# Cryptographic Processors—A Survey

Ross Anderson, Mike Bond, Jolyon Clulow and Sergei Skorobogatov

*Abstract*— **Tamper-resistant cryptographic processors are becoming the standard way to enforce data-usage policies. Their history began with military cipher machines, and hardware security modules used to encrypt the PINs that bank customers use to authenticate themselves to ATMs. In both cases, the designers wanted to prevent abuse of data and key material should a device fall into the wrong hands. From these specialist beginnings, cryptoprocessors spread into devices such as prepayment electricity meters, and the vending machines that sell credit for them. In the 90s, tamper-resistant smartcards became integral to GSM mobile phone identification and to key management in pay-TV set-top boxes, while secure microcontrollers were used in remote key entry devices for cars. In the last five years, dedicated crypto chips have been embedded in devices from games console accessories to printer ink cartridges, to control product and accessory aftermarkets. The 'Trusted Computing' initiative will soon embed cryptoprocessors in PCs so that they can identify each other remotely.**

**This paper surveys the range of applications of tamper-resistant hardware, and the array of attack and defence mechanisms which have evolved in the tamper-resistance arms race.**

*Keywords*— **cryptoprocessor, HSM, security API, survey, fault analysis, power analysis, semi-invasive attack**

## I. Introduction

The combination of cryptography and tamper-resistance first appeared in military applications such as securing communications links. The spread of Automated Teller Machine (ATM) networks brought the technology into the commercial mainstream. The devices used for protecting ATM networks were subsequently adapted for other applications such as prepayment electricity meter.

A typical high-end cryptoprocessor is a physically tamper-resistant embedded processor which communicates with a conventional PC or mainframe and performs a predefined set of cryptographic operations using keys that are protected within the device.

Such a cryptoprocessor typically enforces a policy on the use of the keys it protects. For example, in an ATM network, the network cryptoprocessor may allow verification of incoming customer Personal Identification Numbers (PINs) but not generation of PINs for new accounts. The Application Programming Interface (API) which such a device presents is called the *security API* and will implement the device's security policy. We discuss security APIs in sections V and VI.

During the 1990s, cryptoprocessors gained more uses: protecting Secure Socket Layer (SSL) keys used by webservers, and defending proprietary software and algorithms from theft by employees; low-cost cryptoprocessors such as smartcards and secure microcontrollers also became commonplace.

The authors are with the Computer Laboratory, University of Cambridge, JJ Thomson Avenue, Cambridge CB3 0FD, UK forename.surname@cl.cam.ac.uk

A whole host of embedded applications for cryptoprocessors now exist: smartcards for holding decryption keys for pay-TV; lottery ticket vending machines; and mobile-phone top-up systems. Modern electronic payment schemes such as EMV use smartcards and readers at the front end, and larger cryptoprocessors at the back end, to control the flow of electronic money. Tamper-resistant hardware is even deployed in an effort to secure electronic voting terminals from attack.

The latest applications of tamper-resistant processors are in Digital Rights Management (DRM) and Trusted Computing (TC). Content owners are looking towards tamper-resistant processors with security APIs that can enforce arbitrary policies on the way that content is processed. The range of possible applications is incredible, and – to some observers – disturbing [10]. The entertainment industry in particular seeks new revenue streams by using security APIs to release media according to new rules, such as music subscription services, and to enforce finer market segmentation.

In section II we describe possible applications in more detail, and in section III we provide a taxonomy of cryptoprocessors and attacks. Section IV considers attacks involving physical access to the device, while sections V and VI describe logical attacks on the security APIs of cryptoprocessors. Finally, sections VII and VIII look at issues of policy and draw some conclusions.

## II. Applications

### A. Automated Teller Machine Security

ATMs were the 'killer application' that got cryptography into wide use outside of military and diplomatic circles, and remain today a high-volume use for tamper-resistant hardware. In the 70s, IBM developed a system for authenticating customers to ATMs. Bank customers were issued with PINs, computed from their account numbers using a secret DES key, the *PIN derivation key*. References [3], [4] include descriptions of the launch of the 3614 ATM series [36] and its accompanying back-end processor, the 3848. This was the first commercial *hardware security module*, or HSM, as stand-alone cryptoprocessors have come to be known in the financial sector. HSMs controlled access to the PIN derivation keys, and also kept PINs secret in transit through the network. HSMs are still used during all stages of PIN management, including *acquisition* of PINs at the ATMs (the keypad is often integrated into a low-cost HSM based on the Dallas DS5002 microcontroller); *verification* at the card issuing bank, and also during *generation* processes, e.g. at PIN mailing sites.

The HSM's API is designed to allow legitimate operations on PINs and PIN derivation keys, for instance veri-

fying a PIN against an encrypted trial PIN from an ATM, or sending a clear PIN to a special printer to be sent to the customer. It must however prevent abuse by malicious employees; for instance it must not allow the wholesale discovery of a PIN for every bank account.

