CHAPTER

16

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

The amount of careful, critical security thinking that has gone into a given security device, system or program is inversely proportional to the amount of high-technology it uses.

— Roger Johnston

16.1 Introduction

Low-cost tamper-resistant devices are becoming almost ubiquitous. Examples I've discussed so far include:

- smartcards used as SIMs in mobile phones and as bank cards in Europe;
- accessory control chips used in printer toner cartridges, mobile phone batteries and games-console memory modules;
- the TPM chips being shipped in PCs and Macs to support hard-disk encryption, DRM and software registration;
- security modules used to manage bank PINs, not just in bank server farms but in ATMs and point-of-sale terminals;

- security modules buried in vending machines that sell everything from railway tickets through postage stamps to the magic numbers that activate your electricity meter.

Many of the devices on the market are simply pathetic, like the banking terminals whose failures I described in section 10.6.1.1: those terminals could be trivially compromised in under a minute using simple tools, despite having been evaluated by VISA and also using the Common Criteria framework.

Yet some tamper-resistant processors are getting pretty good. For example, I know of one firm that spent half a million dollars trying, and failing, to reverse-engineer the protocol used by a games console vendor to stop competitors making memory modules compatible with its equipment[1]. But a few years ago this was not the case. Serious tamper resistance emerged out of an arms race between firms that wanted to lock down their products, and others who wanted to unlock them. Some of the attackers were respectable companies exercising their legal rights to reverse engineer for compatibility. Others were lawyers, reverse engineering products to prove patent infringements. There are half a dozen specialist firms that work for the lawyers, and the legal reverse engineers. There are academics who hack systems for glory, and to push forward the state of the art. There are bad guys like the pay-TV pirates who clone subscriber cards. And finally there are lots of grey areas. If you find a way to unlock a particular make of mobile phone, so that it can be used on any network, is that a crime? The answer is, it depends what country you're in.

There are now many products on the market that claim to be tamper-resistant, from cheap microcontrollers through smartcards to expensive cryptoprocessors. Some of them are good; many are indifferent; and some are downright awful. It is increasingly important for the security engineer to understand what tamper resistance is, and what it can and can't do. In this chapter I'm going to take you through the past fifteen years or so, as ever more clever attacks have been met with successively more sophisticated defenses.

It has long been important to make computers resist physical tampering, as an attacker who can get access can in principle change the software and get the machine to do what he wants. While computers were massive objects, this involved the techniques discussed in the previous few chapters — physical barriers, sensors and alarms. In some applications, a computer is still made into a massive object: an ATM is basically a PC in a safe with banknote dispensers and alarm sensors, while the sensor packages used to detect unlawful nuclear tests may be at the bottom of a borehole several hundred feet deep and backfilled with concrete.

Where tamper resistance is needed purely for integrity and availability, it can sometimes be implemented using replication instead of physical protection. A

[1]Eventually the memory module was cracked, but it took a custom lab with chip testing equipment and a seven figure budget.

service may be implemented on different servers in different sites that perform transactions simultaneously and vote on the result; and the threshold schemes discussed in section 13.4 can also provide confidentiality for key material. But tamper-resistant devices can 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.

16.2 History

The use of tamper resistance in cryptography goes back centuries [676]. Naval codebooks were weighted so they could be thrown overboard if capture was imminent; 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.

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 key material was sold to the other side by cipher staff — such as the notorious Walker family in the USA, who sold U.S. Navy key material to the Russians for over 20 years [587] — engineers paid more attention to the question of how to protect keys in transit too. 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 dispenses crypto keys in a controlled way. Nowadays this function is usually performed using a small security processor such as a smartcard; as with electricity meters, it may be packaged as a 'crypto ignition key'. Control protocols range from a limit on the number of times a key can be dispensed, to mechanisms using public key cryptography to ensure that keys are only loaded into authorized equipment. The control of key material also acquired broader purposes. In both the USA and the UK, 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 authentication and key agreement protocols. I already talked about many of the basic tools, such as key diversification, in the chapter on protocols in Part I, and I'll have more to say on protocols later in the chapter in API attacks. Here, I'm going to look first at the physical defenses against tampering.

16.3 High-End Physically Secure Processors

An example worth studying is the IBM 4758 (Figures 16.1 and 16.2). This is important for three reasons. First, it was the first commercially available processor to have been successfully evaluated to the highest level of tamper resistance (FIPS 140-1 level 4) [938] then set by the U.S. government. Second, there is an extensive public literature about it, including the history of its design evolution, its protection mechanisms, and the transaction set it supports [1195, 1328, 1330]. Third, as it was the first level-4-evaluated product, it was the highest profile target in the world of tamper resistance, and from 2000–2005 my students and I put some effort into attacking it.

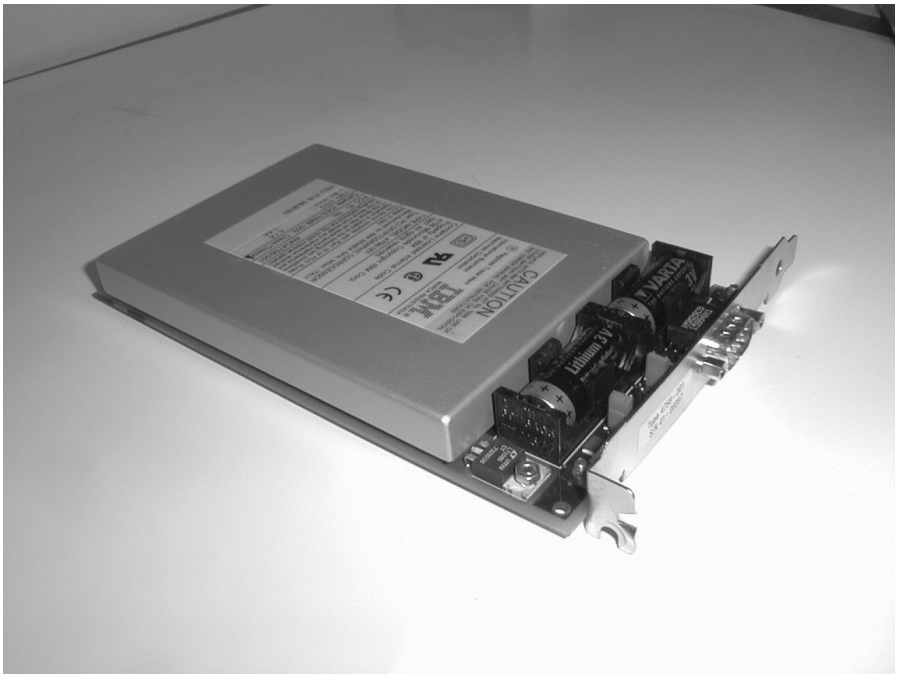

Figure 16.1: The IBM 4758 cryptoprocessor (courtesy of Steve Weingart)

Figure 16.2: The 4758 partially opened showing (from top left downward) the circuitry, aluminium electromagnetic shielding, tamper sensing mesh and potting material (courtesy of Frank Stajano)

The evolution that led to this product is briefly as follows. The spread of multi-user operating systems, 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. The reaction of the military computing community, which I described in Chapter 9, was the Anderson report and multilevel security. The reaction of the banking community was to focus on particularly sensitive data — and specifically on long-term cryptographic keys and the personal identification numbers (PINs) used by bank customers to identify themselves to cash machines. It was realized in the early 1980s that the level of protection available from commercial operating systems was likely to remain insufficient for these 'crown jewels'.

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. This was accomplished by wiring the power supply to the key memory through a number of lid switches. So whenever the maintenance crew came to replace batteries, they'd open the lid and destroy the keys. Once they'd finished, the device operators would then reload the key material. In this way, the device's owner could be happy that its keys were under the unique control of its own staff.

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 that were loaded into a special socket in the device to be read during initialization, or as strings of numbers which 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 key components, and make the device master keys the exclusive-or of all the components. The PROMs can then be kept in different safes under the control of different departments. (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 tedious and such procedures tend to degrade. 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. The managers with custodial responsibility will often give their 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.

