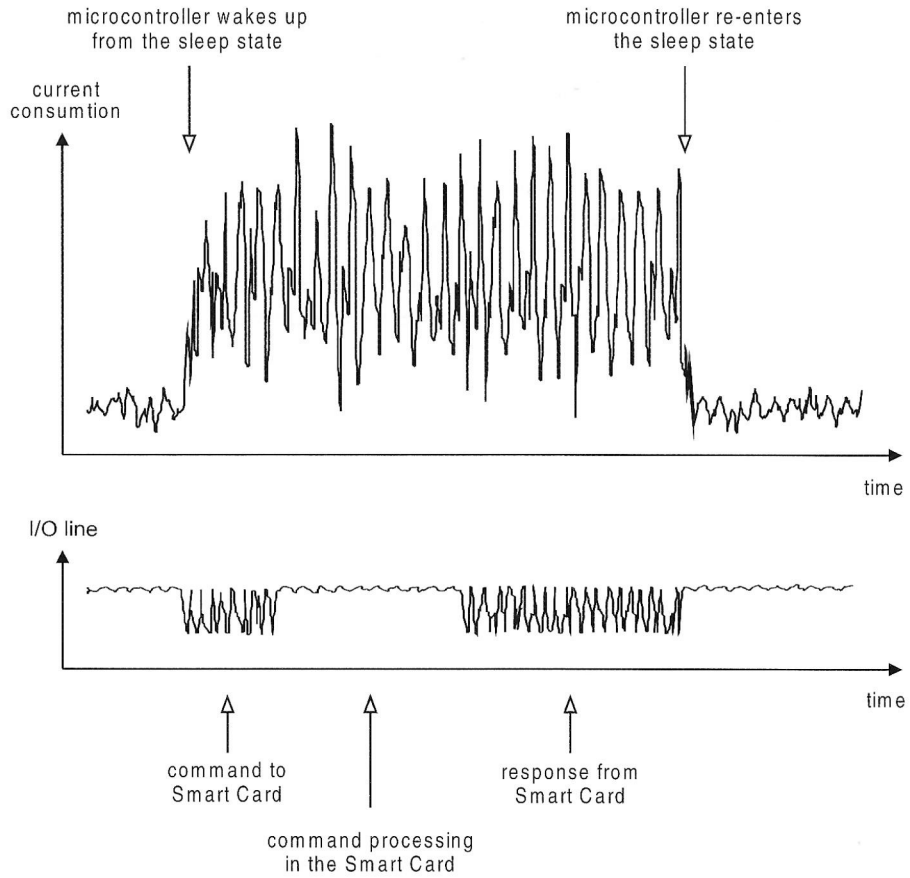


Figure 8.33 Simplified representation of the current consumption of a Smart Card microcontroller in the idle state and the current variations during operation. Based on the current drawn by the microcontroller, it is possible to recognize when the microcontroller awakes from the sleep state following the first falling edge on the I/O line, following which it exhibits a continuously varying power consumption that depends on the machine instruction being executed.

In the paper by Kocher *et al.*, 'high-order differential power analysis' (HO-DPA) is mentioned as a further extension of DPA. This involves measuring not only the current consumption of the microcontroller, but also other quantities that depend on the program being executed by the processor, such as the electromagnetic radiation of the chip. The measurement data collected in this manner, for both known and unknown data, can then be used in the same way as in the DPA technique to calculate differences that can be used to compute the secret data.

These three types of power analysis for Smart Card microcontrollers represent very serious forms of attack on hardware and software that has not been protected using suitable countermeasures. This is because the current consumption of some microcontrollers is definitely dependent on the machine instruction being executed as well as the data being processed by the instruction. In addition, the cost and complexity of the equipment needed for a successful attack using this method is relatively limited. However, there is a series of effective countermeasures, which are based on the one hand on suitably improved hardware and on the other hand on modified software.

Exhibit 1017
Page 082 of 102

The simplest hardware solution is to incorporate a fast-acting voltage regulator in the chip, which utilizes a series sense resistor to ensure that the current consumption of the microcontroller is independent of the machine instructions and data. Artificial current-noise generators on the chip are also an effective solution. A technically more complicated solution consists of modifying the semiconductor design of the processor such that it always has a constant power consumption.

There is presently an immense spectrum of possible software countermeasures. We can briefly describe some representative examples here. The simplest approach is to use exclusively machine instructions that have very similar current consumptions. In this case, machine instructions whose current consumption is significantly different from the average level are not allowed to be used in the assembler code. An additional solution is to have several different procedures for performing the same computations in cryptographic algorithms, and to randomly select one of these procedures each time it is actually needed. This makes it considerably more difficult for the observer to recognize a correlation between known and unknown machine instructions or processed data. In order to make it more difficult to obtain the data that must be acquired before a power analysis can be successfully carried out, all keys should be protected by irreversible retry counters. In addition, it is necessary to block free access to all commands (such as INTERNAL AUTHENTICATE) that can be used to pass any desired data through a cryptographic algorithm in the Smart Card. If commands of this sort absolutely must be used for some reason, the Smart Card must test the authenticity of the terminal before executing them. This restriction on the use of the available commands also makes it more difficult to collect reference data for a subsequent power analysis.

Analysis and defense: measuring the electromagnetic radiation of the CPU
It is at least theoretically possible to draw conclusions regarding the internal processes of the Smart Card microcontroller from measurements of its electromagnetic radiation, in the same manner as with differential power analysis. Magnetic fields with small dimensions and strengths can be measured using SQUIDs (superconducting quantum interference devices). However, this is technically enormously complicated, and the knowledge of the internal structure of the semiconductor device that is indispensable for this method is not generally available. In addition, ICs can be very effectively protected against this sort of attack by stacking several traces on top of each other, so that even if a magnetic field can be measured, it is not possible to determine which of the tracks is actually carrying the associated current.

