CHAPTER

14

Physical Tamper Resistance

It is relatively easy to build an encryption system that is secure if it is working as intended and is used correctly but it is still very hard to build a system that does not compromise its security in situations in which it is either misused or one or more of its sub-components fails (or is 'encouraged' to misbehave) ... this is now the only area where the closed world is still a long way ahead of the open world and the many failures we see in commercial cryptographic systems provide some evidence for this.

—BRIAN GLADMAN

14.1 Introduction

The techniques discussed in the previous few chapters—physical protection involving barriers, sensors, and alarms—are often used to protect critical information processing resources:

- A bank's main servers will typically be kept in a guarded computer room.
- The seismic sensor packages used to detect unlawful nuclear tests may be at the bottom of a borehole several hundred feet deep which is backfilled with concrete.
- A hole-in-the-wall automatic teller machine is in effect a PC in a one-ton safe with a number of fancy peripherals. These include not just banknote dispensers but also temperature sensors to detect attempts to cut into the device, and accelerometers to detect if it's moved. An alarm should cause the immediate erasure of all crypto key material in the device.

But often it's inconvenient to use a massive construction, and this has spawned a market for portable tamper-resistant processors. These range from smartcards, which typically perform a limited set of operations in support of an application such as pay-

TV; through tamper-resistant cryptoprocessors, which are fitted into the servers that manage PINs in cash machine networks; to elaborate high-end devices used in military command and control.

I should note that there is some similarity between tamper-resistant devices and replicated devices. If a service is implemented as a number of different servers in different sites that perform transactions simultaneously and vote on the result, then it may be possible to provide a high level of integrity protection against many types of attack. The secret sharing schemes I discussed in Section 11.4 can also provide confidentiality for key material. But tamper-resistant devices can at least in theory provide confidentiality for the data too. This is one respect in which the principle that many things can be done either with mathematics or with metal, breaks down.

14.2 History

The use of tamper resistance in cryptography goes back for centuries [428]. Naval code-books have been weighted so they could be thrown overboard and sink in the event of imminent capture; to this day, the dispatch boxes used by British government ministers' aides to carry state papers are lead-lined so they will sink. Codes and, more recently, the keys for wartime cipher machines have been printed in water-soluble ink; Russian one-time pads were printed on cellulose nitrate, so that they would burn furiously if lit; and one U.S. wartime cipher machine came with self-destruct thermite charges so it could be destroyed quickly if need be.

But such mechanisms depended on the vigilance of the operator, and key material was often captured in surprise attacks. So attempts were made to automate the process. Early electronic devices, as well as some mechanical ciphers, were built so that opening the case erased the key settings.

Following a number of cases in which cipher staff sold key material to the other side—such as the notorious Walker family in the United States—engineers paid more attention to the question of how to protect keys in transit as well as in the terminal equipment itself. The goal was 'to reduce the street value of key material to zero', and this can be achieved either by *tamper resistant* devices from which the key cannot be extracted, or *tamper evident* ones from which key extraction would be obvious.

Paper keys were once carried in "tattle-tale containers," designed to show evidence of tampering. When electronic key distribution came along, a typical solution was the "fill gun," a portable device that would dispense crypto keys in a controlled way. Nowadays, this function is usually performed using a small security processor such as a smartcard. Control protocols range from a limit on the number of times a key can be dispensed, to mechanisms that use public key cryptography to ensure that keys are loaded only into authorized equipment. The control of key material also acquired broader purposes. In both the United States and Britain, it was centralized and used to enforce the use of properly approved computer and communications products. Live key material would only be supplied to a system once it had been properly accredited.

Once initial keys have been loaded, further keys may be distributed using various kinds of authentication and key agreement protocols. I already talked about many of the basic tools, such as key diversification, in Chapter 2, "Protocols," and I'll have more to say on protocols later in this chapter. Let's first look at the physical defenses against tampering.

14.3 High-End Physically Secure Processors

An example worth studying is the IBM 4758 (Figure 14.1). This is important for two reasons. First, it is the only commercially available processor to have been successfully evaluated (at the time of writing) to the highest level of tamper resistance (FIPS 140-1 level 4) [576] set by the U.S. government. Second, there is extensive literature about it available in the public domain, including the history of its design evolution, its protection mechanisms, and the transaction set it supports [718, 795, 796].

The evolution that led to this product is briefly as follows. From the earliest days of computing, computers were protected physically because of their high value. However, the spread of multi-user operating systems in the 1960s, and the regularity with which bugs were found in their protection mechanisms, meant that large numbers of people might potentially have access to the data being processed. With particularly sensitive data—such as long-term cryptographic keys and the personal identification numbers (PINs) used by bank customers to identify themselves to cash machines—it was realized that the level of protection available from commercial operating systems was likely to remain insufficient.

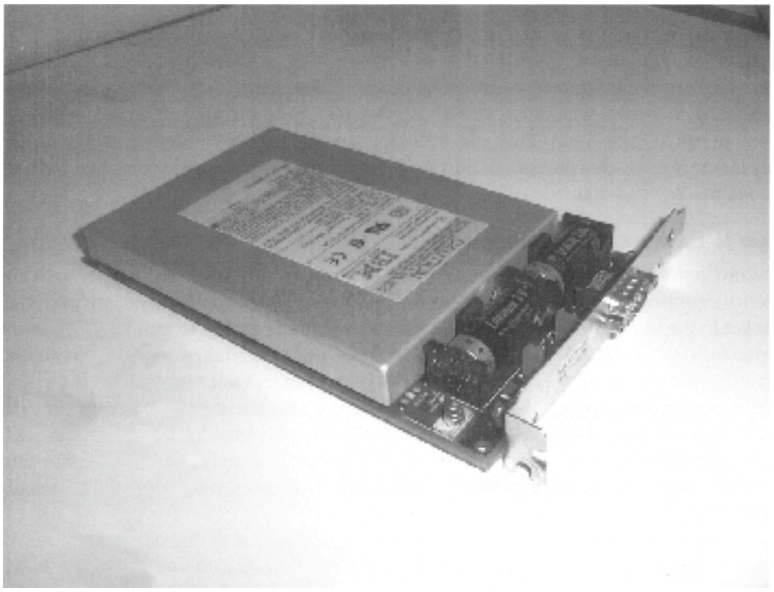

Figure 14.1 The IBM 4758 cryptoprocessor (courtesy of Steve Weingart).

Figure 14.2 The 4758 partially opened, showing (from top left downward) the circuitry, aluminium electromagnetic shielding, tamper-sensing mesh and potting material (courtesy of Frank Stajano).

This led to the development of standalone *security modules* of which the first to be commercially successful were the IBM 3848 and the VISA security module. Both of these were microcomputers encased in robust metal enclosures, with encryption hardware and special *key memory*, which was static RAM designed to be zeroized when the enclosure was opened (Figure 14.2). This was accomplished by wiring the power supply to the key memory through a number of lid switches. The device operator would then have to reload the key material.

How to Hack a Cryptoprocessor (1)

The obvious attack on such a device is for the operator to steal the keys. In early banking security modules, the master keys were kept in PROMs which were loaded into a special socket in the device, to be read during initialization, or as strings of numbers that were typed in at a console. The PROMs could easily be pocketed, taken home and read out using hobbyist equipment. Cleartext paper keys were even easier to steal.

The fix was shared control—to have two or three PROMs with master keys, and make the device master keys the exclusive-or of all the components. These devices can then be kept in different safes. (With the virtue of hindsight, the use of exclusive-or for this purpose was an error, and a hash function should have been used instead. I'll explain why shortly.)

However, this procedure is somewhat tedious, and may well degrade as it becomes routine. In theory, when a device is maintained, its custodians should open the lid to erase the live keys, let the maintenance engineer load test keys, and then re-load live keys afterwards. But the managers with custodial responsibility will often give the PROMs to the engineer rather than bothering with them. I've even come across cases of the master keys for an automatic teller machine being kept in the correspondence file in a bank branch, where any of the staff could look them up. Thus, the goal was to

minimize the number of times that a reload would be necessary, such as for maintenance or following a power failure. So security modules typically have batteries to back up the mains power supply (at least to the key memory). This meant that in practice, the custodians had to load the keys only when the device was first installed, and after occasional maintenance visits subsequently.

It has been debated whether frequent or infrequent key loading is best. If key loading is very infrequent, the responsible personnel will likely never have performed the task before, and may either delegate it out of ignorance or be hoodwinked by a more technically astute member of staff into doing it in an insecure way (see [19] for a case history of this). The modern trend is toward devices that generate keys (or have them loaded) in a secure facility after manufacture but before distribution. Such keys may be kept on smartcards and used to bootstrap the keying of more substantial devices.

How to Hack a Cryptoprocessor (2)

Early devices were vulnerable to attackers cutting through the casing, and to maintenance engineers who could disable the lid switches on one visit and extract the keys on the next. Second-generation devices dealt with the easier of these problems, namely physical attack, by adding more sensors such as photocells and tilt switches. These may be enough for a device kept in a secure area to which access is controlled. But the hard problem is to prevent attacks by the maintenance staff.