### B. Electronic Payment Schemes

HSM-based cryptography is spreading from ATM networks to general debit and credit card processing. Electronic payment systems use it to secure communications between banks and merchants, and to store verification keys to authenticate cards presented at point of sale terminals. Merchant devices may contain low-cost cryptoprocessors; supermarkets and chain stores may use full-blown HSMs as concentrators for till networks. HSMs are also an integral part of the back-end systems at banks which process these transactions, preventing insiders from exploiting their positions. The new EMV standard [33] for payment adds a third location for a cryptoprocessor – on the customer's card. These cards may store customer PINs, and allow end-to-end security for communication between the smartcard and the issuing bank.

Internet banking and payment have brought a range of concerns, many of which can be assuaged by using HSMs – which may in turn help home banking become ubiquitous. The challenge is to establish an island of trust in the user's home: some banks await generic "trusted computing" for PCs (fitting every computer with a cryptoprocessor), while others have issued stand-alone tamper-resistant authorisation devices (such as the RSA SecurID) that enable the use to enter a time-dependent password, or answer a random challenge, based on a key in the device. In effect, these devices export a security API to the PC through the user.

Finally, futuristic online electronic payment schemes such as "digital cash" have been tried out. Trusted third parties mint electronic tokens of some form, which can circulate from one user to another. The hard problem here is to prevent a user spending the same electronic coin twice. The issuers can use a combination of tamper-resistant devices and clever cryptography to deal with this. So long as the payment device remains tamper-proof, double-spending is prevented; and should a customer ever manage to defeat the tamper-resistance and spend a coin twice, the act of double-spending will automatically reveal his identity to the issuer [25], [23].

### C. Prepayment Electricity Meters

HSMs are a critical part of the prepayment electricity meter systems used to sell electric power to students in halls of residence, to the third-world poor, and to poor customers in rich countries [6]. They are typical of the many systems that once used coin-operated vending, but have now switched to tokens such as magnetic cards or smartcards. The principle of operation is simple: the meter will supply a certain quantity of energy on receipt of an encrypted instruction – a 'credit token', then interrupt the supply. These credit tokens are created in a token vending machine, which contains an HSM that knows the secret key in each local meter. The HSM is designed to limit the loss if a vending machine is stolen or misused; this enables the supplier to entrust vending machines to marginal economic players ranging from student unions to third-world village stores.

The HSM inside the vending machine must be tamper-resistant, to protect the meter keys and the value counter. The value counter enforces a credit limit; after that much electricity has been sold, the machine stops working until it is reloaded. This requires an encrypted message from a device one step higher up the chain of control – and would typically be authorized by the distributor once they have been paid by the machine operator. If an attempt is made to tamper with the value counter, then the cryptographic keys should be erased so that the vending machine will no longer function at all. Without these controls, fraud would be much easier, and the theft of a vending machine might compel the distributor to re-key all the meters within its vend area. There are other security processors all the way up the value chain, and the one at the top – in the headquarters of the power company – may be controlling payments of billions of dollars a year.

### D. Trusted Computing

Trusted Computing (TC) is an umbrella term for new technologies designed to embed cryptoprocessors in current computing platforms, including PCs and PDAs. An industry consortium, the Trusted Computing Group (TCG), designed a special cryptoprocessor, the Trusted Platform Module (TPM), to serve in a wide range of roles, and in particular to build an island of trust within the desktop PC [70]. The first generation TPM was designed for key storage and passive measurement of the machine state, and has been mass-produced. Notably, it is deployed in IBM Thinkpad Laptops. A new TPM is currently being designed, which will serve as the base for more sophisticated TC projects using desktop computers, for instance to allow corporations to lock down employee workstations to a particular OS configuration. It will also play a part in Microsoft's plans for a trusted Windows operating system (formerly called Palladium/NGSCB) [53]. This TPM, in combination with a software micro-kernel which it validates, will form a virtual cryptoprocessor, which many different applications can use.

The key idea is that a TC machine will be able to certify to other TC machines that it is faithfully executing a particular program. This means that the TPM, in conjunction with the microkernel, can certify both a program and the platform on which it is executing. Together with the trusted operating system, it can prevent one program from interfering with another.

The major application of TC is Digital Rights Management: the control of the distribution and use of data. In this context, a TC machine can assure a content vendor that it is sending a song or movie to a true copy of a media player program, rather than to a hacked copy. The vendor gets better assurance that the song or movie will remain under its control, and is less likely to appear in an unpro-

tected form on file-sharing networks. Present DRM mechanisms are based on software obfuscation, and eventually get hacked; the promise of TC to the content industry is that cryptoprocessors will slow down this process. Digital media have made lossless copying possible, and some vendors believe that more effective technical DRM mechanisms are the best way to protect their revenue streams. They may also enable alternative marketing strategies, such as subscription services for listening to music. (IBM's Enhanced Media Management System has an optional secure hardware component [42] based around its 4758 cryptoprocessor, but most existing DRM solutions do not yet offer cryptoprocessor-based tamper-resistance.)