Manipulation of the Smart Card microcontroller

Manipulation and defense: altering the memory contents of the Smart Card microcontroller
Directly reading the memory contents of a microcontroller is a possible attack scenario whose danger can be seen at first glance. A similar scenario that is almost as strong a form of attack is intentionally altering the data contents of a Smart Card microcontroller memory. This does not mean scattering random errors in the computation process of a cryptographic algorithm, which is used as the basis for differential fault analysis (DFA), but instead selectively changing the values of certain bits or bytes in the ROM or EEPROM.

Non-selective changes in all types of memory can be produced by (for example) exposing the module to X-rays or shining ultraviolet light on the exposed chip. EEPROM cells can be discharged by exposing them to ultraviolet light, which causes their contents to

Exhibit 1017
Page 083 of 102

take on the value of the lowest-energy state. This process is exactly the same as erasing a conventional EPROM with an ultraviolet lamp. However, it cannot reasonably be used for an attack, since the attacker has no control over which EEPROM cells are switched.

The ultraviolet lamp can however be replaced by a collimated beam of light, or light from a laser, and this can be focussed to a fine point. This could certainly be used to alter the contents of individual memory cells. The advantage of using a laser is that it can supply enough power to also modify the contents of ROM cells. A focussed ion beam can also be used in a similar manner to change memory cells.

The changes that are possible can certainly be used for theoretically effective attacks. For example, the random number generator could be manipulated such that it no longer produces random numbers, but instead always supplies the same value. If this were possible, the authentication of the terminal by the Smart Card could be broken by a replay attack using a previously employed value.

Other types of attack could also be definitely undertaken if the contents of specific memory bits could be intentionally modified. For example, all S boxes of the DES algorithm could be intentionally altered to a uniform value of zero or one. This would mean that the DES would no longer be an encryption algorithm, but only a linear transformation [Anderson 96a].

If the exact location of the DES key in the EEPROM is known, and it is also possible to modify individual bits in the EEPROM (using focussed ultraviolet light, for example), it is naturally possible to utilize these conditions to mount an effective attack. The attack consists of setting an arbitrary bit of the key to 0 and then calling a command that employs the DES algorithm with the modified key. If the return code indicates a parity error in the key, the bit that has been modified was originally set to 1, while if no parity error is reported, the bit was already set to 0. The same procedure is then followed for the remaining 55 bits of the key, with the result that the secret key is known [Zieschang 98].

Many other types of attacks along the same lines are possible, such as selectively modifying program processes or altering pointer values. These attacks may look very simple and attractive on paper, but it would be very difficult to carry them out in actual practice. This is because the necessary conditions for a successful attack are not exactly easy to achieve, with the result that this type of attack remains an interesting but theoretical concept.

In order to alter bits selectively, an attacker must have detailed knowledge of the physical addresses of data and program code in the memory, and must also know the scrambling scheme used for the memory in question. In addition, all data and program routines that are important with regard to security are protected using checksums that are recalculated each time before using the data or routine. An attacker would therefore also have to selectively modify the checksum to match the modified data. You should also not overlook the fact that all protective layers covering the memory in question must be neutralized before any manipulation can take place. All of these considerations together reduce the attractiveness of this type of attack to almost nothing, even though it must be admitted that it sounds very attractive in theory.

8.2.4.2 Attacks on the logical level

The main prerequisite for attacks on the security of a Smart Card at the logical level is an understanding of the communications and information flow between the terminal and the Smart Card. It is more important to understand the software processes than to understand the processes that take place at the hardware level. In terms of information technology, the

sample scenarios that are described here are located one level above attacks that primarily exploit the characteristics of the hardware.

Attack and defense: dummy Smart Cards
Probably the simplest imaginable type of attack is to use a Smart Card that has been custom programmed and includes additional logging and analysis functions. Up until a few years ago, this was almost unfeasible, since only a few companies could obtain Smart Cards or the microcontrollers used for making them. Nowadays, though, Smart Cards and configuration programs can be openly purchased from a number of firms. This naturally increases the options that are open to an attacker. Even without this, with some effort it is possible to assemble a working Smart Card using a plastic card, a standard microcontroller in an SMD package and a bit of dexterity. Such a card can at least be made to imitate the electrical interface of a real Smart Card and to behave the same way for data transfers. New possibilities are also offered by the Java technology for Smart Cards, which makes it easy to generate programs and load them into dummy cards.

With such a dummy card, it would be possible to log at least a part of the communications with a terminal and subsequently evaluate this information. After several attempts, it would probably be possible to execute part of the communications in exactly the same way as a genuine Smart Card.

Whether this can be put to advantage is doubtful, since all professionally designed applications have cryptographic protection for important activities. As long as the secret key is not known, the attack will not go any farther than the first authentication. Such an attack could be successful if the secret key is known, or if the complete application runs without any cryptographic protection. If such an application should exist, it is very doubtful that any advantages that could be gained by this type of attack would be significant enough to justify the effort needed to mount the attack.

Analysis: determining the command set of a Smart Card
The instruction classes and commands that are supported by a Smart Card are of course not often published, but it is very simple to determine what they are. This is more interesting with regard to completely determining the command set of a Smart Card than it is for an attack on the security of the Smart Card. It is however conceivable that an attack could be constructed on the basis of this information.

The structure of the procedure used to determine the command repertoire is illustrated in Figure 8.34. The first step is to generate a command APDU and send it to the Smart Card using a freely programmable terminal. The class byte in the APDU is changed for each APDU to cover the range from '00' to 'FF'. As soon as a return code other than 'invalid class' is received, the first valid class byte has been determined. There are usually two or three valid instruction classes, which can then be used to try out all possible instruction bytes in the next round. Command APDUs with various instruction bytes are sent to the Smart Card, and those that yield a return code other than 'unknown instruction' are noted. Given suitable software in the terminal, this method can determine which commands are supported by a particular Smart Card in one to two minutes. To a certain degree, some of the possible parameters of the commands so identified can also be determined in a similar manner.