The strategy adopted by many of the better products is to separate all the components that can be serviced (such as batteries) from the core of the device (such as the tamper sensors, crypto, processor, key memory, and alarm circuitry). The core is then "potted" in a solid block of a hard, opaque substance such as epoxy. The idea is that any physical attack will be "obvious" in that it involves an action such as cutting or drilling, which can be detected by the guard who accompanies the maintenance technician into the bank computer room.

How to Hack a Cryptoprocessor (3)

However, if a competent person can get unsupervised access to the device for even a short period of time (or if the guard has not been properly trained), then potting the device core is inadequate. For example, it is often possible to scrape away the potting with a knife, and drop the probe from a logic analyzer on to one of the bus lines in the core. Most common cryptographic algorithms, such as RSA and DES, have the property that an attacker who can monitor any bitplane during the computation can recover the key [370]. So an attacker who can get a probe anywhere into the device while it is operating can likely extract secret key material.

So the high-end products have a tamper-sensing barrier whose penetration triggers destruction of the secrets inside. An early example appeared in IBM's ABYSS system in the mid-1980s. This used loops of 40-gauge nichrome wire, which were wound loosely around the device as it was embedded in epoxy, then connected to a sensing circuit [795]. Bulk removal techniques such as milling, etching, and laser ablation break the wire, which erases the keys.

But the wire-in-epoxy technique can be vulnerable to slow erosion using sand blasting; when the sensing wires become visible at the surface of the potting, shunts can be connected around them. So the next major product from IBM, the 4753, used a metal shield combined with a membrane printed with a pattern of conductive ink and surrounded by a more durable material of similar chemistry. The idea was that any attack would break the membrane with high probability.

How to Hack a Cryptoprocessor (4)

The next class of methods an attacker can try involve the exploitation of *memory remanence*, the fact that many kinds of computer memory retain some trace of data that have been stored there. Sometimes, all that is necessary is that the same data were stored for a long time. An attacker might bribe the garbage truck operator to obtain a bank's discarded security modules: as reported in [44], once a certain security module had been operated for some years using the same master keys, the values of these keys were *burned in* to the device's static RAM. On power-up, about 90% of the relevant bits would assume the values of the corresponding keybits, which was more than enough to recover the keys.

Memory remanence affects not just static and dynamic RAM, but other storage media as well. For example, the heads of a disk drive change alignment over time, so that it may be impossible to completely overwrite data that were first written some time ago. The relevant engineering and physics issues are discussed in [362]. The NSA has published guidelines (the *Forest Green Book*) on preventing remanence attacks [243].

The better third-generation devices have *RAM savers*, which function in much the same way as screen savers; they move data around the RAM to prevent it being burned in anywhere.

How to Hack a Cryptoprocessor (5)

A further problem is that computer memory can be frozen by low temperatures. By the 1980s it was realized that below about –20°C, static RAM contents can persist for some time—seconds to minutes—after power is removed. Data remanence gets steadily longer at lower temperatures. So an attacker might freeze a device, remove the power, cut through the tamper sensing barrier, extract the RAM chips containing the keys and power them up again in a test rig. RAM contents can also be *burned in* by ionizing radiation. (For the memory chips of the 1980s, this required a fairly serious industrial X-ray machine; but as far as I'm aware, no-one has tested the current, much smaller, memory chip designs.)

So the better devices have temperature and radiation alarms. These can be difficult to implement properly, as modern RAM chips exhibit a wide variety of memory remanence behaviors, with the worst of them keeping data for several seconds even at room temperature [712]. (This shows the dangers of relying on a property of some component to whose manufacturer the control of this property is unimportant.) Some military devices use protective detonation; there are memory chips potted in steel cans with a thermite charge precisely calculated to destroy the chip without causing gas release from the can.

How to Hack a Cryptoprocessor (6)

The next set of attacks on cryptographic hardware involve either monitoring the RF and other electromagnetic signals emitted by the device, or even injecting signals into it and measuring their externally visible effects. This technique, which is variously

known as *Tempest* or *power analysis*, is such a large subject that I devote the next chapter to it. As far as the 4758 is concerned, the strategy is to have solid aluminium shielding, and to low-pass-filter the power supply to block the egress of any signals at the frequencies used internally for computation.

The 4758 also has an improved tamper-sensing membrane, in which four overlapping zig-zag conducting patterns are doped into a urethane sheet, which in turn is potted in a chemically similar substance so that an attacker cutting into the device has difficulty even detecting the conductive path, let alone connecting to it. This potting surrounds the metal shielding which in turn contains the cryptographic core (see Figure 14.2). The design is described more detail in [718].

I don't know how to attack the hardware of the 4758. IBM declined to sell us samples for attack, but we did come up with a number of ideas after scrutinizing one, such as:

How to Hack a Cryptoprocessor (7)

Here are some speculative ideas about how to break into a 4758.

- The straightforward approach would be to devise some way to erode the protective potting, detect mesh lines, and connect shunts around them. Probably the first thing I'd try is a magnetic force microscope.
- One could invent a means of drilling holes eight millimeters long and only 0.1 millimeters wide (that is, much less than the mesh line diameter). This isn't feasible with current mechanical drills, which are limited to an aspect ratio of 15 or so, and the same holds for laser ablation and ion milling. However I speculate that some combination of nanotechnology and ideas from the oil industry might make such a drill possible eventually. Then one could drill right through the protective mesh with a fair probability of not breaking the circuit.
- Having dismantled a few instances of the device and understood the operation of its hardware, the attacker might use shaped explosive charges to send plasma jets of the kind discussed in Section 11.5 into the device to destroy the tamper-sensing and memory zeroization circuitry before they have time to react.

The success of such attacks is uncertain, and they are likely to remain beyond the resources of the average villain for some time.

When I shipped the first draft of this book in September 2000, I wrote at this point: "So by far the most likely attacks on 4758 based systems involve the exploitation of logical rather than physical flaws." By the time I edited this paragraph at the proof stage, this had come true in spades. Most users of the 4758 use an application called CCA which is described in [388] and contains many features that make it difficult to use properly. Having been suspicious of the complexity of this instruction set, I passed the manual to a new research student, Mike Bond, and asked him if he could find any vulnerabilities. By the middle of November, he had found a number of problems, including a protocol-level attack that enables a capable opponent to extract all the interesting keys from the device. We'll discuss this attack below.

Finally, it should be mentioned that the main constraints on the design and manufacture of security processors are remarkably similar to those encountered with more general alarms. There is a trade-off between the false alarm rate and the missed alarm rate, and thus between security and robustness. Security processors often need to be handled with care; if they self-destruct at temperatures of -20°C, they cannot be dis-

tributed through normal computer industry channels, where goods are often subjected to –40°C in winter. Vibration, power transients, and electromagnetic interference can also be a problem with some designs. Military equipment makers have a particularly tough problem. For example, if exposing the crypto processor of a military tactical radio to radiation causes it to self-destruct, then hardening the device sufficiently might make it too heavy to carry.

14.4 Evaluation

A few comments about the evaluation of tamper-resistant devices are in order before we go on to discuss cheaper devices.

The IBM paper that describes the design of the 4758's predecessor, the 4753 [4], proposed the following classification scheme for attackers:

1. Class 1 attackers—'clever outsiders'—are often very intelligent but may have insufficient knowledge of the system. They may have access to only moderately sophisticated equipment. They often try to take advantage of an existing weakness in the system, rather than try to create one.
2. Class 2 attackers—'knowledgeable insiders'—have substantial specialized technical education and experience. They have varying degrees of understanding of parts of the system but potential access to most of it. They often have highly sophisticated tools and instruments for analysis.
3. Class 3 attackers—'funded organizations'—are able to assemble teams of specialists with related and complementary skills backed by great funding resources. They are capable of in-depth analysis of the system, designing sophisticated attacks, and using the most advanced analysis tools. They may use class 2 adversaries as part of the attack team.

Within this scheme, the 4753 was aimed at blocking knowledgeable insiders, while its successor, the 4758, is aimed at (and certified for) blocking funded organizations.

The FIPS certification scheme is operated by laboratories licensed by the U.S. government and set out in the FIPS 140-1 standard. This sets out four levels of protection, with level 4 being the highest (currently, the 4758 is the only device certified at this level). There is a large gap between level 4 and the next one down, level 3, where only potting is required; this means that attacks which exploiting electromagnetic leakage, memory remanence, drilling, sandblasting, and so on may still be possible. I have handled a level 3 certified device from which I could scrape off the potting with my Swiss army knife! So while FIPS 140-1 level 3 devices can be (and have been) defeated by class 1 attackers in the IBM sense, the next step up—FIPS 140-1 level 4—is expected to keep out an IBM class 3 opponent. There is no FIPS level corresponding to a defense against IBM's class 2.