Prudent cryptography designers try to minimize the number of times that a key reload will be necessary, whether because of maintenance or power failure. So modern security modules typically have batteries to back up the mains power supply (at least to the key memory). Thus, in practice, the custodians have to load the keys only when the device is first installed, and after occasional maintenance visits after that.

It has been debated whether frequent or infrequent key loading is best. If key loading is very infrequent, then 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 [33] for a case history of this). The modern trend is toward devices that generate master keys (or have them loaded) in a secure facility after manufacture but before distribution. But not all keys can be embedded in the processor at the factory. Some keys may be kept on smartcards and used to bootstrap key sharing and backup between processors; others may be generated after distribution, especially signature keys that for legal reasons should always be under the customer's unique control.

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 further 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 man.

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 circuits). The core is then 'potted' into 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 engineer into the bank computer room. (That at least was the theory; my own experience suggests that it's a bit much to ask a minimum-wage guard to ensure that a specialist in some exotic piece equipment repairs it using some tools but not others.) At least it should leave evidence of tampering after the fact. This is the level of protection needed for medium-level evaluations under the FIPS standard.

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 — and, to be realistic, that's what the maintenance engineer probably has, even if the guard is breathing down his neck — 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 [580]. 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 μABYSS system in the mid 1980s. This used loops of 40-gauge nichrome wire that were wound loosely around the device as it was embedded in epoxy, and then connected to a sensing circuit [1328]. 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 round 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 [69], 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 [566] and [568], while [1184] explains how to extract data from Flash memory in microcontrollers, even after it has been 'erased' several times. The NSA has published guidelines (the 'Forest Green Book') on preventing remanence attacks by precautions such as careful degaussing of media that are to be reused [378].

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 several seconds after power is removed. This extends to minutes at the temperatures of liquid nitrogen. 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 ionising radiation. (For the memory chips of the 1980s, this required a 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 some of them keeping data for several seconds even at room temperature. What's worse, remanence seems to have got longer as feature sizes have shrunk, and in unpredictable ways even within standard product lines. The upshot is that although your security module might pass a remanence test using a given make of SRAM chip, it might fail the same test if fitted with the same make of chip purchased a year later [1182]. This shows the dangers of relying on a property of some component to whose manufacturer the control of this property is unimportant.

Temperature sensors are also a real bugbear to security module vendors, as a device that self-destructs if frozen can't be sent reliably through normal distribution channels. (We've bought cryptoprocessors on eBay and found them dead on arrival.)

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', 'power analysis,' 'side-channel attacks' or 'emission security', 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 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. The design is described in more detail in [1195].

How to hack a cryptoprocessor (7)

I don't know how to attack the hardware of the 4758. My students and I found a number of novel software vulnerabilities, which I'll describe later in the chapter on API Attacks. But here are a few ideas for keen grad students who want to have a go at the hardware:

- The straightforward approach would be to devise some way to erode the protective potting, detect mesh lines, and connect shunts round them. A magnetic force microscope might be worth a try.
- 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 straightforward with standard mechanical drills, 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 its hardware, the attacker might attempt to use destroy the tamper responding circuitry before it has time to react. One possibility is to use an industrial X-ray machine; another would be to use shaped explosive charges to send plasma jets of the kind discussed in section 13.5 into the device.

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

So by far the attacks on 4758-based systems involve the exploitation of logical rather than physical flaws. The device's operating system has been subjected to formal verification, so the main risk resides in application design errors that allow an opponent to manipulate the transactions provided by the device to authorized users. Most users of the 4758 use an application called CCA that is described in [619] and contains many features that make it difficult to use properly (these are largely the legacy of previous encryption devices with which 4758 users wished to be backward compatible.) Starting in 2000, we discovered that the application programming interface (API) which the 4758 exposed to the host contained a number of serious flaws. (Most of the other security modules on the market were worse.) The effect was that a programmer with access to the host could send the security module a series of commands that would cause it to leak PINs or keys. I'll discuss these API attacks in Chapter 18.

Finally, it should be mentioned that the main constraints on the design and manufacture of security processors are remarkably similar to those we 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. Vibration, power transients and electromagnetic interference can be a problem, but temperature is the worst. I mentioned the difficulty of passing security processors that self-destruct at -20° C through normal shipping channels, where goods are often subjected to -40° C in aircraft holds. Military equipment makers have the converse problem: their kit must be rated from -55° to $+155^\circ$ C. Some military devices use protective detonation; memory chips are potted in steel cans with a thermite charge precisely calculated to destroy the chip without causing gas release from the can. Meeting simultaneous targets for tamper resistance, temperature tolerance, radiation hardening and weight can be expensive.

16.4 Evaluation

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

The IBM paper which describes the design of the 4753 [6] proposed the following classification of attackers, which has been widely used since:

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 typical SSL accelerator card is aimed at blocking clever outsiders; the early 4753 aimed at stopping clever insiders, and the 4758 was aimed at (and certified for) blocking funded organizations. (By the way, this classification is becoming a bit dated; we see class 1 attackers renting access to class 3 equipment and so on. It's better to create an attacker profile for your application, rather than try to reuse one of these old industry standard ones.)

The FIPS certification scheme is operated by laboratories licenced by the U.S. government. The original standard, FIPS 140-1, set out four levels of protection, with level 4 being the highest, and this remained in the next version, FIPS 140-2, which was introduced in 2001. At the time of writing, only three vendors — IBM, AEP and Thales — have managed to get products certified at level 4. There is a large gap between level 4 and the next one down, level 3, where only potting is required and attacks that exploit 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 defence against IBM's class 2.

In fact, the original paper on levels of evaluation was written by IBM engineers and proposed six levels [1330]; the FIPS standard adopted the first three of these as its levels 1–3, and the proposed level 6 as its level 4 (the 4758 designer Steve Weingart tells the story in [1329]). 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.

At the time of writing (2007) there is a revised Federal standard, FIPS 140-3, out from NIST for consultation. This increases the number of levels from four to five, by adding a fifth level with additional testing required. However the standard does not deal with the problem of API attacks, and indeed neither did FIPS 140-1 or 140-2. The FIPS standard unfortunately covers only the device's resistance against direct invasive and semi-invasive hardware

attacks, of the kind discussed in this chapter, and against noninvasive attacks using techniques like power analysis, which I discuss in the next.

16.5 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.

16.5.1 The iButton

The original iButton from Dallas Semiconductor was designed to be a minimal, self-contained cryptographic processor for use in applications such as postal meters. It has a microcontroller, static RAM for keys and software, a clock and tamper sensors encased in a steel can with a lithium battery that can maintain keys in the RAM for a design life of ten years (see Figure 16.3). 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 UK 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 ticketing for the İstanbul mass transit system, parking meters in Argentina, and tokens used by bar staff to logon rapidly to tills.

The iButton was a pure security product when I wrote the first edition in 2000 and some versions even had a cryptoprocessor to do public-key operations. In December 1999 the first break of an iButton took place; the printer vendor Lexmark had started incorporating iButtons in some of its printer cartridges in 1998, to stop aftermarket vendors selling compatible ones, and Static Control Components finally broke the system and produced a compatible chip for the aftermarket in December 1999 [1221]. Since then Dallas has become a subsidiary of Maxim and the iButton has evolved into a broader product range. There are now versions containing temperature sensors that provide a dependable temperature history of perishable goods in transit, simple versions that are in effect just RFID tokens, and complex versions that contain a JVM. So it's no longer the case that all iButtons are 'secure': any given device might not be designed to be, or it might have been programmed (deliberately or otherwise) not to be. However the range still does include tamper-resistant devices that use SHA-1 to support cryptographic authentication, and it's still widely used in security applications.