Exhibit 1017
Page 085 of 102

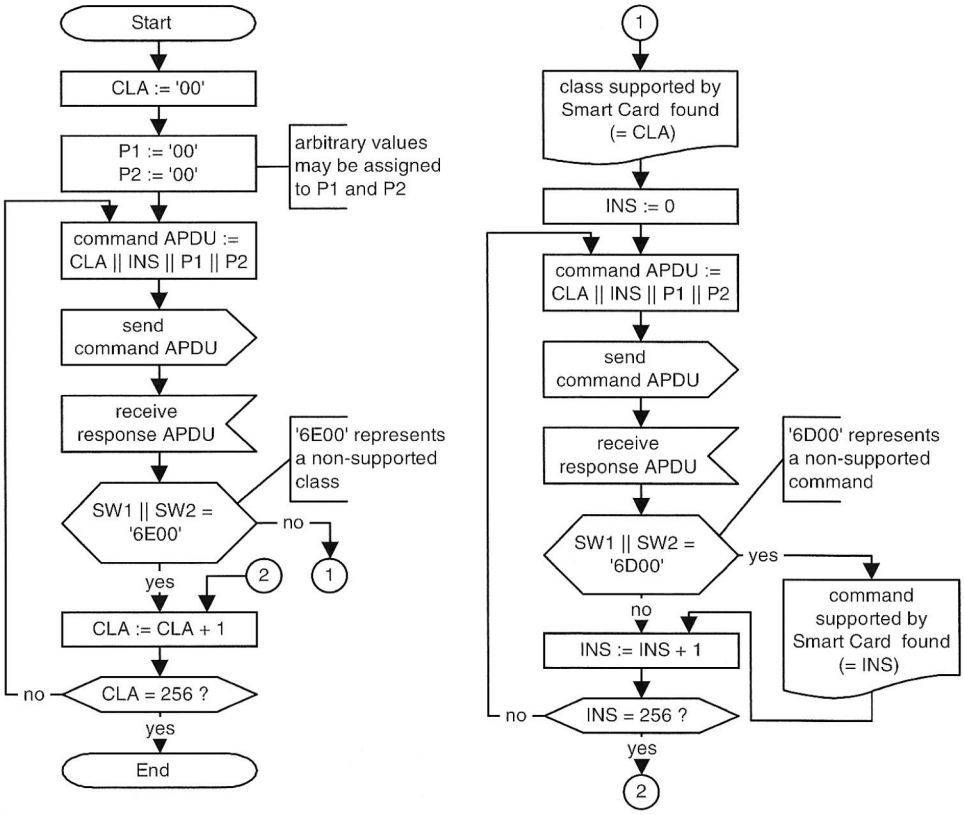


Figure 8.34 Basic procedure of an exhaustive search for all commands supported by a Smart Card operating system. This procedure will identify all supported commands only if command calling is not controlled by a state machine. Its works on the principle of systematically trying out all class byte (CLS) and instruction byte (INS) codes in turn, with no regard to any command contents (secure messaging, logical channels, V_{pp} control and so on).

The illustrated algorithm can be made considerably faster by using only class byte codes that are allowed by the ISO/IEC 7816-4 standard, and allowing the instruction byte to be an indexed variable. This strongly reduces the number space of the class byte by taking secure messaging and logical channels into account. A similar improvement can be made by using only even-valued instruction bytes, since as you know, odd-valued codes only contain the V_{pp} control information, which is no longer necessary.

The reason that this simple search algorithm for instruction classes, commands and parameters can be so effective is that practically all command interpreters in Smart Card operating systems evaluate received commands by starting with the class byte and working through the following bytes. The process is terminated as soon as the first invalid byte is recognized, and a suitable return code is generated and sent back to the terminal.

However, this procedure will only work if the Smart Card does not have a global state machine that monitors the command sequence. If such a state machine is present, it is at least possible to use this procedure to determine the command sequence in a step by step manner.

The advantage that an attacker can derive from this procedure may not appear to be that great, since the command set is normally not secret. However, it does at least provide a simple and fast method to determine all the commands. It can also be very useful in determining whether the producer of the operating system has incorporated any undocumented commands in the software.

Attack: tapping into the data transfer
A slightly modified Smart Card can be used to eavesdrop on the data transfer during a session and to manipulate the data if desired. The modifications consist of gluing an insulated dummy contact on top of the I/O contact surface, so that the original I/O interface is no longer connected to the I/O contact. The new (dummy) contact and the original I/O contact are connected to a fast computer. With suitable programming, this computer can delete or insert any desired data within the communications between the terminal and the Smart Card. If the computer is sufficiently fast, neither the terminal nor the card will detect any difference between normal and manipulated communications.

It is clear that the course of a session can be very strongly influenced using this method. Whether an attacker can derive any benefit from this method depends primarily on the application in the Smart Card. A well-known design principle says that eavesdropping on the communications, deleting data and inserting data must not be allowed to impair security. If this principle is not observed, an attacker can certainly obtain an advantage using this method. There are known cases of fraud using simulated memory cards.

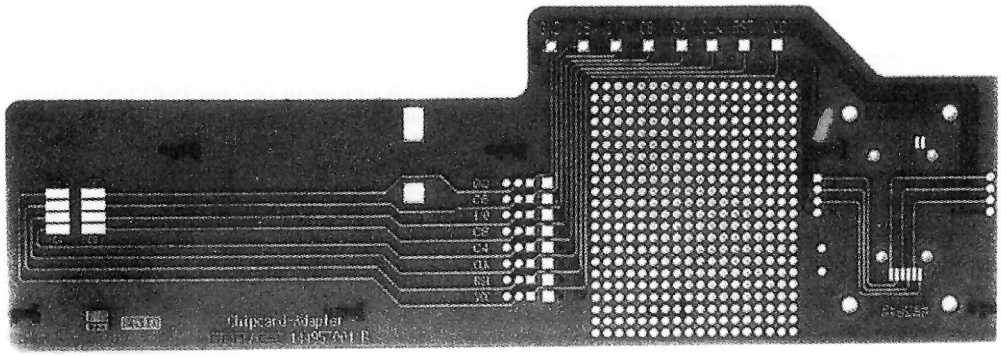


Figure 8.35 An adapter that can be used to extend a Smart Card outside of a terminal enclosure so that measurements can be made on the card. The eight contacts can be seen on the left, while an area for constructing electronic circuits can be seen on the right.