The original paper on levels of evaluation, written by IBM engineers, had proposed six levels [796]: the FIPS standard adopted the first three of these as its levels 1–3, and the proposed level 6 as its level 4. The gap, commonly referred to as "level 3.5," is where many of the better commercial systems are aimed. Such equipment certainly attempts to keep out the class 1 attack community, while making life hard for class 2, and expensive for class 3.

That said, I am not convinced that the IBM classification is correct. I know of one large funded organization that bought chip-testing equipment, tried to break into a smartcard, and failed; they concluded that smartcards were completely tamper-proof. However, as we shall see shortly, many smartcards have been broken by level 1 attackers. The persistence and cunning of the attacker is far more important than the number of people on his employer's payroll.

14.5 Medium-Security Processors

Good examples of level 3.5 products are the iButton and 5002 security processors from Dallas Semiconductor, and the Capstone chip used to protect U.S. military communications up to Secret. While the 4758 costs $2000, these products cost of the order of $10–20. Yet mounting an attack on them is far from trivial.

14.5.1 The iButton

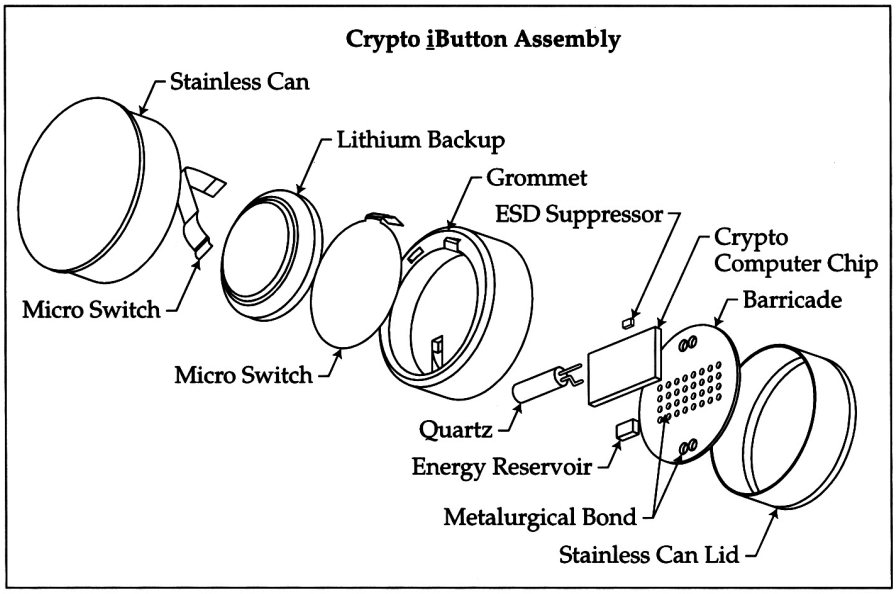

Figure 14.3 iButton internals (courtesy of Dallas Semiconductor Inc.).

The iButton from Dallas Semiconductor is designed to be a minimal, self-contained cryptographic processor. It has an 8051 microcontroller with a modular exponentiation circuit, static RAM for keys and software, a clock, and tamper sensors. These are encased in a steel can with a lithium battery, which can maintain keys in the RAM for a design life of 10 years (see Figure 14.3). It is small enough to be worn in a signet ring or carried as a key fob. An early application was as an access token for the "Electronic Red Box", a secure laptop system designed for use by U.K. government ministers. To access secret documents, the minister had to press his signet ring into a reader at the side of the laptop. (One of the design criteria had been: "Ministers shall not have to use

passwords.") Other applications include the Istanbul mass transit system, parking meters in Argentina, and the electronic stamps for the U.S. Postal Service that I mentioned in the previous chapter [753]. The device is now being equipped with a Java interpreter, and marketed as the Java ring, a tamper-resistant device that users can program for their own applications.

How might an iButton be attacked? The most obvious difference from the 4758 is the lack of a tamper-sensing barrier. So one might try drilling in through the side, then either probe the device in operation or disable the tamper-sensing circuitry. Because the iButton has lid switches to detect the can being opened, and its processor is mounted upside-down on the circuit board (with a mesh in the top metal layer of the chip), this is unlikely to be a trivial exercise. It might well involve building custom jigs and tools. In short, it's a tempting target for the next bright graduate student who wants to win their spurs as a hardware hacker.

14.5.2 The Dallas 5002

Another medium-grade security device from Dallas is the DS5002 microcontroller, which is widely used in devices such as point-of-sale terminals, where it holds the keys used to encrypt customer PINs.

The ingenious idea behind this device is *bus encryption*. The chip has added hardware that encrypts memory addresses and contents on the fly as data are loaded and stored. This means that the device can operate with external memory, and is not limited to the small amount of RAM that can be fitted into a low-cost tamper-sensing package. Each device has a unique master key, which is generated at random when it is powered up. The software is then loaded through the serial port, encrypted, and written to external memory. The device is then ready for use. Power must be maintained constantly, or the internal register that holds the master key will lose it; this also happens if a physical tampering event is sensed (like the iButton, the DS5002 has a tamper-sensing mesh built into the top metal layer of the chip).

An early version of this processor (1995) fell victim to an ingenious protocol level attack by Markus Kuhn, the *cipher instruction search attack* [477]. The idea is that some of the processor's instructions have a visible external effect such as I/O. In particular, one instruction causes the next byte in memory to be output to the device's parallel port. The trick is to intercept the bus between the processor and memory using a test clip, and feed in all possible 8-bit instruction bytes at some point in the instruction stream. One of them should decrypt to the parallel output instruction, and output the plaintext version of the next "encrypted memory" byte. By varying this byte, a table could be built up of corresponding plaintext and ciphertext. After using this technique to learn the encryption function for a sequence of seven or eight bytes, the attacker could encipher and execute a short program to dump the entire memory contents.

The full details are a bit more intricate. Dallas has since fixed the problem, but it is a good example of the completely unexpected things that go wrong when trying to implement a clever new security concept for the first time.

14.5.3 The Capstone/Clipper Chip

In 1993, the security world was convulsed when the U.S. government introduced the Clipper chip as the replacement for DES. Clipper, also known as the Escrowed Encryption Standard (EES), was a tamper-resistant chip that implemented the Skipjack block cipher in a protocol designed to allow the U.S. government to decrypt any traffic encrypted using Clipper. The idea was that when a user supplied Clipper with a string of data and a key with which to encrypt it, the chip returned not just the ciphertext but also a Law Enforcement Access Field, or LEAF, which contained the user-supplied key encrypted under a key embedded in the device and known to the government. To prevent people cheating and sending along the wrong LEAF with a message, the LEAF had a cryptographic checksum computed with a "family key," shared by all interoperable Clipper chips. This functionality was continued into the next-generation chips, called Capstone, which incorporate ARM processors to do public key encryption and digital signature operations.

Almost as soon as Capstone chips hit the market, a vulnerability was found in the LEAF mechanism [113]. The cryptographic checksum used to bind the LEAF to the message was only 16 bits long, making it possible to feed random message keys into the device until one with a given LEAF was found, thus enabling a message to be sent with a LEAF that would remain impenetrable to the government. The Clipper initiative was abandoned and replaced with other policies aimed at controlling the "proliferation" of cryptography. Nevertheless, Capstone quietly entered government service and is now widely used in the Fortezza card, a PCMCIA card used in PCs to encrypt data at levels up to Secret. The Skipjack block cipher, which was initially classified, has since been placed in the public domain [577].

Of greater interest here are the tamper protection mechanisms used, as they are perhaps the most sophisticated in any single-chip tamper resistant device, and were claimed at the time to be sufficient to withstand a "very sophisticated, well-funded adversary" [578]. Although the NSA claimed that the Clipper chip would be unclassified and exportable, I've not been able to get hold of one for dismantling, despite repeated attempts.

Its successor is the QuickLogic military FPGA, designed to enable its users to conceal proprietary algorithms from their customers; it is advertised as being "virtually impossible to reverse-engineer." Like Clipper, it uses *vialink read-only memory* (VROM), in which bits are set by blowing antifuses between the metal 1 and metal 2 layers on the chip. A programming pulse at a sufficiently high voltage is used to melt a conducting path through the polysilicon that separates the two metal layers. Further details and micrographs can be found in the data book [347].

There are basically three approaches to reverse engineering an antifuse FPGA.

- The first is to determine the presence of blown antifuses using optical or electron microscopy, having first removed the top metal layer of the chip. This can be extremely tedious; even if the bits are read out correctly, a lot more work remains to figure out what they mean.
- A smarter approach is to abuse the programming circuit. This sends a pulse to the fuse and stops it once the resistance drops, which means that the metal has melted and established contact; if the pulse isn't stopped, the metal might vaporize and go open-circuit again. Thus, circuits for detecting whether a fuse is

open or short must be provided; and if they aren't sufficiently disabled after programming, they can be used to read the device out.