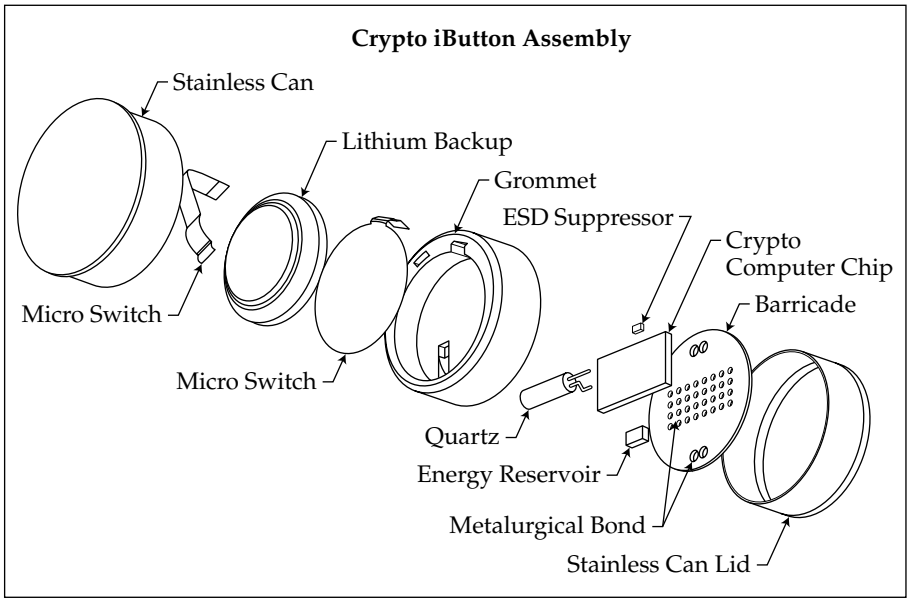

Figure 16.3: iButton internals (courtesy of Dallas Semiconductor Inc.)

How might a secure 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 and then either probing the device in operation, or disabling the tamper sensing circuitry. As the iButton has lid switches to detect the can being opened, and as the processor is mounted upside-down on the circuit board and is also protected by a mesh in the top metal layer of the chip, this is unlikely to be a trivial exercise and might well involve building custom jigs and tools. I wrote in the first edition, 'it's a tempting target for the next bright graduate student who wants to win his spurs as a hardware hacker', and SCC appear to have risen to the challenge. (Lexmark then sued them under the DMCA and lost; I'll discuss this further in section 22.6 on accessory control.)

16.5.2 The Dallas 5000 Series

Another medium-grade security device from Maxim / Dallas is the 5000 series secure 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 which 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 which 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 the 5002 (1995) fell to an ingenious protocol attack by Markus Kuhn, the *cipher instruction search attack* [748]. The idea is that some of the processor's instructions have a visible external effect, such as I/O. In particular, there is one instruction which causes the next byte in memory to be output to the device's parallel port. So 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. The problem has since been fixed, and Dallas is selling successor products such as the 5250. However, the attack on bus encryption 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.

16.5.3 FPGA Security, and the Clipper Chip

In 1993, the security world was convulsed when the US government introduced the *Clipper chip* as a proposed 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 has 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 protocol vulnerability was found [183]. As the cryptographic checksum used to bind the LEAF to the message was only 16 bits long, it was possible to feed message keys into

the device until you got one with a given LEAF, 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, but 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 [939].

Of more interest here are the tamper protection mechanisms used, which were perhaps the most sophisticated in any single-chip device at the time, and were claimed at the time to be sufficient to withstand a 'very sophisticated, well funded adversary' [937]. Although it was claimed that the Clipper chip would be unclassified and exportable, I was never able to get hold of one for dismantling despite repeated attempts.

Its successor is the QuickLogic military FPGA. This is designed to enable its users to conceal proprietary algorithms from their customers, and 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 which separates the two metal layers. Further details and micrographs can be found in a paper in the relevant data book [547].

FPGA security has since become a thriving research field. There are basically four approaches to reverse engineering an antifuse device.

- The easiest way to read out an FPGA is usually to use the test circuit that's provided in order to read back and verify the bitstream during programming. Until recently, many chips disabled this by melting a single fuse after programming. If you could get sample devices and a programmer, you could find out where this fuse was located, for example by differential optical probing [1186]. You could then use a focussed ion beam workstation to repair the fuse. Once you have re-enabled the test circuit you can read out the bitstream. Turning this into a netlist isn't entirely straightforward but it's not impossible given patience. This attack technique works not just for antifuse FPGAs but also for the Flash and EEPROM varieties.
- With antifuse FPGAs, it's sometimes possible to abuse the programming circuit. This circuit is designed to send a pulse to melt the fuse, but stop once the resistance drops, as this means the metal has melted and established contact. If the pulse weren't stopped then the metal might vaporise and go open circuit again. So the programming circuit has sensors to detect whether a fuse is open or short, and if these aren't sufficiently disabled after programming they can be used to read the bitstream out.

- The brute-force approach is to determine the presence of blown antifuses using optical or electron microscopy, for example by removing the top metal layer of the chip and looking at the vias directly. This can be extremely tedious, but brute force attacks have been done on some security processors.
- Where the device implements a cryptographic algorithm, or some other piece of logic with a relatively compact description, an inference attack using side-channel data may be the fastest way in. Most devices manufactured before about 2000 are rather vulnerable to power analysis, which can be used to reconstruct not just the keys of block ciphers but also the block cipher design by studying the device's power consumption. I'll describe this in more detail in the next chapter. A more direct attack is to drop microprobes directly on the gate array's bus lines and look at the signals. Other possible sensors include optical probing, electromagnetic coils, voltage contrast microscopy and a growing number of other chip testing techniques. In any case, the point is that it may be faster to reconstruct an algorithm from observing on-chip signals than from doing a full circuit reconstruction.

So this technology isn't infallible, but used intelligently it certainly has potential. Its main use nowadays is in FPGAs that are used to protect designs from reverse engineering.

In recent years, most FPGAs sold have had conventional memory rather than antifuse, which means that they can be made reprogrammable. They also have a higher memory density, albeit at some cost in power, size and performance compared to ASICs or antifuse devices. The current market leaders are volatile FPGAs that use SRAM, and many chips don't really have any protection beyond the obscurity of the bitstream encoding; you can read the unencrypted bitstream in transit to the FPGA on power-up. Others have one or more embedded keys kept in nonvolatile memory, and the bitstream is uploaded and then decrypted on power-up. Non-volatile devices generally use Flash for the bitstream, and reprogramming involves sending the device an encrypted bitstream with a message authentication code to guarantee integrity. In both cases, there may be a few fuses to protect the key material and the protection state. The better designs are so arranged that an attacker would have to find several fuses and antifuses and wire them in a particular way in order to extract the bitstream. Saar Drimer has a survey of FPGA design security [400].

Such FPGAs are used in equipment from routers through printers to cameras, and the main threat models are that a manufacturing subcontractor may try to overbuild the design so as to sell the extra products on the grey market, or that a competitor will try to make compatible accessories. In the former case, the vendor can separate programming from assembly, and also use the

upgrade cycle to undermine the value of any black-market copies. In the latter case the FPGA logic typically implements an authentication mechanism of some kind.

One trap to watch out for when designing a field-upgradeable product using an FPGA is to prevent service denial attacks via the upgrade mechanism. For example, a Flash FPGA usually only has enough memory for one copy of the bitstream, not two; so the usual drill is to read in the bitstream once to decrypt it and verify the MAC, and then a second time to actually reprogram the part. If the bitstream supplied the second time is corrupt, then the likely outcome is a dead product. The typical SRAM-based FPGAs doesn't have bitstream authentication, so people could usually replay old bitstreams and thus escape an upgrade if they wish. It's also possible that if an attacker gets your products to load a random encrypted bitstream, this could cause short circuits and do permanent damage. So it's worth thinking whether anyone might be malicious enough to try to destroy your installed product base via a corrupt upgrade, and if so you might consider whether your product needs a secure bitstream loader.