In order to provide protection against this type of attack, some terminals have shutters that cut off any wires attached to the Smart Card. Secure messaging can also be used very effectively here to allow any manipulation of the data during the data transfer to be reliably detected.

Many terminals may be used only under supervision, which makes it difficult to use manipulated cards with leads to an accompanying computer in such terminals. In summary, this type of attack can be regarded as theoretically very interesting and quite promising, but in practice as improbable and above all not too likely to be successful.

Attack and defense: cutting off the power supply

A type of attack that was successful with many Smart Cards until recently is to cut off power to the card at a certain time while a command is being executed. The background for this attack is that, with conventional programming, all write operations on EEPROM pages are carried one after the other. If the programmer has not been clever in arranging the order of the writing operations, cutting off the power at the right time can benefit the attacker.

This can be briefly illustrated using a very simplified example. In an electronic purse application, if the balance is increased before the log file is updated when a purse loading command is executed, at attacker would have a good chance of being to load a Smart Card for free. He would only have to switch off the power at the right time, or jerk the card out of the terminal with millisecond precision (!). The contents of the purse balance would then have been changed to the new value, but there would no log record for this transaction and no response to the command. With simple electronic purse systems in the past, such an attack was certainly a real possibility.

In order to determine the exact time to terminate processing, the attacker only has to use an electronic counter to count the number of clock pulses between the time when the command is sent, and then make a series of experiments with increasing clock counts until the proper time is determined. It hardly needs to be said that the entire procedure can be more or less automated using a computer.

	purse balance	purse balance file (in binary notation)
1. current purse balance	100 DM	°0110 0100°
2. deduct 10 DM	90 DM	
3. erase the EEPROM	255 DM	°1111 1111°
4. write the new purse balance	90 DM	°0101 1010°

Figure 8.36 Procedure for writing a new balance in an electronic purse. In this example, it is assumed that the erased state of the EEPROM represents a 1. This means that, due to the way the EEPROM works, it is necessary to erase the entire EEPROM page (which means setting all of its bits to 1) if only one bit in the page must be changed from 0 to 1. In this example, if the power for the Smart Card is cut off exactly after the EEPROM has been erased, which means after step 3, the purse balance would be set to its maximum value and the attacker would have created money. This can however be reliably prevented by using atomic processes.

Although though this type of attack sounds interesting and easy to copy, there are in practice several effective countermeasures. The simplest approach is to use a well-chosen sequence for the EEPROM write instructions. The EN 1546 standard for multi-sector electronic purses is well worth examining in this regard, since all the electronic purses described in this standard are explicitly protected against this sort of attack.

However, even a perfectly ordered sequence of writing operations cannot by itself achieve absolute protection. This can be made clear using another example. When the electronic purse of our previous example is being loaded, it may be necessary to erase the EEPROM before the writing process. If the erased state of the EEPROM corresponds to the

maximum value of the purse balance, which by the way is usually the case, then it is only necessary to cut off the power at the right time to have the purse loaded to its maximum value. The suitable moment is when the erasing operation has just been completed and the writing operation has not yet been started.

Operating system designers know an effective countermeasure for this type of attack, which is the use of atomic processes as described in detail in Section 5.9. The characteristic of an atomic process is that it is indivisible, which means that it is executed either completely or not at all. This provides fully adequate protection against the type of attack just described. Even the optimally ordered EEPROM writing operations described in the EN 1546 standard require atomic processes in several locations to prevent this type of attack from being realized.

Attack and defense: current analysis during PIN comparisons

A technically very interesting attack on comparison features, such as PINs, can be carried out using a combination of physical measurements of a parameter and variation of logical values. This type of attack relates to all mechanisms in which data are sent to the Smart Card and compared in the card with corresponding values, with a retry counter being incremented according to the result of the comparison.

The attack works on the principle of measuring the current drawn by the card, for example by measuring the voltage drop across a resistor in the V_{CC} lead. If a suitable command containing the comparison data is sent to the card, it is possible to see from the current measurement whether the retry counter has been incremented, even before the return code has been received. The comparison value can be determined using this method if the return code is sent before the retry counter is written when the result of the comparison is positive. This can be done by sending all possible variations of the comparison value to the Smart Card, and cutting off power to the card before the retry counter has been incremented if the result is negative. A positive result can be clearly recognized by the associated return code, which is sent before the retry counter is written.

There are two basic ways to defend against this type of attack. The simplest defense consists of always incrementing the retry counter before making the comparison, and then reducing its value afterwards if necessary. Regardless of when the attacker cuts off the power to the card, it is not possible to gain any advantage, since the retry counter will have already been incremented. The second defense is more complicated, but it satisfies the same protective function. In this approach, the retry counter is incremented following a negative comparison and written to an unused EEPROM cell following a positive comparison. Both of these write accesses occur at the same time within the process, so the attacker can draw no conclusions with regard to the result of the comparison. He learns the result of the comparison only after receiving the return code, and at this point it is too late to prevent a write access to the retry counter by cutting off the power.

Attack and defense: timing analysis of PIN comparisons

Programmers always give a lot of attention to making programs execute as quickly as possible. Normally this is also an important consideration. However, the fact that the execution time of a process has been minimized can be used for an attack that definitely has a good chance of succeeding. If a PIN is sent to a Smart Card for comparison, the responsible comparison routine normally compares the PIN that it has received with the stored PIN value byte by byte. A programmer who is not security-conscious would program this routine such that the first difference between the two compared values would cause the

routine to immediately terminate and jump back to the calling program. This causes minute differences in the execution time for the comparison, which can definitely be measured using suitable equipment (such as a storage oscilloscope). This information could be used by an attacker to determine the otherwise secret PIN code in a relatively simple manner.