- The fastest approach, which is particularly easy when the cryptographic algorithm being executed is known, is to drop microprobes directly on to the gate array and look at the signals. Suitable analysis techniques, such as those described in Section 15.4, should quickly yield the key. Signals can also be collected using electromagnetic or electro-optic sensors, voltage contrast microscopy and a growing number of other chip-testing techniques. Even where the algorithm isn't known initially, it may be faster to reconstruct it from observing on-chip signals than from doing a full circuit reconstruction.

This technology isn't infallible, but used intelligently it certainly appears to have some potential.

14.6 Smartcards and Microcontrollers

The most common secure processors nowadays are smartcards and similar self-contained security processors. These cost maybe a dollar each in bulk, and are being deployed in environments such as telephone cards, pay-TV subscriber cards, hotel door locks, and even (in some countries) bank cards.

In such applications, the opponent can often obtain many sample devices and take them away to probe at their leisure. As a result, many attacks on them have been developed.

Although they are now being marketed as the "new" security solution, smartcards actually go back a long way, with the early patents (which date from the late 1960s through mid-1970s) having long since expired [247]. For a history of the development of smartcards, see [358]. For many years, they were mostly used in France, where much of the original development work was done with government support. In the late 1980s and early 1990s, they started to be used on a large scale outside France, principally as the *subscriber identity modules* (SIMs) in GSM mobile phones and as subscriber cards for pay-TV stations.

A smartcard is a self-contained microcontroller, with a microprocessor, memory and a serial interface integrated on to a single chip that is packaged in a plastic card. Smartcards used in banking and in the older mobile phones use a standard-size bank card, while in the newer, smaller mobile phones, a much smaller size is used. Smartcard chips are also packaged in other ways. For example, most U.K. prepayment electricity meters use them packaged in a plastic key, as do Nagravision pay-TV set-top boxes. In the STU-III secure telephones used in the U.S. government, each user has a *crypto ignition key*, which is also packaged to look and feel like a physical key.

The single most widespread application that uses smartcards is the GSM mobile phone system, a digital standard used in some U.S. networks and in almost all countries outside the United States. The telephone handsets are commodity items, and are personalized for each user by means of a SIM, a smartcard which contains not just your personal phone book, call history and so on, but also a cryptographic key with which you authenticate yourself to the network.

The strategy of using a cheap smartcard to provide the authentication and other security functions of a more expensive consumer electronic device has a number of advantages. The expensive device can be manufactured in bulk, with each unit being exactly the same; while the smartcard, which provides subscriber-level control, can be replaced relatively quickly and cheaply in the event of a successful attack. This has led many pay-TV operators to adopt smartcards. The satellite TV dish and decoder become commodity consumer durables, while each subscriber gets a personalized smartcard containing the key material needed to decrypt the channels to which they have subscribed.

Chipcards are also used in a range of other applications, from hotel keys to public payphones—though in such applications it's common for the card to contain no microprocessor but just some EEPROM memory to store a counter or certificate, and some logic to perform a simple authentication protocol.

Devices such as prepayment electricity meters are typically built around a microcontroller that performs the same kind of functions as a smartcard but has less sophisticated protection. Typically, this consists of setting a single "memory protection" bit that prevents the EEPROM contents being read out easily by an attacker. There have been many design defects in particular products; for example, a computer authentication token called iKey had a master password that was hashed using MD5 and stored on an EEPROM external to the processor, enabling a user to overwrite it with the hash of a known password and assume complete control of the device [459].

Many other low-cost security products are based on some kind of microcontroller (or dedicated logic that performs an authentication protocol of some kind). An increasing number are *contactless*, and function as *radio frequency identifiers* that provide theft protection or just "smart labels" for a wide range of products. As for more systemic vulnerabilities, the attacks on smartcards also tend to work on microcontroller-based devices, so I won't treat them separately from this point on. For more details of attacks specific to microcontrollers, see [43].

14.6.1 Architecture

The typical smartcard consists of a single die of up to 25 square millimeters of silicon, containing an 8-bit microprocessor (such as an 8051 or 6805), although some of the newer devices are starting to appear with a 32-bit processor such as the ARM. It also has serial I/O circuitry and a hierarchy of three classes of memory: ROM to hold the program and immutable data; EEPROM to hold customer-specific data, such as the registered user's name and account number as well as crypto keys, value counters and the like; and RAM registers to hold transient data during computation.

The memory is very limited by the standards of normal computers. A typical card on sale in 2000 might have 16 Kbytes of ROM, 16 Kbytes of EEPROM and 256 bytes of RAM. The bus is not available outside the device; the only connections supplied are for power, reset, a clock, and a serial port. The physical, electrical, and low-level logical connections, together with a file-system-like access protocol, are specified in ISO 7816.

14.6.2 Security Evolution

When I first heard a sales pitch from a smartcard vendor—in 1986, when I was working as a banker—I asked how come the device was secure. I was assured that because the machinery needed to make the card cost $20 million, just as for making banknotes, the system must be secure. I didn't believe this, but didn't then have the time or the tools to prove the claim wrong. I later learned from industry executives that none of their customers were prepared to pay for serious security until about 1995, so until then they relied on the small size of the devices, the obscurity of their design, and the relative unavailability of chip-testing tools.

The application that changed all this was satellite TV. Operators broadcast their signals over a large footprint—such as all of Europe—and gave subscribers smartcards that would compute the keys needed to decipher the channels they'd paid for. Since the operators had usually only purchased the rights to the movies for one or two countries, they couldn't sell the subscriber cards elsewhere. This created a black market in pay-TV cards, into which forged cards could be sold. Another major factor was that *Star Trek*, which people in Europe had been picking up from U.K. satellite broadcasts for years, was suddenly encrypted in 1993. This motivated a lot of keen young computer science and engineering students to look for vulnerabilities.

Since then, major financial frauds have been carried out with cloned cards. The first to be reported involved a smartcard used to give Portuguese farmers rebates on fuel. The villain conspired with petrol stations that registered other fuel sales to the bogus cards in return for a share of the proceeds. The fraud, which took place in February–March 1995, is reported to have netted about thirty million dollars [557].

How to Hack a Smartcard (1)

The earliest hacks targeted the protocols in which the cards were used. For example, some early pay-TV systems gave each customer a card with access to all channels, then sent messages over the air to cancel those channels to which the customer hadn't subscribed after an introductory period. This allowed an attack whereby a device was inserted between the smartcard and the decoder to intercept and discard any messages addressed to the card. Subscribers could then cancel their subscription without the vendor being able to cancel their service.

The same kind of attack was launched on the German phone card system. A hacker called Urmel tipped off Deutsche Telekom that it was possible to make phone cards that gave unlimited free calls. He had discovered this by putting a laptop between a card and a phone to analyze the traffic. Telekom's experts refused to believe him, so he exploited his knowledge by selling handmade chipcards in brothels and in hostels for asylum seekers [726]. Such low-cost attacks were particularly distressing to the phone companies, as the main reason for moving to smartcards was to cut the cost of having to validate cheaper tokens online [78]. I'll discuss these protocol failures further in the chapter on copyright enforcement systems. There has also been a fairly full range of standard computer attacks, such as stack overwriting by sending too long a string of parameters. In the following, I concentrate on the attacks that are peculiar to smartcards.

How to Hack a Smartcard (2)

Smartcards use an external power supply, and store security state such as crypto keys and value counters in EEPROM, so an attacker could freeze the EEPROM contents by removing the programming voltage, V_{PP}. Early smartcards received V_{PP} on a dedicated connection from the host interface. This led to very simple attacks: by covering the V_{PP} contact with sticky tape, cardholders could prevent cancellation signals from affecting their card. The same trick could be used with some payphone chipcards; a card with tape over the appropriate contact had "infinite units."

The fix was to generate V_{PP} internally from the supply voltage V_{CC} using a voltage multiplier circuit. However, this isn't entirely foolproof as this circuit can be destroyed by an attacker. So a prudent programmer, having (for example) decremented the retry counter after a user enters an incorrect PIN, will read it back and check it. She will also check that memory writing actually works each time the card is reset, as otherwise the bad guy who has shot away the voltage multiplier can just repeatedly reset the card and try every possible PIN, one after another.

How to Hack a Smartcard (3)

Another early attack was to slow down the card's execution, or even single-step it through a transaction by repeatedly resetting it and clocking it n times, then $n + 1$ times, and so on. In one card, it was possible to read out RAM contents with a suitable transaction after reset, as working memory wasn't zeroized. With many cards, it was possible to read the voltages on the chip surface using an electron microscope. (The low-cost scanning electron microscopes generally available in universities can't do voltage contrast microscopy at more than a few tens of kilohertz, hence the need to slow down the execution.)

Now many smartcard processors have a circuit to detect low clock frequency, which will either freeze or reset the card. But, as with burglar alarms, there is a trade-off between the false alarm rate and the missed alarm rate. This leads to many of the alarm features provided by smartcard chip makers simply not being used by the OEMs or application developers. For example, with cheap card readers, there can be wild fluctuations in clock frequency when a card is powered up, causing so many false alarms that some developers do not use the feature. Clearly, low clock frequency detectors need careful design.