16.6 Smartcards and Microcontrollers

In volume terms, the most common secure processors nowadays are self-contained one-ship security processors containing nonvolatile memory, I/O, usually a CPU, often some specialised logic, and some mechanisms to prevent memory contents from being read out. They range from microcontrollers with a memory-protect bit at the low end, up to smartcards and the TPMs that now ship with most computer motherboards. Smartcards have quite a range of capabilities. At the low end are phone cards costing under a dollar; then there are bank cards that will do conventional crypto, with 8/16 bit processors; at the top end there are bank cards that will do public-key crypto and often have 32-bit processors. There are also 'contactless' smartcards — essentially RFID devices that can perform cryptographic protocols rather than just emitting static data — that are used in applications such as transport ticketing.

As these devices are cheap and sold in large volumes, an opponent can often obtain many samples and take them away to probe at her leisure. So many attacks on them have been developed[2]. Pay-TV subscriber cards in particular have been subjected to many cloning attacks and nowadays have a lot of special protection mechanisms. In this section, I'll tell the story of how smartcard security evolved with successive attacks and defenses.

[2]As this book was about to go to press at the end of 2007, there was an announcement at the Chaos Computer Club conference in Berlin that the widely-used Mifare contactless system had been reverse engineered and turned out to use a weak 48-bit stream cipher.

16.6.1 History

Although smartcards are marketed as a 'new' security solution, they actually go back a long way, with the early patents (which date from the late 1960s through mid 1970s) having long since expired [383]. For a history of the development of smartcards, see [563]. 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. They started being used as bank cards in France in 1994, and in the rest of Europe from about 2005.

A smartcard is a self-contained microcontroller, with a microprocessor, memory and a serial interface integrated in a single chip and packaged in a plastic card. Smartcards used in banking use a standard size bank card, while in most mobile phones a much smaller size is used. Smartcard chips are also packaged in other ways. For example, many UK 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 TPM chips built into computer motherboards to support 'Trusted Computing' are in effect public-key smartcard chips but with added parallel ports. Contactless smartcards contain a smartcard chip plus a wire-loop antenna. In what follows I'll mostly disregard the packaging form factor and just refer to these products as 'smartcards' or 'chipcards'.

Their single most widespread application is the GSM mobile phone system, the standard in some U.S. networks and in almost all countries outside the U.S. The telephone handsets are commodity items, and are personalized for each user by the SIM, a smartcard that 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 personalisation and authentication 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 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 he has subscribed.

Chipcards are also used in a range of other applications ranging from hotel keys to public payphones — though in such applications it's still common to find low-cost devices that 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 — a chip 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 which was hashed using MD5 and stored on an EEPROM external to the processor, so a user could overwrite it with the hash of a known password and assume complete control of the device [719]. For more details of attacks specific to microcontrollers, see [68, 1181, 1183].

16.6.2 Architecture

The typical smartcard consists of a single die of up to 25 square millimeters of silicon containing a microprocessor (larger dies are more likely to break as the card is flexed). Cheap products have an 8-bit processor such as an 8051 or 6805, and the more expensive products either have a 32-bit processor such as an ARM, or else a dedicated modular multiplication circuit to do public-key cryptography. It also has serial I/O circuitry and a hierarchy of three classes of memory — ROM or Flash 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 2007 might have 64 Kbytes of ROM, 8–32 Kbytes of EEPROM and 2K 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. These specifications have not changed much since 2000; prices for volumes of a few hundred cards haven't changed much either (maybe $3 for an 8-bit card that will do DES and AES, $7 for a card with a cryptoprocessor to do elliptic curve crypto, and $12 for one beefy enough for RSA). Large-volume users pay much less. The main change over the past seven years is that development kits are now widely available, whereas in 2000 most vendors screened purchasers and imposed NDAs. The implication is of course that if an opponent can reverse engineer your smartcard application, she has no difficulty buying and programming cards.

16.6.3 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 since the machinery needed to make the card cost $20m, 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, and so until then they relied on the small size of the devices, the obscurity of their design, and the rarity of chip testing tools to make attacks more difficult.

The application that changed all this was satellite TV. TV operators broadcast their signals over a large footprint — such as all of Europe — and give each subscriber a card that will compute the keys needed to decipher the channels they've 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. A critical factor was that 'Star Trek', which people in Europe had picked up from UK satellite broadcasts for years, was suddenly encrypted in 1993. In some countries, such as Germany, it simply wasn't available legally at any price. This motivated a lot of keen young computer science and engineering students to look for vulnerabilities.

There have since been large financial frauds 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 [900].

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, and then sent messages over the air to cancel those channels to which the customer hadn't subscribed after an introductory period. This opened an attack in which a device was inserted between the smartcard and the decoder which would 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 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 chip cards in brothels and in hostels for asylum seekers [1210]. 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 [124]. 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 what follows, we'll concentrate on the attacks that are peculiar to smartcards.

How to hack a smartcard (2)

As smartcards use an external power supply, and store security state such as crypto keys and value counters in EEPROM, 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 the 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 very 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.)

So many smartcard processors have a circuit to detect low clock frequency and either freeze or reset the card; others use dynamic logic to achieve the same effect. 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. So 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. Until a few years ago, most smartcards had no protection against physical tampering except 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 [136]. In most cases, a few milliliters of fuming nitric acid are all that's required to dissolve the epoxy, and the passivation layer is 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 (see Figure 16.4). They may have specialized accessories, such as a laser to shoot holes in the chip's passivation layer.

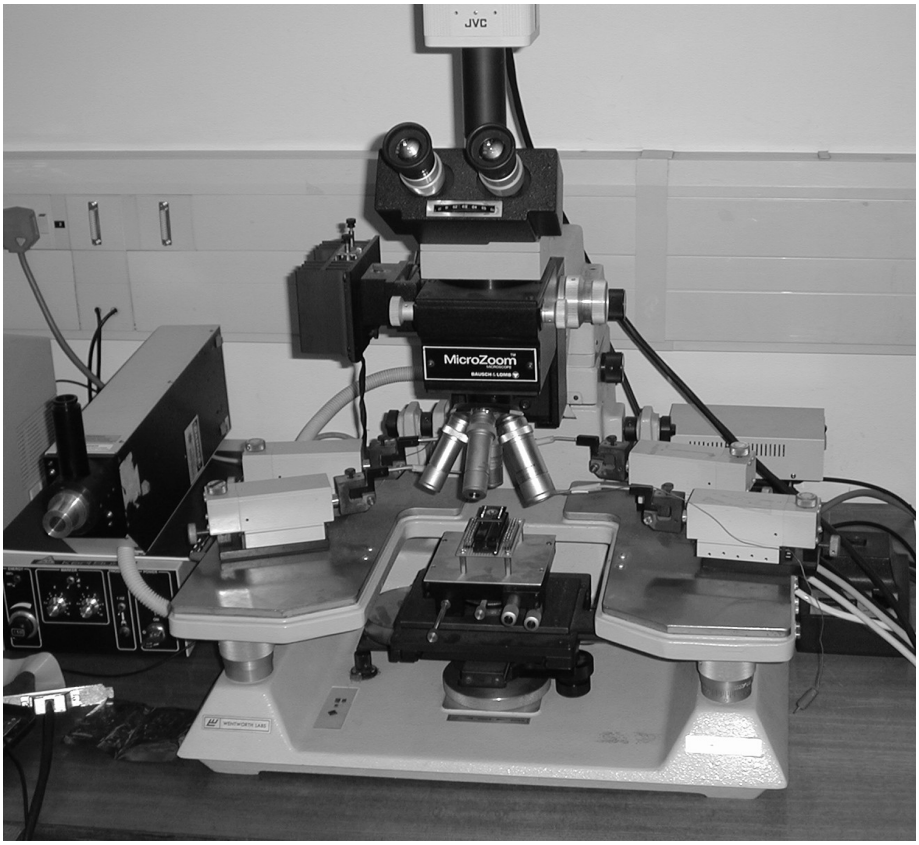

Figure 16.4: Low-cost probing station

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) which will give him a complete listing of the card memory contents. So the attacker will identify the bus and expose the bus lines for probing (see Figure 16.5). Indeed, if the chip is performing a known cryptographic protocol with well understood algorithms, then unless there's some defense mechanism such as lots of dummy instructions, a trace from a single bus line is likely to give away the key [580].