This type of attack was still effective with Smart Cards up to a few years ago. Nowadays, this is a known type of attack, and the comparison routines are constructed such that all digits of a PIN are always compared. This means that there is no time difference between positive and negative comparison results.

Protection: noise-free cryptographic algorithms

The security of a Smart Card application is based on the secret keys used for the cryptographic algorithm. In order to perform certain types of access or take certain actions with the card, the terminal must always first authenticate itself with the help of a secret key. It is understandable that the authentication of the terminal by the card is an interesting target for an attacker. The authentication of a Smart Card by the terminal is not interesting with regard to an attack on the card, since a Smart Card can be manipulated as desired using a (dummy) terminal.

The Smart Card authenticates the terminal by sending it a random number, which the terminal then encrypts and returns to the card. The Smart Card then performs the same encryption and compares the result with the value received from the terminal. If they match, the terminal has been authenticated and it receives a corresponding return code. If the authentication fails, the card sends a different return code. The starting point for the attacker is an analysis of the processing time between when the command is sent and when the response is returned by the Smart Card.

In the early 1990s, there were still some cryptographic algorithms in use whose execution times were significantly dependent on the key and plaintext involved. Based on the reduction in the key space that this represents, an attacker can use a brute force attack to search for the secret key. How long the search will take depends strongly on how noisy the algorithm is. The larger the time differences are, the smaller is the key space, and thus the easier and faster the search for the key. If the exact implementation of the algorithm in question on the target computer is known, this information can also be included as a reference for generating the timing tables. This sort of attack was made public under the name 'timing attack' in a publication by Paul Kocher in 1995 [Kocher 95], which is primarily concerned with time dependences of the RSA and DSS algorithms.

In principle, a timing analysis is very dangerous threat to the security of a Smart Card. However, since this type of attack has been known for a relatively long time, all present-day Smart Cards employ only noise-free cryptographic algorithms, which means that the time taken for encryption and decryption is independent of the input values. This blocks this sort of attack. However, the programmer has conflicting interests in this regard, since a noise-free algorithm usually requires more program code and is always slower than the noisy version. The reason for this is that a noise-free algorithm must be designed such that the path through the program is equally long for all plaintext data, ciphertext data and keys. This means that the longest necessary path is the reference value, and all other paths must be suitably modified to match this length.

To provide additional security, in some applications all authentication keys have their own retry counters, so that only a limited number of unsuccessful authentications can be carried out. Once the retry counter has reached its maximum value, the Smart Card blocks all further attempts at authentication.

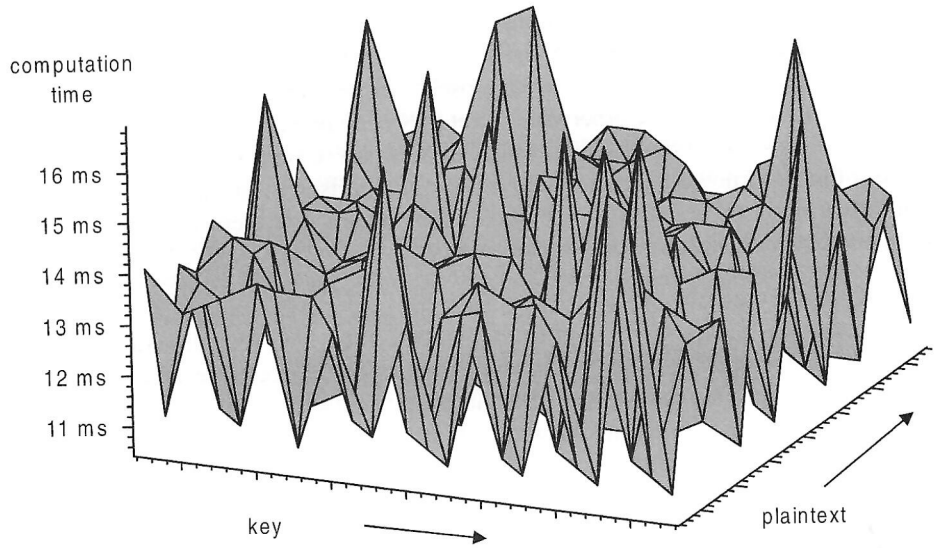


Figure 8.37 Example of the effects of ciphertext and plaintext data on a noisy cryptographic algorithm. The diagram shows a portion of the plaintext and ciphertext space. It was generated using an old implementation of the DES algorithm, with 100,000 iterations per measurement value.

Manipulation: differential fault analysis (DFA)

As is well known, the operation of electronic devices can be considerably impaired when they are exposed to electromagnetic disturbances. For instance, a mobile telephone can cause the processors of many types of small computer-controlled appliances to crash. The cause lies in the memory cells, whose contents can be altered by the high-frequency AC fields.

In 1996, Dan Boneh, Richard DeMillo and Richard Lipton published a study [Boneh 96] that describes a theoretical model for the determination of secret keys for asymmetric cryptographic algorithms by introducing scattered hardware faults. Since the three discoverers of this method worked at the Bell Communications Research (Bellcore) Laboratories at the time, this type of attack is often called the Bellcore attack.

Only two months later, Eli Biham and Adi Shamir published an extension of the Bellcore attack called *differential fault analysis* (DFA) [Biham 96], which also included symmetric cryptographic algorithms such as the DES. This meant that, at least in theory, many Smart Card applications were threatened with a new and serious form of attack.

The basic principle of both of these attacks is relatively simple. In the first step, an arbitrary plaintext is encrypted using the key to be broken, and the resulting ciphertext is saved. Following this, the operation of the card is disturbed while it is processing the cryptographic algorithm, for example by exposing it to ionizing radiation or high-frequency radiation, so that a single bit of the key in an arbitrary location is altered while the computation is being carried out. This yields a ciphertext that is incorrectly encrypted, due to the changed bit. This process is repeated many times, and all the results are saved for analysis. The rest of the process of determining the value of the key is purely mathematical. It is comprehensively described in the papers just mentioned.