How to Hack a Smartcard (4)

Once pay-TV operators had fixed most of the simple attacks, pirates turned to attacks using physical probing (see Figure 14.4). Most smartcards have no protection against physical tampering beyond the microscopic scale of the circuit, a thin glass *passivation layer* on the surface of the chip, and potting, which is typically some kind of epoxy. Techniques for depackaging chips are well known, and discussed in detail in standard works on semiconductor testing, such as [80]. In most cases, a few milliliters of fuming nitric acid are all that's required to dissolve the epoxy; the passivation layer is then removed where required for probing.

Probing stations consist of microscopes with micromanipulators attached for landing fine probes on the surface of the chip. They are widely used in the semiconductor manufacturing industry for manual testing of production-line samples, and can be obtained second-hand for under $10,000. They may have specialized accessories, such as a laser to shoot holes in the chip's passivation layer (see Figure 14.5).

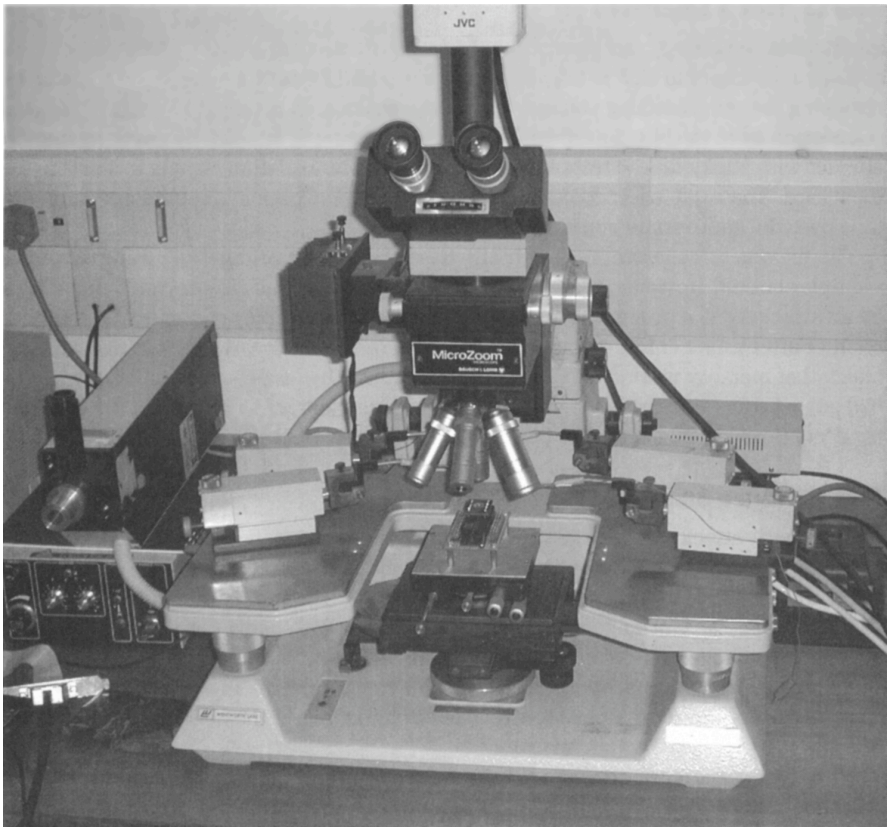

Figure 14.4 Low-cost probing station.

The usual target of a probing attack is the processor's bus. If the bus traffic can be recorded, this gives a trace of the program's operation with both code and data. If the attacker is lucky, the card designer will have computed a checksum on memory immediately after reset (a recommended defense industry practice), and this operation will immediately give him a complete listing of the card memory contents. So the attacker will identify the bus, and expose the bus lines for probing.

The first defense used by the pay-TV card industry against attacks of this kind was to endow each card with multiple keys and/or algorithms, and arrange things so that only those in current use would appear on the processor bus. Whenever pirate cards appeared on the market, a command would be issued over the air to cause the legitimate card population to activate new keys or algorithms from a previously unused area of memory. In this way, the pirates' customers would suffer a loss of service until the probing attack could be repeated and either new pirate cards, or updates to the existing ones, could somehow be distributed.

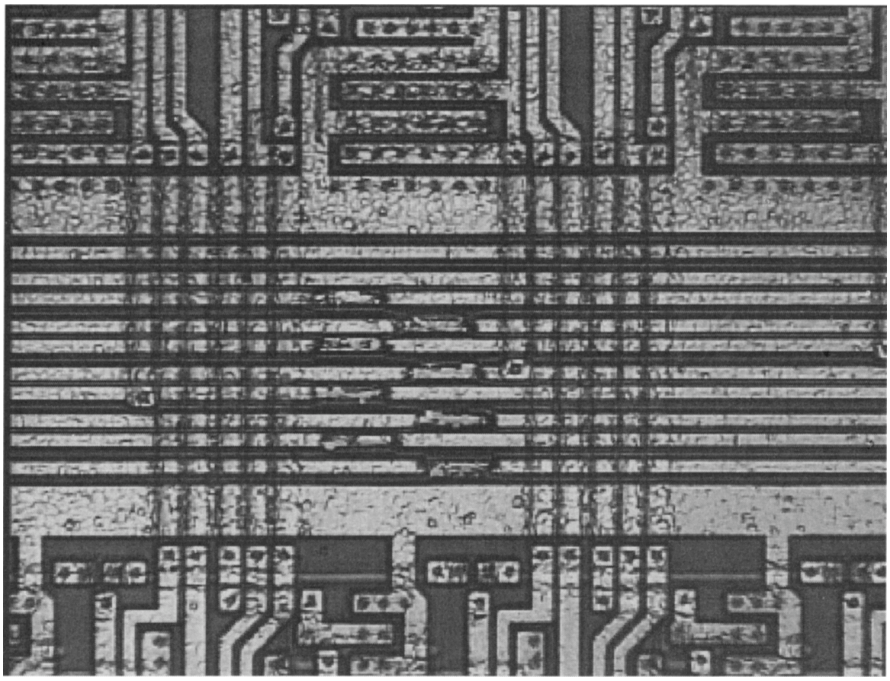

Figure 14.5 The data bus of an ST16 smartcard prepared for probing by excavating eight trenches through the passivation layer with laser shots (photo courtesy Oliver Kömmerling).

How to Hack a Smartcard (5)

The defeat for this strategy was Oliver Kömmerling's *memory linearization attack*, whereby the analyst damages the chip's instruction decoder in such a way that instructions which change the program address other than by incrementing it—such as jumps and calls—become inoperable [470]. One way to do this is to drop a grounded microprobe needle on the control line to the instruction latch, so that whatever instruction happens to be there on power-up is executed repeatedly. The memory contents can now be read off the bus. In fact, once some of the device's ROM and EEPROM are understood, the attacker can skip over unwanted instructions and cause the device to execute only instructions of his choice. So with a single probing needle, he can get the card to execute arbitrary code, and in theory could get it to output its secret key material on the serial port. But probing the memory contents off the bus is usually more convenient.

In practice, there are often several places in the instruction decoder where a grounded needle will have the effect of preventing programmed changes in the control flow. So even if the processor isn't fully understood, memory linearization can often be achieved by trial and error. Some of the more modern processors have traps that prevent memory linearization, such as hardware access control matrices which prevent particular areas of memory being read unless some specific sequence of commands is presented. But such circuits can often be defeated by shooting away carefully chosen gates using a laser or an ion beam.

Some cards could be attacked through their test circuitry. A typical smartcard chip has a self-test routine in ROM that is executed in the factory and allows-all the memory contents to be read and verified. After this has been done, a polysilicon fuse is blown in the chip to stop an attacker using the same facility. All that the attacker had to do was to find the fuse and repair it—which could involve as little as bridging it with two probing needles [130]. Then, in some cases, the entire memory contents could be read out over the serial port. A more careful design might put the test circuitry on the part of the silicon that is sawn away when the wafer is diced into individual chips.

How to Hack a Smartcard (6)

The next thing the pay-TV card industry tried was to incorporate hardware cryptographic processors, to force attackers to reconstruct hardware circuits rather than simply clone software, and to force them to use more expensive processors in their pirate cards. In the first such implementation, the crypto processor was a separate chip packaged into the card. This design had an interesting protocol failure: it would always work out the key needed to decrypt the current video stream, then pass it to the CPU, which would decide whether or not to release it to the outside world. Hackers broke this system by developing a way to tap into the wiring between the two chips.

More modern implementations have the crypto hardware built into the CPU itself. Where this consists of just a few thousand gates, it is feasible for an attacker to reconstruct the circuit manually from micrographs of the chip. But with larger gate counts and deep submicron processes, a successful attack may require automatic layout reconstruction: successively etching away the layers of the chip, taking electron micrographs, and using image processing software to reconstruct a 3-D map of the chip, or at least identify its component cells [121]. However, assembling all the equipment, writing the software, and integrating the systems involves significant effort and expense.