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 legitimate cards 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.

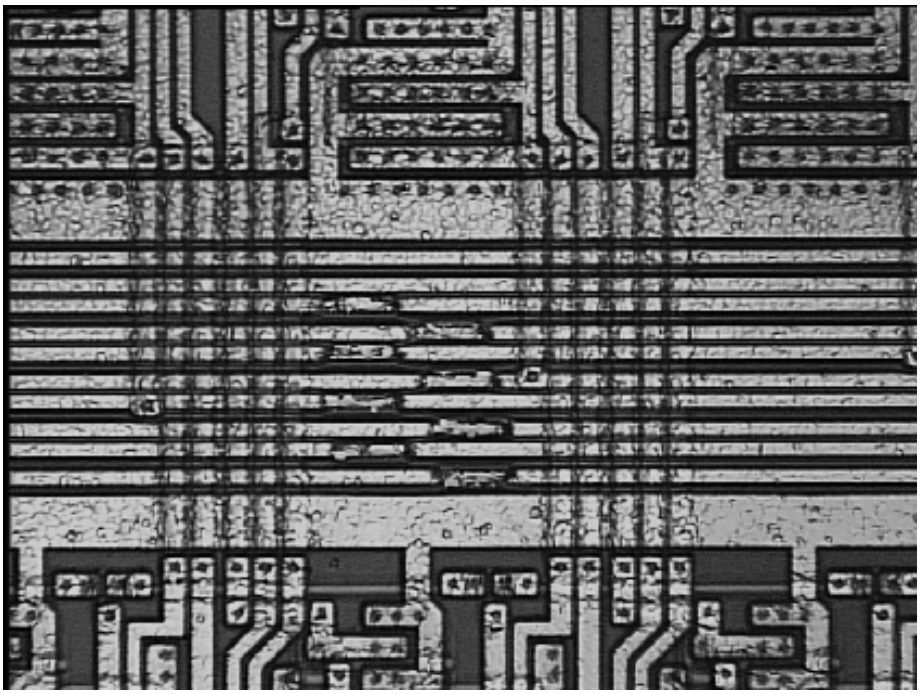

Figure 16.5: The data bus of an ST16 smartcard prepared for probing by excavating eight trenches through the passivation layer with laser shots (Photo courtesy Oliver Kömmerling)

How to hack a smartcard (5)

The defeat for this strategy was Oliver Kömmerling's *memory linearization attack* in which 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 [733]. 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 is 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 which 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.

Memory linearization is an example of a *fault induction attack*. There are quite a few examples of this; faults can be injected into smartcards and other security processors in a number of ways, from hardware probing through power transients to our old friend, the stack overflow. A variant on the theme is given by attacks on the card's 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. As with FPGAs, a fuse is then blown in the chip to stop an attacker using the same facility. All that the attacker had to do was to cause a fault in this mechanism — by finding the fuse and repairing it. This could be as simple as bridging it with two probing needles [212]. 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.

It's also possible to exploit already existing hardware bugs. For example, Adi Shamir pointed out that if a CPU has an error in its multiply unit — even just a single computation $ab = c$ whose result is returned consistently wrong in a single bit — then it's possible to send it an RSA ciphertext for decryption (or an RSA plaintext for signature) constructed so that the computation will be done correctly mod p but incorrectly mod q; if this result is returned to the attacker, he can instantly work out the private key [1148]. For this reason, a careful programmer will check the results of critical computations.

How to hack a smartcard (6)

The next thing the pay-TV card industry tried was to incorporate hardware cryptographic processors, in order 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, and it had an interesting protocol failure: it would always work out the key needed to decrypt the current video stream, and then pass it to the CPU which would decide whether or not to pass it on to the outside world. Hackers broke this by developing a way to tap into the wiring between the two chips.

Later 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 [196]. However, assembling all the equipment, writing the software and integrating the systems involves huge effort and expense.

A much simpler, and common, attack is for pirates to ask one of the half 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 integrated circuits on behalf of the chip maker's competitors, looking for patent infringements. They also reverse engineer chips used for accessory control in consumer electronic devices, as doing this for compatibility rather than piracy is quite lawful. So they are used to operating in conditions of some secrecy, and there's been at least one case in which a pirate had a pay-TV card reverse engineered by an above-board company by pretending to be a bona fide business.

How to hack a smartcard (7)

The next defense 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 and trigger a self destruct mechanism.

We mentioned such meshes in connection with the Dallas processors; after the usual initial crop of implementation blunders, they 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* or FIB. For a detailed description of FIBs and other semiconductor test equipment that can be used in reverse engineering,

see Martin [837]; in brief, a FIB is a device similar to a scanning electron microscope but uses a beam of ions instead of electrons. By varying the beam current, it can be used either as a microscope or as a milling machine, with a useful resolution under 10 nanometers. 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 the prices are tumbling. Many universities and industrial labs now have one, and time on them can 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 16.6).

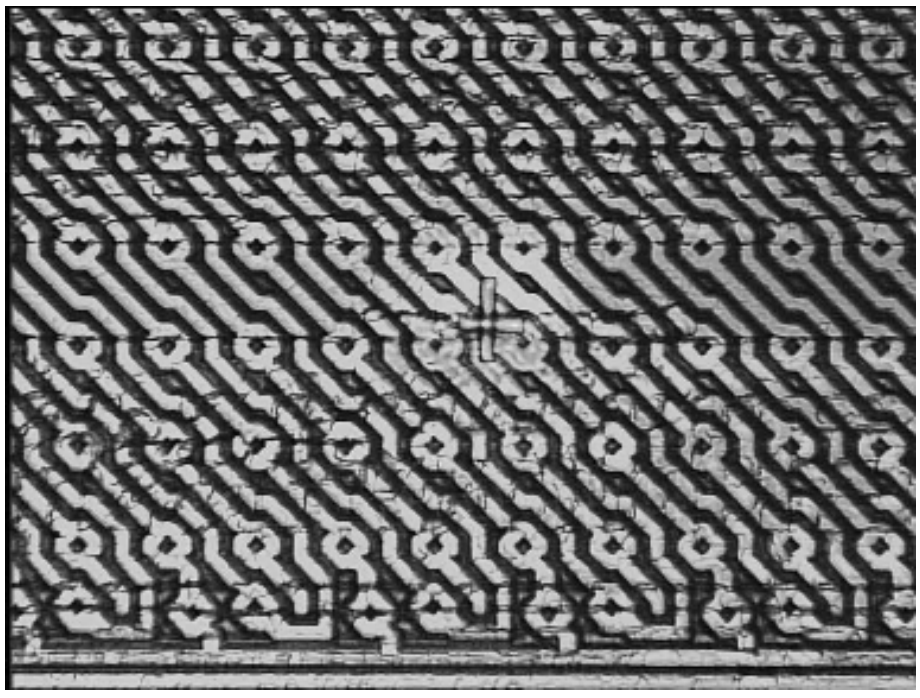

Figure 16.6: The protective mesh of an ST16 smartcard with a FIB cross for probing the bus line visible underneath (Photo courtesy Oliver Kömmerling)

Defeating a sensor mesh that is continually powered up is much harder, but some tools exist. 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.

Many other defensive techniques can force the attacker to do more work. Some chips are packaged in much thicker glass than in a normal passivation layer. The idea is that the obvious ways of removing this (such as hydrofluoric acid) are likely to damage the chip. 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, Md). Such coatings can force the FIB operator to go slowly rather than damage the chip through a build-up of electrical charge.