The strength of this attack is primarily due to the fact that it is not even necessary to know the location of the altered bit in the secret key. Biham and Shamir state in their publication that with a single corrupted key bit, 200 ciphertext blocks are sufficient to compute the value of the secret DES key. If a triple DES (with a 168-bit key) is used in place of a simple DES, the number of required ciphertexts does not increase significantly. Even if more than one bit is altered, this attack remains effective; the only consequence is that more incorrectly encrypted ciphertexts are needed.

However, this type of attack is in practice not as simple as it sounds. If at all possible, only one bit should be altered, or at least only very few bits. However, if the entire microcontroller is simply bathed in microwave radiation, in most cases so many bits will be altered that the processor will generally crash without a chance. Consequently, attempts are made to induce the processor to make isolated computational errors by injecting specially prepared glitches[10] into the power or clock supplies. If the filter on the associated input leads cannot neutralize these glitches, they can produce the desired processing errors.

However, a Smart Card is not totally helpless in the face of either a Bellcore attack or DFA, as long as suitable preventative measures are employed. The simplest defense is to just compute the cryptographic algorithm twice and compare the two results. If they are identical, there has been no external attempt to alter any bits. This assumes that intentionally introduced scattered errors can never alter the same bit twice in a row. This is also a realistic assumption, since if it were ever to become possible to selectively alter specific bits in a Smart Card processor, then attacks that are much simpler and faster than DFA would be possible. The main disadvantage of double computation is the additional time that it requires, which can cause problems. This applies primarily to attacks on time-consuming asymmetric cryptographic processes such as RSA and DSS.

Another effective defensive measure against differential fault analysis can be achieved by always encrypting different plaintexts. The simplest solution is to prefix the plaintext to be encrypted with a random number. This means that the cryptographic algorithm always encrypts different data, which makes DFA no longer possible.

In summary, the Bellcore attack and differential fault analysis are unquestionably dangerous types of attack that can succeed with Smart Cards that do not have suitable protective measures. However, all Smart Card operating systems and applications were modified to protect them against these attacks shortly after they became known, so that neither the Bellcore attack nor DFA currently represents a serious threat.

Protective elements: Smart Card operating systems

Protective mechanisms in the hardware form the basis for the protective mechanisms in the operating system software. No potential weakness may be overlooked, since the three parts of the protective mechanisms – hardware, operating system and application – are linked in a logical AND relationship. This is comparable to a chain, in which the weakest link determines its breaking strength. If a particular mechanism fails in a Smart Card, the complete security collapses. In particular, the operating system is in turn the basis for the actual application, whose information and processes must be protected.

The following paragraphs deal specifically with protective measures against typical attacks, rather than general Smart Card security functions. However, most of these functions also contribute significantly to operational security and protection against attacks. For this reason, you are explicitly referred to the appropriate sections of Chapter 5.

10 A glitch is a very brief voltage dropout or voltage spike.

Protection: hardware and software tests following a reset
When the operating system is initialized, at least the most important parts of the hardware must be tested to see if they are in proper working order. A test of the RAM is for example indispensable, since all access conditions are stored in the RAM as long as the chip is operating, and the failure of a single bit could cause the entire security to collapse. The calculation and comparison of checksums for the most important parts of the ROM and EEPROM is equally necessary. The CPU is at least implicitly tested by sending the ATR, since the bulk of all possible machine instructions must be executed faultlessly for this to be possible. Explicit tests of the CPU, or any NPU that may be present, can usually be limited to sample testing, since completely testing all functions for flawless operation would take too much time and code.

If the operating system discovers a hardware or checksum error, there are two possible ways to proceed. The first option is for the software to immediately jump to an endless loop, which means that the ATR cannot be sent and no subsequent commands can be received. The main disadvantage of this is that the cause of this behavior cannot be recognized from the outside. You cannot tell, for example, whether the problem is a broken bonding wire, a fractured chip or a checksum error in the EEPROM. The second option, in which the Smart Card attempts to send a special ATR before switching itself off or entering an endless loop, is thus better. The error ATR at least gives the outside world an indication of what has happened inside the Smart Card. However, it must not be overlooked that just sending an error ATR requires a largely functional CPU, a few bytes of RAM and several hundred bytes of program code in the ROM.

Protection: layer separation in the operating system
Layer separation, with clearly defined transfer parameters between the individual layers, is a sign of a stable and robust Smart Card operating system. The consequences of design or programming errors that may be present in the operating system are minimized by a clean separation of the layers within the operating system. This naturally does not mean that such errors will not occur, but the effects of the errors will not be as extensive as in an operating system that is programmed in very compact and condensed code. Separation of the layers makes it difficult for any error that occurs in one layer to propagate to other layers.

Protection: controlling data transfers
Another very important element of security is to control the data transfer process in order to protect the memory against unauthorized accesses. The entire communication to and from the Smart Card takes place via an I/O interface that is controlled by the operating system. No other form of access is possible. This represents an effective form of memory protection in the Smart Card, since this ensures that the operating system always retains control of accesses to the memory regions.

The transmission protocol, which is controlled by the transport manager, must intercept all possible incorrect inputs. There must be no possibility of influencing the data transfer process by manipulating transfer blocks in order to cause data to be sent from the memory to the terminal without proper authorization.

Protection: checksums for important memory contents
The file structure, and in particular the file headers (or in other words the file descriptors), should be protected by means of checksums. This makes it possible for the operating system to at least detect any unintentional change to the data stored in memory. This

requirement is especially important in light of the fact that the object-oriented access conditions for each file are stored in this part of the file.

All memory regions of the EEPROM that are vitally important for the Smart Card operating system must be protected using checksums (EDCs). Whenever such a region is accessed or the code it contains is called to be executed, the consistency of its contents must be verified before the access or code execution is allowed to proceed.