A much simpler, and common, attack is for pirates to ask one of the dozen or so existing commercial reverse-engineering labs to reconstruct the relevant area of the chip. Such labs get much of their business from analyzing commercial integrated circuits on behalf of the chip maker's competitors, looking for possible patent infringements. They are used to operating in conditions of some secrecy, and it doesn't seem to be too difficult for a pirate to sneak in a sample that is targeted for piracy rather than litigation.

How to Hack a Smartcard (7)

The next defense that the card industry thought up was to furnish the chip with protective surface mesh, implemented in a top metal layer as a serpentine pattern of ground, power and sensor lines. The idea was that any break or short in the pattern would be sensed as soon as the chip was powered up, thereby triggering a self-destruct mechanism.

I mentioned such meshes in connection with the Dallas processors; after the usual initial crop of implementation blunders, they have proved to be an effective way of pushing up the cost of an attack. The appropriate tool to defeat them is the *Focused Ion Beam Workstation* (FIB). This is a device similar to a scanning electron microscope, but it uses a beam of ions instead of electrons. By varying the beam current, it is possible to use it as a microscope or as a milling machine. By introducing a suitable gas, which is broken down by the ion beam, it is possible to lay down either conductors or insulators with a precision of a few tens of nanometers.

FIBs are such extremely useful devices in all sorts of applications—from semiconductor testing through metallurgy and forensics to nanotechnology—that they are rapidly becoming widely available, and their prices are tumbling. Many universities and industrial labs now have one. FIB time can also be rented from a number of agencies for a few hundred dollars an hour.

Given a FIB, it is straightforward to attack a sensor mesh that is not powered up. One simply drills a hole through the mesh to the metal line that carries the desired signal, fills it up with insulator, drills another hole through the center of the insulator, fills it with metal, and plates a contact on top—typically, a platinum L or X a few microns wide, which is easy to contact with a needle from the probing station (see Figure 14.6).

Defeating a sensor mesh that is continually powered up is much harder, but the necessary tools are starting to emerge from the labs of the chip-testing industry. For example, there are techniques to mill through the back side of a chip with a suitably equipped FIB, and make contact directly to the electronics without disturbing the sensor mesh at all.

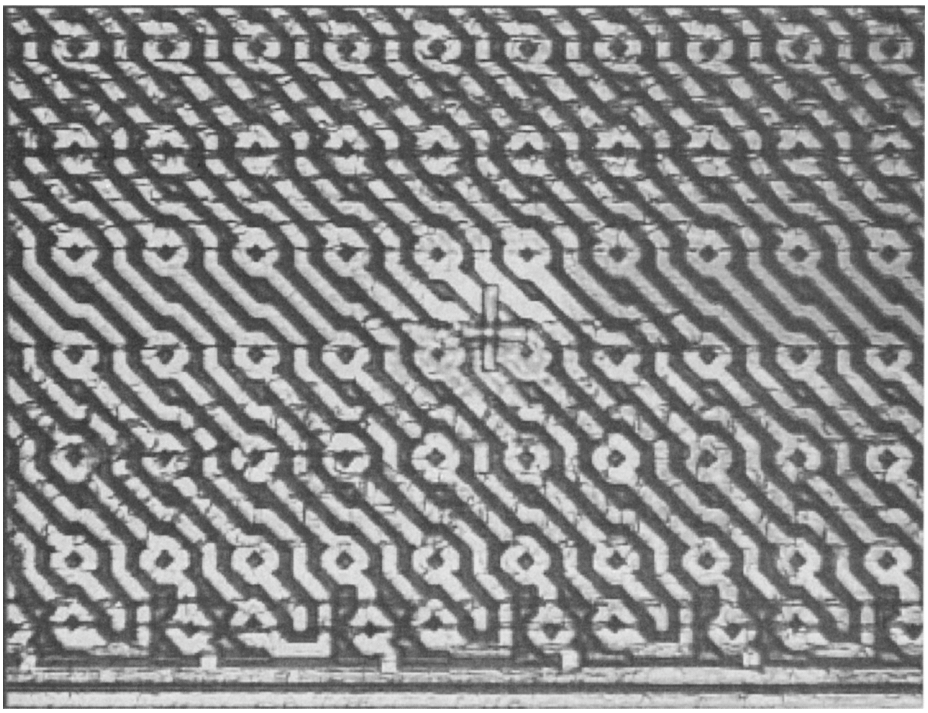

Figure 14.6 The protective mesh of an ST16 smartcard with a FIB cross for probing the bus line visible underneath (photo courtesy Oliver Kömmerling).

Many other defensive techniques can force the attacker to do more work. Some chips are said to be packaged in much thicker glass than in a normal passivation layer. The idea is that the obvious ways of removing this (such as applying hydrofluoric acid) are likely to damage the chip. However, optoelectronic techniques developed in the past few years enable an attacker to read out a voltage directly using a laser [11]. Other chips have protective coatings of substances such as silicon carbide or boron nitride. (Chips with protective coatings are on display at the NSA Museum at Fort Meade,

Maryland). Such coatings can force the FIB operator to go slowly, rather than damage the chip through a build-up of electrical charge. However, protective layers in smartcard chip packaging are, like much else in the security industry, often a matter of marketing rather than engineering. The one chip that our team has dismantled recently and whose vendors claimed to have such a layer, turned out to have no special protection at all.

14.6.3 The State of the Art

At the time of writing, I know of no technology, or combination of technologies, that can make a smartcard resistant to penetration by a skilled and determined attacker. Some industry experts even believe that absolute protection in chip-sized packages will remain infeasible, because it's not economic to fabricate devices that you can't test.

Despite this, smartcards are certainly a lot harder to copy than magnetic stripe cards, and there is room for them to be made harder still. The latest cards have up to three layers of defensive mesh; registers that use dynamic logic, making it impossible to just shoot away a low clock frequency detector, then single-step the chip; circuits that insert dummy instructions from time to time so that if you probe the bus lines one after another, you may have to do a lot of work to align the traces you get; 32-bit processors, which make trace alignment even harder; proprietary instruction sets; and a whole host of other tricks. But as industry insiders say, 'the man with the ion beam will always get in'.

So what sort of strategies are available to you if you are designing a system that depends on smartcards?

14.6.3.1 Defense in Depth

The first, used by pay-TV companies, is *defense in depth*. Smartcards may combine a whole menagerie of the tricks described above, and even obscure proprietary encryption algorithms. Normally, using home-brewed encryption schemes is a bad thing: Kerckhoffs' principle almost always wins in the end, and a bad scheme, once published, can be fatal. Defense in depth of pay-TV provides an interesting exception. The goal is to minimize, insofar as possible, the likelihood of a shortcut probing attack, and to force the attacker to go to the trouble of reverse engineering substantially the whole system.

It's prudent to assume that even an amateur will be able to drop a probe on to a number of signal lines in the device. If it is performing a known cryptographic protocol with well-understood algorithms, then unless there's an effective mechanism to introduce lots of dummy instructions, a trace from a single bus line is likely to give away the key [370]. Using a proprietary (and complicated) encryption algorithm can force the attacker to do a more complete analysis and delay him for a few weeks or months. This can make a huge difference to the economics of piracy in an industry such as pay-TV where cards are replaced about once a year. (Of course it's even more essential with a proprietary design to have it evaluated thoroughly by competent experts—and for the experts to analyze not just the abstract cryptographic strength of the algorithm, but how easily it can be reconstructed from observable signals.)

Technical measures on their own are not enough, though. Over the last few years of the twentieth century, the pay-TV industry managed to reduce piracy losses from over 5% of revenue to an almost negligible proportion. More complex smartcards played a

role, but much of the improvement came from legal action against pirates, and from making technical and legal measures work together efficiently. I'll discuss this further in Chapter 20, when we explore the world of copyright.

14.6.3.2 Tamper Resistance versus Tamper Evidence

It can often be very useful to draw a careful distinction between devices that are tamper-resistant and those that are merely tamper-evident. Even if the former can't yet be produced for mass markets, it is more or less within our grasp to make smartcards against which the only attacks are invasive, such as probing, and therefore leave physical evidence behind. (This is still harder than it looks—in the next chapter we'll discuss noninvasive attacks.)

For example, in a banking application where smartcards are used to manufacture and verify electronic checks, the bank might have a rule that disputes will be considered only if customers can produce their smartcard undamaged. This is not quite as simple as it seems, as smartcards can always be damaged by accident. Maybe 1% of smartcards issued to the public will be destroyed every year by material failures or static electricity; consumer laws in many countries may prevent banks from disregarding claims when that happens. Once again, the legal and engineering aspects of the problem interact. Nonetheless, cards that are tamper-evident (as well as being fairly difficult to probe) can be a useful part of a risk management strategy.

14.6.3.3 Stop Loss

Whether one goes for the defense-in-depth approach or the tamper-evident approach will depend on the extent to which one can limit the losses that result from a single card being successfully probed.