How to hack a smartcard (8)

In 1998, the smartcard industry was shaken when Paul Kocher announced a new attack known as *differential power analysis*. This relied on the fact that different instructions consumed different amounts of power, so by measuring the power consumed by a smartcard chip it was possible to extract the key. This had always been known to be theoretically possible, but Kocher came up with efficient signal processing techniques that made it easy, and which I'll describe in the following chapter on emission security. He came up with even simpler attacks based on timing; if cryptographic operations don't take the same number of clock cycles, this can leak key material too. On larger processors, it can be even worse; a number of researchers have developed attacks on crypto algorithms based on cache misses. Power and timing attacks are examples of *side-channel attacks*, where the opponent can obtain and exploit some extra information about the processor's state while a cryptographic computation is going on. Essentially all the smartcards on the market at that time turned out to be vulnerable to power analysis or other side-channel attacks, and this held up the industry's development for a couple of years while countermeasures were developed.

We classify mechanical probing as an *invasive attack* as it involves penetrating the passivation layer, and power analysis as a *noninvasive attack* as the smartcard is left untouched; the attacker merely observes its operation. Noninvasive attacks can be further classified into local attacks, where the opponent needs access to the device, as with power analysis; and remote noninvasive attacks, where she could be anywhere. Timing attacks, and protocol attacks, are examples of the latter.

A lot of engineering effort was invested in the late 1990s and early 2000s in making noninvasive attacks more difficult. This is actually more complex than it might seem, as the measures one can take to make one noninvasive

attack more difficult often turn out to make another one easier. I'll discuss the problems in more detail in the next chapter.

How to hack a smartcard (9)

Mechanical probing techniques worked until the early 2000s, but since then they have been increasingly difficult because of shrinking feature sizes. The next attack technology to develop was optical probing. There was a first report from Sandia National Laboratories in 1995 of a way to read out a voltage directly using a laser [17]. Since 2001 optical probing has been developed into an effective and low-cost technology, largely by my colleague Sergei Skorobogatov at Cambridge. In 2002 we reported using a photographic flashgun, mounted on the microscope of a probing station, to induce transient faults in selected transistors of an IC [1187]. The light ionises the silicon, causing transistors to conduct. Once you can focus the light on single transistors, whether by using a metal mask or by upgrading from a flashgun to a laser, this enables direct attacks on many chips. For example, microcontrollers can be opened by zapping the flip-flop that latches their protection state.

Later that year, Sergei reported using a laser mounted on the same cheap microscope to read out the memory contents of a microcontroller directly [1111]. The basic idea is simple: if you use a laser to make a transistor conduct, this will increase the device's power consumption unless it was conducting already. By scanning the laser across the device, it is possible to make a map of which transistors are off and which are on. We developed this into a reasonably dependable way of reading out memory cells [1111].

How to hack a smartcard (10)

At the time of writing (2007), smartcard vendors are using 0.18 and 0.13 micron processes which typically have seven metal layers. Direct optical probe attacks on the logic from the surface of the chip are now difficult because of the feature size and because the metal layers get in the way, dispersing the laser light. The faults that are induced are thus more random and less targeted. In addition, the sheer size and complexity of the chips makes it difficult to know where to aim. This is made worse by the use of *glue logic* — essentially randomised place-and-route.

As you can see in Figure 16.7, the older MC68 device has clearly distinguishable blocks, and quite a lot can be learned about its structure and organisation just by looking at it. Bus lines can be picked out and targeted for attack. However, the SX28 in Figure 16.8 just looks like a random sea of gates. The only easily distinguishable features are the EEPROM (at top left) and the RAM (at top right). And the SX28 is a relatively old design; more modern chips are planarised by chemical-mechanical polishing so that even fewer details are visible, unless infrared light is used.

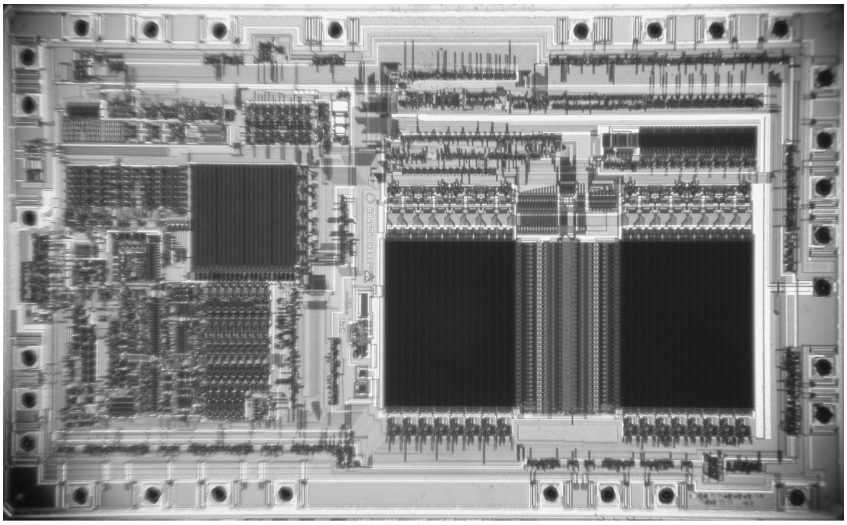

Figure 16.7: MC68HC705P6A microcontroller (courtesy of Sergei Skorobogatov)

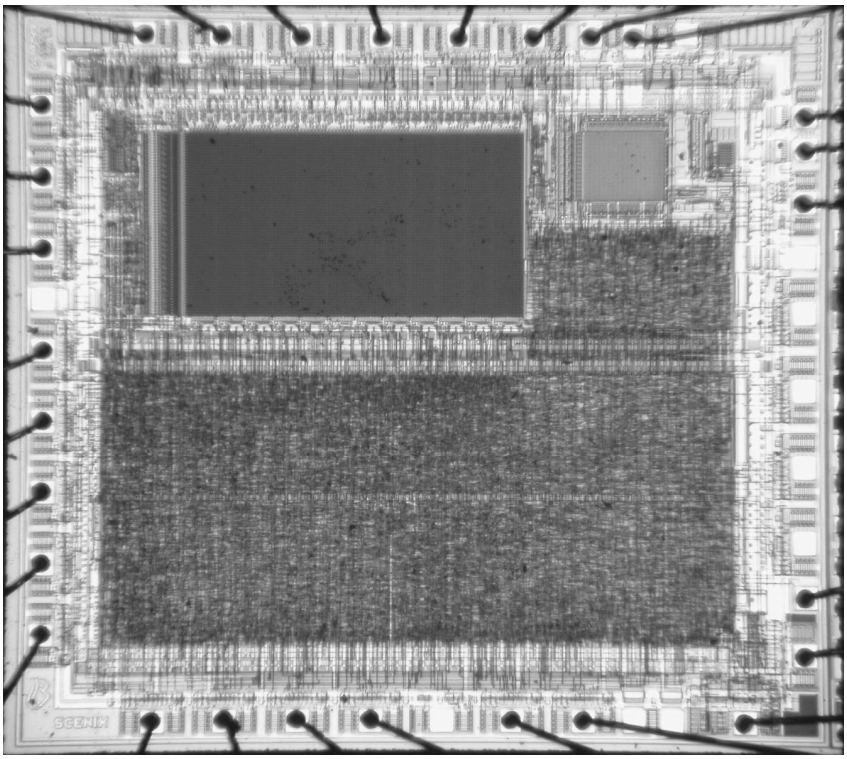

Figure 16.8: SX28 microcontroller with 'glue logic' (courtesy of Sergei Skorobogatov)

The two current windows of vulnerability are the memory and the rear side. Static RAM cells are still fairly large; even with 0.13 micron process an SRAM cell is about 1.2 by 1.7 microns and can thus be individually targeted, whether for fault induction or for read-out of its contents. Rear-side optical attacks use the fact that silicon is transparent at wavelengths greater than 1.1 microns, while rear-side probing attacks use FIBs with special equipment for the purpose.