Protection: encapsulation of applications
Some operating systems encapsulate the individual DFs that contain the applications along with their files, so that individual applications are isolated from each other. This is however based on software protection alone, with no support from the hardware of the chip. The level of protection is thus not as high as it could be. Nonetheless, even this software approach to application encapsulation can be very advantageous in case of an error, since it makes it impossible for the file manager to exceed the boundaries of a DF without an explicit prior selection. The consequences of a memory error in a file are thereby at least limited to the associated DF.

If a memory management unit (MMU) is present in the hardware to provide hardware support to the operating system, different applications can be fully isolated from each other. In this case, even manipulated software within an application cannot obtain unauthorized access to the memory regions of other applications.

Protection: Camouflaging the activities of the operating system
Whenever data have to be written to the EEPROM, the charge pump in the chip must first be switched on. This increases the current consumption of the chip, which can easily be detected using simple measuring equipment. The design of the operating system must thus take into account the fact that it is always possible to determine externally when EEPROM write accesses occur. The software in the Smart Card must prevent an attacker from being able to take advantage of this.

It is therefore very important that it should not be possible to draw any useful conclusions about the processes and decisions in the machine program from measurements of the current drawn by the card. It would for example be fatal if were possible to use such measurements to reliably judge the outcome of a PIN comparison before the command execution has been completed and the return code sent. This information could very easily be used to analyze the value of the PIN.

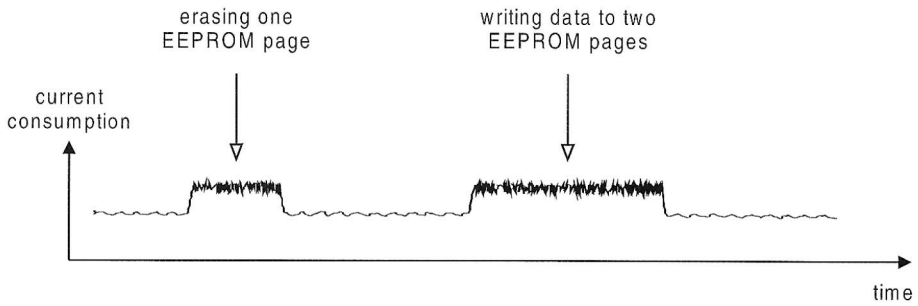


Figure 8.38 Approximate representation of the variation in the current consumption of a Smart Card when the charge pump is switched on

Protection: object-oriented access conditions
Early Smart Card applications were always based on two centrally managed access mechanisms. A disadvantage of centralized access controllers is that software or memory errors can affect the overall security of the Smart Card. Modern object-oriented file management systems, with access conditions stored in the individual files, have the advantage that only a single file is affected by a memory error, while the security of all other files remains intact. This is actually a fundamental property of all distributed systems. They are somewhat more difficult to program, but they provide significantly stronger security against attacks and errors due to their self-sufficiency.

Attack and defense: random number generators
The random numbers generated by the Smart Card are used in authentication to individualize a session, which means to make each session unique and different from all preceding and following sessions. The objective of this is to make it impossible to successfully replay data that has been obtained by tapping into a previous session. Another form of the attack would be to let the Smart Card generate so many random numbers that their sequence becomes predictable. Yet another possibility is to keep requesting random numbers from the Smart Card until the EEPROM memory of the random number generator no longer works properly, so that the same number is generated over and over again.

Any of these attacks could, if successful, bypass the authentication of the terminal by the Smart Card. Without exception, they work only with the first generation of Smart Cards. With modern operating systems, they will all fail. The cycle length of the random numbers that are generated is nowadays so great that the same random number never appears twice within the lifetime of an individual Smart Card. It is also no longer of any use to generate so many random numbers that problems start to occur with the EEPROM. If this happens, the generation of random numbers is simply blocked and any further authentication is thus prevented.

A high-quality random number generator must meet some additional requirements, such as producing non-predictable random numbers and having a long cycle length (the number of numbers that are generated before the generator repeats itself). In addition, all Smart Cards within a particular application must generate different random numbers. This may sound extremely obvious, but problems have repeatedly occurred in the past in this regard! This differing behavior is achieved by entering a starting value for the pseudorandom number generator when the Smart Card is initialized or personalized. This starting value is often called a *seed number*, in allusion to a biological seed that determines the growth of a plant. The design and evaluation criteria for pseudorandom number generators are extensively discussed in Section 4.9 ('Random Numbers'), along with methods to measure the quality of random numbers.

Protective elements of the Smart Card application
The protective mechanisms of the application are based on the corresponding mechanisms of the hardware and the operating system. The application is dependent on the extent to which these two lower levels fulfill their obligations with regard to protection, since it cannot correct for any errors in the hardware or the operating system. For example, if it is possible to read the contents of the EEPROM using an analysis procedure, even the most complicated and secure encryption processes are of no use at all, since the keys can be taken directly from the EEPROM by an attacker. An application must nevertheless be

constructed such that the entire system is not compromised if an individual card is compromised.

Protection: simple mechanisms
In order to provide effectively protection against attacks, all mechanisms of an application should be designed to be as uncomplicated as possible, and they should always conform to the generally applicable principle of 'keep it simple'. To start with, this makes the implementation easier, and later on, it makes it easier to check the correct operation of the mechanisms and the effectiveness of protective mechanisms. It is extremely dangerous to assume that if something is only made complicated enough, it can thereby be protected against all possible attacks. As a rule, exactly the opposite is true. Complicated processes and mechanisms usually result in various things being forgotten or overlooked, which makes things that much easier for an attacker.

Fundamentally, the protective mechanisms provided by the operating system should always be utilized by the application. These have been tested for reliability, and the defense that they provide starts at a lower software level than that of the application. This is not intended to mean that an application does not need to have any protective mechanisms of its own, but the mechanisms that are already present in the operating system should always be used.

Protection: conservative access privileges
In addition to the principle of 'keep it simple', there is a second generally valid rule. This is that access privileges for the files and commands of a Smart Card should be granted as conservatively as possible. Access should be generally prohibited, and only allowed when it is absolutely necessary.

