system design considers what will happen when an attack occurs, and works out ways to contain the damage and recover from the attack.

Attacks Against the Cryptography

Sometimes, products even get the cryptography wrong. Some rely on proprietary encryption algorithms. Invariably, these are very weak. Counterpane has had considerable success breaking published encryption algorithms; our track record against proprietary ones is even better. Keeping the algorithm secret isn’t much of an impediment to analysis, anyway—it only takes a couple of days to reverse-engineer the cryptographic algorithm from executable code. One system we analyzed, the S/MIME 2 electronic-mail standard, took a relatively strong design and implemented it with a weak cryptographic algorithm. The system for DVD encryption took a weak algorithm and made it weaker.

We’ve seen many other cryptographic mistakes: implementations that repeat “unique” random values, digital signature algorithms that don’t properly verify parameters, hash functions altered to defeat the very properties they’re being used for. We’ve seen cryptographic protocols used in ways that were not intended by the protocols’ designers, and protocols “optimized” in seemingly trivial ways that completely break their security.

Attack Prevention vs. Attack Detection

Most cryptographic systems rely on prevention as their sole means of defense: the cryptography keeps people from cheating, lying, abusing, or whatever. Defense should never be that narrow. A strong system also tries to detect abuse and to contain the effects of any attack. One of our fundamental design principles is that sooner or later, every system will be successfully attacked, probably in a completely unexpected way and with unexpected consequences. It is important to be able to detect such an attack, and then to contain the attack to ensure it does minimal damage.

More importantly, once the attack is detected, the system needs to recover: generate and promulgate a new key pair, update the protocol and invalidate the old one, remove an untrusted node from the system, etc. Unfortunately, many systems don’t collect enough data to provide an audit trail, or fail to protect the data against modification. Counterpane has done considerable work in securing audit logs in electronic commerce systems, mostly in response to system designs that could fail completely in the event of a successful attack. These systems have to do more than detect an attack: they must also be able to produce evidence that can convince a judge and jury of guilt.

Building Secure Cryptographic Systems

Security First Innovations, LLC, Exhibit 2008
Page 2008 - 5
IPR2024-00215, Google LLC v. Security First Innovations, LLC

Security designers occupy what Prussian general Carl von Clausewitz calls “the position of the interior.” A good security product must defend against every possible attack, even attacks that haven’t been invented yet. Attackers, on the other hand, only need to find one security flaw in order to defeat the system. And they can cheat. They can collude, conspire, and wait for technology to give them additional tools. They can attack the system in ways the system designer never thought of.

Building a secure cryptographic system is easy to do badly, and very difficult to do well. Unfortunately, most people can’t tell the difference. In other areas of computer science, functionality serves to differentiate the good from the bad: a good compression algorithm will work better than a bad one; a bad compression program will look worse in feature-comparison charts. Cryptography is different. Just because an encryption program works doesn’t mean it is secure. What happens with most products is that someone reads *Applied Cryptography*, chooses an algorithm and protocol, tests it to make sure it works, and thinks he’s done. He’s not. Functionality does not equal quality, and no amount of beta testing will ever reveal a security flaw. Too many products are merely “buzzword compliant”; they use secure cryptography, but they are not secure.

Categories: Computer and Information Security, Trust

Tags: Information Management & Computer Security

← The Challenge of Cryptography

Security for Remote Access VPNs Must Be Simple →

Powered by WordPress Hosted by Pressable

Security First Innovations, LLC, Exhibit 2008
Page 2008 - 6
IPR2024-00215, Google LLC v. Security First Innovations, LLC