16.6.4 The State of the Art

The levels of physical tamper-resistance available from the best vendors has improved significantly over the last few years. Much of this comes from the smaller features, more metal layers, and greater design complexity of the latest devices. In the 2001 edition of this book, I wrote, 'there isn't any technology, or combination of technologies, known to me which can make a smartcard resistant to penetration by a skilled and determined attacker'. This is still almost true, but nowadays, it can take a lot of time and money to penetrate a good design. If you can't find an easier way in and have to reverse engineer the entire chip, you'll have to cope with hundreds of thousands of transistors laid out in randomized glue logic with perhaps seven metal layers. One approach is to have sophisticated layout-reconstruction software; another is to send micrographs to a subcontractor in China that just throws a lot of people at the problem. Either way, you can be looking at a year's delay, a budget of over a million dollars, and no certainty of success.

I know of one case where companies spent a huge amount of money (and several years) trying to reverse an authentication chip. That was the Sony MagicGate, a chip found in Playstation accessories and which is recognised by security logic in the Playstation's graphics chip. This used some interesting protection tricks, and an authentication protocol that was both simple (so protocol attacks couldn't be found) and randomised (so that attackers couldn't learn anything from repeating transactions). Most designs aren't that good. Many chips remain vulnerable to side-channel attacks such as power analysis, which I'll discuss in Chapter 17. Many are too complex, or become complex over time, leading to logical vulnerabilities; I will discuss API attacks in Chapter 18.

A final problem is our old friend, the market for lemons. There are smartcard products out there that give very high levels of protection, and others that don't. Needless to say, the vendors all claim that their products are secure. Some of them have Common Criteria evaluations, but these are difficult to interpret; there was at least one case of a chip advertised with a very high rating, that was easy to probe (a careful reading of the evaluation certificate would have revealed that it applied only to the card's operating system, not to its hardware).

So although, at the silicon level, smartcards are usually much harder to copy than magnetic stripe cards, the level of protection they give in an actual application will depend critically on what products you select and how cleverly you use them.

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

16.6.4.1 Defense in Depth

The first, used by pay-TV companies, is defense in depth. Smartcards may use nonstandard processors with custom instruction sets, hardware crypto-processors, registers that use dynamic rather than static RAM to prevent single-stepping, and 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 attack, and to force the attacker to go to the trouble of reverse engineering substantially the whole system. If you are prepared to spend serious money on revenue protection, and you have access to serious expertise, the full-custom route is the way to go.

Technical measures on their own are not enough, though. It's important to think through the business model. Over the last few years of the 20th 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 the chapter on copyright.) And if you can be sure that the opponent has to reconstruct your chip fully before he can threaten your revenue, and you're confident that will take him a year and a million dollars, then you can keep him perpetutally a year behind, and this may be enough. If you also have the ability to refresh your security quickly — say by sending your subscribers a new smartcard — then you can undermine his revenue expectations still further and deter him from entering into the game.

16.6.4.2 Stop Loss

Whether you go for the defense-in-depth approach will often depend on the extent to which you can limit the losses that result from a single card's 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 can 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 22.2.4.3. This makes the cards less attractive targets.

In other systems, such telephone SIM cards and EMV bank cards, each device contains a key to authorise transactions — whether calls or debits — on a single customer's account. Probing a key out of a card is thus pretty well equivalent to stealing a card, or setting up an account in the target's name and getting a genuine card issued against his credit rating. Attacks that cost millions to research and many thousands per card to carry out are not usually the real problem here. The main threat rather comes from noninvasive attacks, where people are duped into putting their cards into wicked terminals. Although it is conceivable that such a terminal could do a power analysis attack, in practice it's the protocol vulnerabilities that provide the weakest link. So it is quite logical for banks and phone companies to use cheap commodity cards. The system of merchant floor limits, random online authorizations, lists of hot cards and so on, limits the possible damage. (It's still objectionable, though, for the banks to say 'We use smartcards which are secure, therefore any disputed transaction is your fault.')

16.7 What Goes Wrong

There are failure modes of systems involving tamper-resistant processors that are more or less independent of whether the device is a low-cost smartcard or a high-end banking security module.

16.7.1 The Trusted Interface Problem

None of the devices described in the above 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 a given metal key (or smartcard) can perform certain privileged transactions. But whether you use a $2000 4758 or a $2 smartcard to do digital signatures, you still trust the PC that drives them. If it shows you a text saying '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. Indeed, it may even make your situation worse. Banks in many countries used to move to electronic systems as an opportunity to change their contract terms and conditions, so as to undermine consumer protection [201]. In the UK in particular, smartcards have been more a liability engineering technology than a protective one; complaints are answered with a standard refrain of 'your chip and PIN card was used, so it's your fault.'

This is usually a user interface issue, though not always. Recall the digital tachographs we discussed in Chapter 12; after vendors made the sensor in the truck's gearbox into a secure processor, with its own unique crypto key, the attackers build a gearbox emulator that surrounded it and fed tampered data into the sensor. Even where you put some effort into building a user interface into a tamper-resistant device, it may be compromised via the environment; the classic example here is the skimmer that is fitted to the card slot of an ATM, reads your card on the way in, and captures your PIN using a tiny camera.

Tamper-resistant processors are often able to add more value where they do not have to authenticate users or sense the environment (beyond detecting tampering attempts). Recall the example of prepayment electricity metering, in Chapter 13: there, tamper-resistant processors are used to limit the loss when a token vending machine is stolen. Tokens are generated using keys kept in a cryptoprocessor that maintains and decrypts a value counter, enforcing a credit limit on each vending machine. Postal meters work in exactly the same way.

The critical difference is that the banking application needs to authenticate the customer, while the electricity and postal metering applications don't. Other applications that use crypto chips but don't care who uses them range from accessory control in printer ink cartridges and games consoles to prepaid phone cards. There, the vendor only cares that only one person uses the product and often that they only use it so much.

16.7.2 Conflicts

A further set of issues is that where an application is implemented on a number of devices, under the control of different parties, you have to consider what happens when each party attacks the others. In banking, the card issuer, the terminal owner and the customer are different; all the interactions of cloned cards, bogus terminals and cheating banks need to be thought through. This is quite different from a building entry control application, where one firm owns the cards and the terminals; phone cards are somewhere in the middle.

Bruce Schneier and Adam Shostack suggest a framework for analysing this in [1136]. Imagine that the customer started off with a secure PC; say something like a palmtop computer with software to generate digital signatures. Now consider what happens when you split this up, for example by requiring her to use a remote keyboard and screen controlled by somebody else, or by requiring her to have the crypto done using a smartcard whose software is controlled by somebody else. Then think of what happens when any one of these components is temporarily subverted: the customer's card is stolen, or her computer gets infected. This gives a lot of abuse cases to consider, and designers habitually miss some of them (especially the cases that are hard to solve with their product, or that are embarrassing to their client). And every time you introduce a split into a system, the interface creates new possibilities

for mayhem; a survey of flaws found by a commercial evaluation laboratory showed that most of them were at the interfaces between physical, logical and organisational measures [213]. I'll discuss this at greater length in the chapter on API attacks.

There are further problems with displacement. It's common for designers to think that, just because their product contains a security processor, its protection needs have somehow been taken care of. The security box has been ticked. But because of interface and other issues, this is rarely so.

A further source of conflict and vulnerability is that many of the users of tamper resistance are firms implementing business models that restrict their customers' use of their product — such as rights management and accessory control. Their customers are therefore in some sense their enemies. They have not only access to the product, but the incentive to tamper with it if they can.

16.7.3 The Lemons Market, Risk Dumping and Evaluation

Each of the product categories discussed here, from hardware security modules down through smartcards to microcontrollers, has a wide range of offerings with an extremely wide variability in the quality of protection provided. Although quite a few products have evaluations, these don't really mean very much.