In early pay-TV systems, the system architecture forced all customer cards to contain the same master secret. Once this secret became known, pirate cards could be manufactured at will, and the card base had to be replaced. The pay-TV systems currently being deployed for digital broadcasting use crypto protocols in which cards have different keys, so that cloned cards can be revoked. I'll describe these protocols in Section 20.2.4.5.

In other systems, such as the banking card application described in Section 2.7.1, there are limits on the amount that can be spent using a stolen or forged card, set by a system of merchant floor limits, random online authorizations, lists of hot cards and so on. Here, a tamper-evident card may be a perfectly adequate solution. Even a card that's relatively easy to forge may be viable, as it's still harder to forge than the magnetic stripe card it replaces.

14.7 What Goes Wrong

There are failure modes of systems involving tamper-resistant processors that are more or less independent of whether the device is low or high end. Many failures occurred because the device was exposed to more capable attackers than its designers anticipated: it just never seems to have occurred to the designers of early chip cards that bad people might have access to semiconductor test equipment. Many more occur because people protect the wrong things, or protect the right things in the wrong way; a survey of flaws found by a commercial evaluation laboratory showed that most of them were at the interfaces between physical, logical, and organizational measures [131].

14.7.1 Protecting the Wrong Things: Architectural Errors

A good example of misuse of the technology is the drive to have smartcards adopted as the preferred device for digital signatures. Some government initiatives give enhanced legal validity to signatures made using an approved smartcard. While this may be a Full Employment Act for the card industry, it makes little sense technically.

None of the devices described in the preceding sections has a really trustworthy user interface. Some of the bank security modules have a physical lock (or two) on the front to ensure that only the person with the metal key can perform certain privileged transactions. But whether you use a $2,000 4758 or a $2 smartcard to do digital signatures, you still trust the PC that drives them. If it shows you a text reading "Please pay amazon.com $37.99 for a copy of Anderson's *Security Engineering*," while the message it actually sends for signature is "Please remortgage my house at 13 Acacia Avenue and pay the proceeds to Mafia Real Estate Inc.," then the tamper resistance has not bought you much.

It may even make your situation worse, as you will have a harder time repudiating the transaction. Information policy experts have pointed out that the proposed approach to digital signatures is likely to undermine the very consumer protection laws that give people confidence when conducting business electronically over the Net [124]. What customers really need is a secure PC—or at least a firewall to shield their PC from the worst of the external threats, such as malicious code. That is a separate engineering problem, and has little to do with hardware security. In fact, researchers are coming to realize that a palmtop computer may be a much better platform for digital signature applications; whatever its vulnerability to probing, customers can at least see what they're signing and protect the device using common sense [69].

An example of more appropriate use of hardware protection technology comes from the prepayment electricity metering system, discussed in Chapter 11. There, the function of tamper resistance was to limit the loss if one of the vending machines that sold meter tokens was stolen. By keeping the keys needed to encrypt the tokens in a secure processor, which also contained a value counter, it was possible to enforce a credit limit on each vendor. Had someone managed to drill into the device, he would have been able to defeat the value counter, extract the crypto keys for the individual meters, and thus sell unlimited numbers of tokens to the meters in the vending area. But this would not have destroyed the whole metering system, just forced the rekeying of a few thousand meters.

14.7.2 Protecting the Wrong Things: Security-by-Obscurity and Evaluation Errors

Many of the smartcard systems that have been broken, in ways that resulted in real frauds, appear to have become vulnerable because their operators did not fully understand the technology and its limitations. This is hardly surprising; until recently, no published information was readily available on how smartcards could be attacked. The industry also sought to keep all serious technical information about its products secret. To this day, one has to sign a nondisclosure agreement to get proper software development tools for smartcards. (There are Java cards, Basic cards, and so on, but these use interpreted languages to shield the developer from the hardware and don't support users who want to run their own machine code on the device).

In fact, the security target used for evaluating smartcards under the Common Criteria focuses on maintaining obscurity of the design. Chip masks must be secret, staff must be vetted, developers must sign nondisclosure agreements—there are many requirements that push up industry's costs. Obscurity is also a common requirement for export approval, and there remains a suspicion that it covers up deliberately inserted vulnerabilities. For example, a card my colleagues tested would always produce the same value when instructed to generate a private/public keypair, and output the public part.

Obscurity certainly does little for the customer in most smartcard applications. Almost none of the actual attacks on fielded smartcard systems used inside information. Most of them started out with a probing attack on a card bought at retail.

Better protection targets were published by VISA, which specify extensive penetration testing [777]. However, as no existing products can pass such a test, the industry took the route of protecting what it could rather than what it should. I'll return to this subject to discuss the underlying economics and politics in Section 23.3.3.1.

14.7.3 Protecting Things Wrongly: Protocol Failure

As elsewhere in security engineering, one of the most pervasive kinds of failure at the technical level is the use of inappropriate protocols. A device such as the 4758 comes with a transaction set of several hundred "verbs," or combinations of cryptographic operations that can be carried out on data passed to the device. Further verbs can be defined by the application developer. How can one be sure that some combination of these verbs won't enable a user to do something that breaks the security policy?

From about 1981 until 1991, there was a protocol attack that worked against many of the security modules used by banks to manage ATM networks. As the working life of a security module is about seven years, the defective devices should all have retired by the time this book appears (but they completely destroy the claim made by many banks in the course of phantom withdrawal litigation in the early 1990s that "nothing could possibly go wrong").

The security modules provided by VISA and VISA-compatible vendors such as Racal had a transaction to generate a key component and print out its clear value on an attached security printer.

They also returned its value to the calling program, encrypted under a master key *KM* which was kept in the tamper-resistant hardware:

$$\text{VSM} \rightarrow \text{printer: } KMT_i$$
$$\text{VSM} \rightarrow \text{host: } \{KMT_i\}_{KM}$$

and another that combined two of the components to produce a terminal key:

$$\text{Host} \rightarrow \text{VSM: } \{KMT_1\}_{KM}, \{KMT_2\}_{KM}$$
$$\text{VSM} \rightarrow \text{host: } \{KMT_1 \oplus KMT_2\}_{KM}$$

The idea was that, to generate a terminal key for the first time, you'd use the first of these transactions twice, followed by the second. Then you'd have $KMT = KMT_1 \oplus KMT_2$. However, there is nothing to stop the programmer taking any old encrypted key and supplying it twice in the second transaction, resulting in a known terminal key (the key of all zeroes, as the key is exclusive-or'ed with itself):

$$\text{Host} \rightarrow \text{VSM: } \{KMT_1\}_{KM}, \{KMT_1\}_{KM}$$
$$\text{VSM} \rightarrow \text{host: } \{KMT_1 \oplus KMT_1\}_{KM}$$

The module also has a transaction that will take two keys, encrypted under the master key and return one encrypted with the other

$$\text{Host} \rightarrow \text{VSM: } \{KMT_1\}_{KM}, \{KMT_2\}_{KM}$$
$$\text{VSM} \rightarrow \text{host: } \{KMT_1\}_{KMT2}$$

(This is intended to allow the terminal master key in a cash machine to be replaced, or a PIN key to be sent to a cash machine encrypted under the terminal master key, to support offline PIN verification.)

The attack is now simple, and devastating. Having a zero key, encrypted under *KM*, we can translate the PIN key (and everything else of interest) from being encrypted under *KM* to being encrypted under the zero key. Thus, the physical protection that was promised to customers was a phantasm: a programmer could extract any key of interest with only two unprivileged instructions.

This is interesting from the scientific point of view, because the security policy enforced by the VSM is a kind of kludge between a multilevel policy ("PINs are Secret and must not leak to any process with a lower classification") and a shared control policy ("no single member of bank staff should be able to work out a customer PIN"). It's also interesting from the public policy viewpoint, as it was known to the equipment vendors at the time of the Munden case described in Section 9.4.3. But the vendors didn't own up to it, despite the fact that its existence would have directly undermined the prosecution testimony in a highly publicized miscarriage-of-justice case. This should be remembered whenever one of the parties in a court case relies on vendor assurances about a system's capabilities.

The fix adopted was to remove the offending instruction. This means that dual control key management now involves a trusted process at the host, which will have access to key material. (This has always been the case with the ATM support application, CCA, supplied for the 4758.) A better fix would have been to compute terminal keys

using a hash function, such as $KMT = SHA1(KMT_1,KMT_2)$, but this would not have been backward-compatible. With hindsight, the choice of a combining function with arithmetic properties meant that all the protocols subsequently developed on this foundation should have been checked for ways in which these properties could misbehave. In other words, the choice of combining function raised the complexity of transaction set verification.