The advantages of this approach are that it makes it less likely that access to important data and commands will be granted unintentionally, and that it costs an attacker additional effort to obtain each piece of necessary information. This can considerably reduce the attractiveness of an attack, since it increases the overall amount of effort that is required.

Protection: state machines for command sequences
Attacking a Smart Card application is considerably more difficult if it is not possible to execute every command at any desired time an unlimited number of times. This can be realized by using a state machine that determines the allowed sequences of commands. If for example a mutual authentication of the terminal and the Smart Card is specified as the first required action, an attacker will have to overcome this protective barrier before he or she can execute any further commands.

Protection: double access security
The attacker's job is made considerably more difficult if the Smart Card files are protected not only by access conditions stored in the objects, but also by the use of a command state machine with predefined command sequences and allowable parameters. With this, the attacker cannot discover the specific features of the system by just trying each command or combination of commands in turn. If the command sequences are supervised by a state machine, only the commands defined in the application can be executed in the Smart Card. All other commands will be blocked in principle by the state machine. This considerably reduces the scope of the possibilities available to an attacker with regard to command manipulation.

Protection: various test levels

With bank notes, it has been standard practice for many years to support various test levels. This has to do with security features that can be checked by different groups of people or different types of machines, independent of each other. For example, many of the visual features, such as security threads and watermarks, can be checked by anyone on the street. For the checking the next level, you need an ultraviolet lamp in order to be able to recognize the fluorescent pigments in the paper. The features belonging to the next higher testing level are used by automated equipment to verify that the notes are genuine. A typical example is the infrared characteristics of the bank note. Yet other level of independent features is provided for tests performed by the central bank.

This entire system can easily be transferred to Smart Cards, with the logical consequence that not everybody or every piece of equipment can check all of the features. It is thus conceivable, for example, that a retail terminal for an electronic purse system could contain only some of the keys used for signature verification, rather than all of them. This does not weaken the system in a cryptographic sense, and it has the advantage that an attacker cannot compromise the entire system by learning the master key of a retail terminal. The only entity that knows all the keys in the system that are necessary for a complete transaction data set is the system operator, who thus can always recognize an attack by noting that the signatures are forged, and who can then introduce suitable countermeasures.

Protection: security features

Features incorporated into the microcontroller can offer additional security when a Smart Card is used. These are supplementary functional elements that are added to the microcontroller and that can be tested by the terminal in parallel with the software in the chip. Both analog and digital elements are used for this purpose. The security of these features is based on concealment, and the features are different for each application, which means that the chips are application-specific.

Protection: secure data transfers

There are certain risks involved in transferring data in potentially insecure surroundings. With relatively simple technical manipulations of the interface between the terminal and the card, it is possible to insert or delete almost any desired data within the normal data steam during a session. If this happens while sensitive data related to security are being transferred, an attacker could take advantage of this. In order to prevent this type of relatively simple and easily executed attack, a secure messaging procedure can be employed. However, complete encryption of data should be avoided as much as possible, with encryption only being used for the transfer of secret keys, for example. This is because the procedure requires the double encryption of the transferred data, which very sharply reduces the effective data transfer rate. An additional argument against complete data encryption has to do with data protection legislation. Almost all information that is written to the memory of a Smart Card is public. If this information is encrypted, nobody can then check what is actually written to or read from the card. In order to avoid any suspicions about the encrypted data, which in principle would be justified, the data should as much as possible remain unencrypted while it is being transferred.

Protection: error recovery functions
If a session is terminated for an undefined reason, or there are fundamental uncertainties regarding an earlier session, it is a major advantage if application-specific log files are present in the Smart Card. These files are maintained by the operating system, which updates them regularly during the session to reflect the current state of the application and any signatures or other data that may have been received from the terminal. The logged data are located in a cyclic file, in which the oldest record is always overwritten each time a new entry is made, so that the content of the oldest record is lost. If this file contains twenty records, for example, then information regarding the last twenty sessions can be stored for subsequent analysis of the course of each session. This information can be used to clear up a lot of uncertainties and to unambiguously clarify contested transactions and processes.

An additional argument for detailed log files in the Smart Card is the fact that they make certain error recovery functions possible. With a log file, it is possible to automatically restore the old state of the card (a *roll back*) in case the session is terminated in an undefined manner. This would otherwise require the exact procedure and sequence of events to be analyzed, possibly even with human help.

Protection: authentication
Unilateral authentication, which is well known due to its use with magnetic-stripe cards, ultimately amounts to nothing more than verification by the terminal that the card is genuine. A magnetic-stripe card, due to its passive nature, cannot verify the genuineness of a terminal. The introduction of Smart Cards has fundamentally changed this situation. Now the card can also test whether it has been inserted into a genuine terminal or is connected to a genuine background system. This has extensive consequences with regard to security, since it makes it possible for the card to also take active measures against unauthorized access attempts.

The possibilities that arise from the ability of the card and the terminal to perform a mutual authentication are substantial, and they are usually not exploited at anywhere near their full potential. A Smart Card should at minimum refuse to allow any further access attempts as long as the terminal cannot properly authenticate itself. This would make it impossible to undertake any sort of analysis of the Smart Card operating system in private, even if only to find out what commands are present.

Protection: on-line behavior
Terminals with integrated security modules can be used on their own to run applications with Smart Cards. Of course, periodic uploads and downloads to and from the background system are still necessary, but these usually occur only infrequently. With relatively large applications that have a large number of cards in circulation, however, it must at least be possible for the terminal to quickly make a connection to the background system if necessary, in order to provide direct end-to-end communication between a Smart Card and the background system. The importance of this increases with the size of the system and the advantages that an attacker could obtain by means of fraud. This is because a direct communications link to the Smart Card allows the background system to access the current database of the card, and if necessary to block the card. In addition, the keys in the background system are significantly more secure than those located in the many terminals in the field, even if the terminals have security modules. The background system can make very good statistical evaluations of the card data that it receives via sporadic end-to-end links to the Smart Cards.