First, there are very few offerings at the highest protection level — FIPS-140 level 4 or Common Criteria levels above 4. There are very many at lower levels, where the tests are fairly easy to pass, and where vendors can also shop around for a lab that will give them an easy ride. This leads to a lemons market in which all but the best informed and motivated players will be tempted to go for the cheapest level 3 product, or even an unevaluated offering.

Second, evaluation certificates don't mean what they seem. Someone buying a 4758 in 2001 might have interpreted its level 4 evaluation to mean that it was unbreakable — and then been startled when we broke it. In fact, the FIPS certificate referred only to the hardware, and we found vulnerabilities in the software. It's happened the other way too: there's been a smartcard with a Common criteria level 6 evaluation, but that referred only to the operating system — which ran on a chip with no real defences against microprobing. I'll discuss the failings of our evaluation systems at greater length in Part III.

Third, many firms aim at using secure processors to dump risk rather than minimise it. The banks who say 'your chip and PIN card was used, so it's your fault' are by no means alone. There are many environments, from medicine to defense, where what's sought is often a certificate of security rather than real protection, and this interacts in many ways with the flaws in the evaluation system. Indeed, the main users of security evaluation are precisely those system operators whose focus is on due diligence rather than risk reduction, and they are also disproportionate users of tamper-resistant processors.

16.7.4 Security-By-Obscurity

Until very recently, the designers of cryptoprocessors tried hard to keep their products' design obscure. You had to sign an NDA to get smartcard development tools, and until the mid-1990s there was just no serious information available on how smartcards could be attacked. Their vendors just did not imagine that attackers might buy semiconductor lab tools second-hand and find ingenious ways to use them to probe data out. The security target still used for evaluating many smartcards under the Common Criteria focuses on maintaining obscurity of the design. Chip masks have to be secret, staff have to be vetted, developers have to sign NDAs — there were many requirements that pushed up industry's costs [448]. Obscurity was also a common requirement for export approval, leading to a suspicion that it covers up deliberately inserted vulnerabilities. For example, a card we tested would always produce the same value when instructed to generate a private / public keypair and output the public part.

In short, the industry's security culture was inappropriate. Almost none of the actual attacks on fielded smartcard systems used inside information. Most of them started out with a probing attack or side-channel attack on a card bought at retail. The industry did not do hostile attacks on its own products, so the products were weak and were eventually subjected to hostile attack by others. The culture is now changing and some organisations, such as VISA, specify penetration testing [1300]. However, the incentives are still wrong; a sensible vendor will go to whatever evaluation lab gives him the easiest ride, rather than to an ace attack team that'll find dozens of extra vulnerabilities in the product and destroy its chances of launching on time. We'll return to this subject to discuss the underlying economics and politics of evaluation in section 26.3.3.1.

16.7.5 Interaction with Policy

There are many other unexpected interactions with the world of policy. A good example was the drive that started during the dotcom boom to have smartcards adopted as the preferred device for digital signatures where people interact with government. The European Union passed a law that gave a strong presumption of validity to electronic signatures made using approved smartcards. This was irrational, given the lack of a trusted user interface; for signing documents it would be better to use something like a PDA, where at least the customer can at least see what she's signing and protect the device using common sense [111]. Yet the smartcard vendors lobbied hard and got their law. Thankfully, this was completely counterproductive: the law moved the liability for forged signatures from the relying party to the party whose key was apparently used. By accepting such a device, you were in effect saying, 'I

agree to be bound by any signature that appears to have been made by this device, regardless of whether or not I actually made it'. This is unattractive and has helped limit the use of digital signatures to niche applications.

16.7.6 Function Creep

We've already seen numerous examples of how function creep, and changes in environmental conditions, have broken secure systems by undermining their design assumptions. I mentioned, for example, how replacing paper passports (that are unique) with electronic passports (that are easy to copy) undermines the bail system; criminals awaiting trial could flee the country using copies of their passports. A more general problem is when multiple applications are put on the same card. If you deploy a phone banking system, then the SIM card that was previously just a means of identifying people to the phone company now becomes a token that controls access to their bank accounts. It follows that it may attract a much higher grade of attacker. Does this matter? In the present (2007) environment of growing phishing attacks, I'd say it probably doesn't; using text messages to confirm bank transactions gives a valuable second authentication channel (the main risk, as described in Chapter 2, is probably that a fraudster talks the phone company into issuing a SIM to the wrong person).

But what will happen in five or ten years' time, once everyone is doing it? What if the iPhone takes off as Apple hopes, so that everyone uses an iPhone not just as their phone, but as their web browser? All of a sudden the two authentication channels have shrunk to one; both the SMS messages and the IP traffic to the bank web site go through the same operating system, which is now deployed on a large enough scale to attract attention from the gentlemen in Russia.

16.8 So What Should One Protect?

It's common enough that the purpose for which a technology was first invented or marketed isn't the one for which it takes off. The steam engine was invented for pumping water out of coal mines, and the telephone to enable post office clerks to read text to telegraph operators rather than sending it through a pneumatic tube. Similarly, the inventors of smartcards thought they would be a hit in banking; yet they first took off in phone cards, then pay-TV. Despite their use in banking in Europe, they have yet to catch on in markets like the USA where bank customers have better legal protection. The largest sales volumes of the lowest cost crypto chips are in accessory control applications such as printer cartridges and console games.

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 grey market, but so long as it isn't copied the station operator isn't too concerned. Another example comes from accessory control chips such as the Sony MagicGate; any PlayStation should work with any PlayStation memory cartridge, but not with a cheap Korean competitor. 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. This enables you to link information protection with physical and operational protection.

Second, they can be used to control value counters, as with the electricity prepayment and postal metering discussed in Chapter 12 and in section 16.7.1 above. These typically use devices such as the iButton to hold keys and also a credit counter. Even if the device is stolen, the total value of the service it can vend is limited. A related application is in printers where ink cartridges are often programmed to dispense only so much ink and then declare themselves to be dry.

Third, they can reduce the need to trust human operators. 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 device, and only so 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. In effect, they not only implement a separation of duty policy, but transfer a good portion of the trust from people to things. If these things can be physically controlled — whether by their classification or their sheer mass — that reduces the exposure from both treachery and carelessness.

This is an incomplete list. But what these applications, and their underlying principles, have in common is that a security property can be provided independently of the trustworthiness of the surrounding environment. In other words, be careful when using tamper resistant devices to try to offset the lack of a trustworthy 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.

16.9 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, from hardware security modules that cost thousands of dollars and are certified to resist all currently known attacks, through smartcards whose hardware can now be quite difficult to penetrate, down to low-cost security microcontrollers that can often be defeated with a few hours to days of work.

I've told the story of how hardware tamper-resistance developed through the 1990s through a series of cycles of attack and improved defence, and given many examples of applications. Almost regardless of their price point, security processors are typically vulnerable to attacks on interfaces (human, sensor or system) but can often deliver value in applications where we need to link processing to physical objects and to keep track of value in the absence of a dependable online service.

Research Problems

There are basically two 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 mid-range prices and sizes, and how mid-range 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?

These are intertwined with research into emission security and into the protection of application programming interfaces, which I'll discuss in the next two chapters.

Further Reading

Colleagues and I wrote a survey of security processors [65] and I'd recommend that if you're looking for a more detailed treatment of the material in this chapter. Beyond that, there's a tutorial by Sergei Skorobogatov at [1186]. The best current research in the field usually appears in the proceedings of CHES — the workshop on Cryptographic Hardware and Embedded Systems. FPGA security is reviewed at [400], and the other good reads include Bunnie Huang's book on hacking the Xbox [629].

For the early history of crypto, including things like weighted codebooks and water-soluble inks, the source is of course Kahn [676]. For a handbook on the chip card technology of the mid-to-late 1990s, see [1056], while the gory details of tampering attacks on those generations of cards can be found in [68, 69, 733]. The IBM and Dallas products mentioned have extensive documentation online [641], where you can also find the U.S. FIPS documents [936].