This brings us to the attack found on the 4758 by Mike Bond. This enables any key to be extracted from the device with only a modest keysearch. The vulnerability is that the two-key, triple-DES encryption used internally by the 4758 can have its key pairs cut and spliced. Given a known pair of keys, *KA* and *KB*, and a target pair of keys *KC* and *KD*, one can compare the results of encryption under the spliced keys (*KC*,*KB*) and (*KA*,*KD*) with a brute-force search of all possibilities, thus breaking the target keys one component at a time. This reduces the cost of an attack from the 2^{112} of a standard triple-DES keysearch to the more tractable 2^{56} of single-DES. There is also a time-memory tradeoff available; for example, with 2^{16} trial keys it is possible to break the 4758 with an effort of about 2^{40} test encryptions. For full details, see [125].

Let's step back for a minute and consider the implications. IBM spent over a decade evolving a highly strategic product, that is used by many banks to protect zillions of dollars' worth of transactions. The US government certified it as the most secure crypto processor available to civilian purchasers, and kept it export-controlled. IBM further protected it by refusing to sell us a sample. Yet a typical Cambridge graduate student broke it within six weeks by studying the manuals available from IBM's Web site.

Verifying the correctness of the transaction set of a cryptographic processor is a hard, and as yet unsolved, problem. Verifying an individual protocol is difficult enough, and the research community spent much the 1990s learning how to do it. Yet a protocol might consist of perhaps two to five messages, while a cryptoprocessor might have from dozens to hundreds of verbs. Many protocols fail because their goals are not made sufficiently explicit; yet cryptoprocessors are sold as general-purpose machines, and may be used to enforce a very wide range of security policies. We don't yet really know how to formalize security policies, let alone track them back to crypto primitives. Checking that there isn't some obscure sequence of transactions that breaks your security policy is hard enough; when your policy isn't even precisely stated, it looks impossible.

14.7.4 Function Creep

I've given numerous examples of how function creep, and changes in environmental conditions in general, have broken many secure systems by undermining their design assumptions. The flexibility of some modern cryptographic processors makes this a particular problem.

Function creep can also interact with physical tamper-resistance directly, and is particularly pernicious in smartcard applications. It is easy to move subtly from a system in which smartcards act much like magnetic strip cards and perform transactions on underlying bank accounts that are cleared every night, to a system in which they act like wallets and can transfer value to each other. In the former, cards can have different keys shared only with the bank, and so the compromise of a single card need mean no more than the cloning of a credit card does in the magnetic strip world. In the latter, each card has a key that enables it to transfer money to any other card, and the con-

straints of centralized accounting are relaxed. A relatively low-risk environment suddenly becomes a relatively high-risk one.

Another way a low-risk environment can become a high-risk one is when multiple applications are put on the same card. If a device that was previously just a health insurance card or a welfare claimants' card suddenly starts to double as a national identity card, then it may attract a much higher grade of attacker. If large numbers of different organizations can run their own applications on the card—which is the stated purpose of Java cards—then the *chosen protocol attack* described in Chapter 2 becomes a particularly dangerous threat. A bad man may design an application specifically to attack yours.

14.8 What Should Be Protected?

With many technologies—including the steam engine, telephone, and computer—the original proposed use of the device was not that which eventually took off in the market. (Consider the size of the market today for pumping water out of coal mines; reading text to telegraph operators rather than sending it through a pneumatic tube; and working out artillery range tables.)

The currently fashionable sales pitch for smartcards is that they will be the *advanced electronic signature devices* envisaged in EU electronic commerce regulations—that is, devices that people will use to sign legal documents and that will be sufficiently dependable that the existence of such a signature can be taken as proof that the owner of the device signed it. Quite apart from the obvious legal objections (that it shifts the burden of proof from the party relying on the signature to the device owner, and that devices can always be stolen), there is, as I mentioned earlier, the technical problem that the user doesn't know what the smartcard is signing; and if the PC software, that supplies the matter to be signed, is guaranteed to be bug-free and free from viruses, then what value does the smartcard add?

The industry has been trying for much of the 1990s to sell the idea of a multifunction card, which would replace many of the plastic cards and metal keys that the average person carries with them. The application that makes this finally happen may be putting bank transaction processing into mobile phones. As mobile phones have slots for precisely one smartcard, a bank would have to rent space on the card from the phone network operator. We shall see.

So what value can tamper-resistant devices actually add?

First, they can control information processing by linking it to a single physical token. A pay-TV subscriber card can be bought and sold in a gray market, but as long as it isn't copied, the station operator isn't too concerned. Another example comes from a Dallas product used in quality control in the food industry: it is physically sealed into a food shipment to provide a reliable log of temperature history. Yet another is the use of crypto to enforce evaluation standards in government networks: if you only get key material once your system has been inspected and accredited, then it's inconvenient to connect an unlicensed system of any size to the classified government network.

Second, tamper-resistant devices can give assurance that data are destroyed at a definite and verifiable time. The anti-trust case against Microsoft has highlighted the damage that can be done by the seizure under subpoena of email archives; many corporations would like to enforce a policy that every email be destroyed after a fixed time,

unless either the sender or the recipient takes positive action to preserve it. At my university, for example, we destroy exam scripts and examiners' working notes after four months. If we held on to them for too long, we would have to give the students access under data protection law, but if we destroyed them too soon, we could prejudice an appeal. Once everything is electronic, implementing such a policy will be complicated by all the system backups we keep. A solution is to encrypt archives with keys kept in a device that is programmed to erase them at the right time.

Third, these devices can reduce the need to trust human operators. As I remarked, their main purpose in some government systems was "reducing the street value of key material to zero". A crypto ignition key for a STU-III should allow a thief only to masquerade as the rightful owner, and only if he has access to an actual STU-III telephone, and only as long as neither the key nor the phone have been reported stolen. The same general considerations applied in ATM networks: no bank wanted to make its own customers' security depend on the trustworthiness of the staff of another bank.

Fourth, tamper-resistant devices can be used to control value counters, as with the prepayment electricity discussed in Section 14.7.1. These typically use devices such as the DS5002 or the iButton to hold both the vend keys for local meters and a credit counter. Even if the device is stolen, the total value of electricity tokens it can vend is limited.

This seems to be a special case of a more general application, in which some part of a central server's processing is delegated to a device in the field. But the most compelling examples I can think of concern value. Note that a guarded corporate data-processing center is also a tamper-resistant processor; applications of this type can often be spotted by the fact that they could also be implemented centrally if a completely reliable network existed. For example, if all electricity meters and vend stations were online, then prepayment metering could be done using straightforward authenticated messaging. Note, too, that delegation also occurs between corporate data processing centers, as when banks use hot-card lists to authenticate card transactions on other banks. Here, tamper-resistant devices may be used to provide extra assurance (though often logging mechanisms are sufficient where the continued solvency of the principals can be relied on).

This is an incomplete list. But what these applications have in common is that a security property can be provided independently of the trustworthiness of the surrounding computer environment. In other words, be careful when using tamper-resistant devices to try to offset the lack of a trustworthy user interface. This doesn't mean that no value at all can be added where the interface is problematic. For example, the tamper-resistant crypto modules used in ATM networks cannot prevent small-scale theft using bogus ATMs; but they can prevent large-scale PIN compromise if used properly. In general, tamper-resistant devices are often a useful component, but only very rarely provide a fully engineered solution.

Finally, it is worth noting that tamper-resistance provides little protection against legal attack. If you rely on it to keep algorithms proprietary, your competitors can bring a patent infringement suit (however frivolous) simply to force disclosure of your design. This actually happens!

14.9 Summary

Tamper-resistant devices and systems have a long history, and predate the development of electronic computing. Computers can be protected against physical tampering in a number of ways, such as by keeping them locked up in a guarded room. There are also several cheaper and more portable options.

This chapter looked at a number of them, from devices costing thousands of dollars that are certified by the U.S. government to resist all currently known attacks, down to smartcards that can be penetrated by an attacker with a few thousand dollars' worth of equipment with a few weeks' work. I discussed a number of applications, and a number of failures. Very often, the failures are not the fault of the hardware barriers or alarms themselves, but a result of using the technology in an inappropriate way.

Research Problems

There are basically three strands of research in tamper-resistant processor design. The first concerns itself with making faster, better, cheaper processors: how can the protection offered by a high-end device be brought to products with midrange prices and sizes, and how can midrange protection can be brought to smartcards? The second concerns itself with pushing forward the state of the attack art. How can the latest chip-testing technologies be used to make faster, better, cheaper attacks?

The third strand concerns itself with the logical aspects of protection. Even assuming that you can put a perfectly impenetrable barrier around a processor—imagine, for example, a processor in orbit around Mars—how do you design the transaction set (and the surrounding application) so that it can do useful work, with a high level of assurance that some simple attack won't be found?

Further Reading

For the early history of crypto, including things like weighted code books and water-soluble inks, the source is, of course, Kahn [428]. The IBM and Dallas products mentioned have extensive documentation available online [397]; the U.S. FIPS documents are also online [576]. For an introduction to chip card technology, see [632]; and for the gory details of tampering attacks on chip cards, see [43, 44, 470]. Noninvasive attacks on security processors, such as power analysis, are discussed in the next chapter.