DRM is not just for entertainment media such as music and video, but can also be applied to electronic documents and email. TC technology could support control of information flow within organisations, preventing leaks and making theft of intellectual property more difficult. Microsoft Office 2003 has built-in Information Rights Management facilities integrated into the applications. These extend Microsoft's rights management architecture to documents. For example, it becomes possible to restrict a document to named machines, or to cause it to self-destruct after a fixed period, or to prevent it being printed. Documents are encrypted together with a set of usage rules, written in a rights expression language; these rules are at present enforced by obfuscated software, but may in future be enforced using TPM-based TC mechanisms.

The APIs of modern rights-management systems are becoming full-blown security APIs. A content provider may use a third-party encapsulation of their content within an obfuscated viewer application. The content provider writes a policy governing access to the content, and this is compiled into the viewer. The user then interacts through this API providing various 'unlock codes' or interacting with online components of the system to gain access to the content. If the policy is poorly designed, the user may through careful manipulation be able to gain better access than was intended, for longer time-periods, or on additional machines.

### *E. Public Key Cryptoprocessors*

The arrival of public-key technology spawned a range of new secure communications protocols. Secure Sockets Layer is widely used to secure traffic on the web. It protects sensitive web services such as online payments and electronic banking. Public keys embedded in internet browsers are used to authenticate a chain of certificates that attest to a relationship between a particular domain name and a public key used in the SSL protocol. The user relies on this certificate chain to be sure that she is communicating directly with the webserver of the site in question – a merchant or electronic banking service, for example.

Webservers supporting SSL perform a private-key exponentiation for every connection attempt, so are under considerable load from the computations. This drove the development of the latest generation of public key cryptoprocessors. They are used for SSL acceleration, and well as for the protection and management of keys and certificates associated with providing secure web services.

Cryptoprocessors at server farms may enforce a simple *non-export* policy on an SSL private key – it must never leave the device. This mitigates the risks associated with webserver compromise, whether via network hacking or equipment theft. However, if the specific threats and challenges associated with the certificates can be identified, the HSM can process policy components, and genuinely assist in improving security. For instance, at *Certification Authorities* (CAs), where certificates are managed, HSMs may help enforce stringent policies on key usage: they can enforce dual control policies on the most valuable keys in a CA; they can help supervisors monitor the activities of large numbers of human operators efficiently; and they can keep signed audit trails of activities to allow retrospective monitoring of access.

Such infrastructures of keys and certifications are used for a range of other purposes: for login and for authentication in very large companies; for managing identity on the internet for SSL-enabled websites; and for managing software, by signing everything from web applets to components of operating systems.

### *F. Military Applications*

By World War 2, some military cipher machines had been made capture-resistant with thermite charges that destroyed them if they were opened by persons ignorant of the disarming mechanism. During the Cold War, great damage was caused by traitors, and in particular by the Walker family, who sold US Navy keys and cipher machine designs to the Soviets for over twenty years. This convinced the NSA that cipher machines should, as far as possible, also resist dishonest insiders. Modern military cipher machines not only use classified algorithms in tamper-resistant chips; they also use such chips as *crypto ignition keys* to transport initial key material. The goal is that the street value of these keys should be zero, as most enemies have no access to a machine in which to use them; and even for the few who do have such access, stolen keys should not be useful in deciphering any other traffic. Enforcing such a policy requires that the tamper-resistance, the crypto protocol design and the management procedures all work together well.

Another influential military application has been *nuclear command and control*. According to US military doctrine, a nuclear weapon may not in general detonate without *authorization*, *environment* and *intent*. 'Intent' means an unambiguous arming signal from the officer in charge of the weapon, typically carried over multiple wires with error-control coding. 'Environment' means a condition that is easy to measure yet hard to forge, such as the period of zero gravity experienced by an air-drop bomb on release and by an ICBM in its sub-orbital phase. 'Authorization' means a coded signal from the national command authority – the President and his lawful successors in office. This system was introduced following the Cuban missile crisis, to minimize the probability of a nuclear war starting by

accident or mischance. The authorization codes in particular have some interesting and complex requirements, described in [3]. Tamper-resistance mechanisms are embedded in weapons in order to prevent a stolen weapon being exploded, or being dismantled to reveal an authorization code with which a second stolen weapon could be armed.

### G. Specialist Applications

*Bills of Lading* – the international trade and payments system used documents called bills of lading to represent cargoes in transit. Thus the owner of a shipment of oil in transit across the Atlantic may sell it by endorsing the bill of lading to the new owner. Implementing an electronic version of bills of lading presented a number of interesting challenges, including finding some way to prevent a crook selling the same goods twice. Given that an oil cargo may be worth $100m, and it may take weeks from sale until delivery, it was vital to provide robust mechanisms to prevent such a fraud. The Bolero system uses two independent mechanisms. Electronic bills of lading can only be endorsed from one owner to another using keys kept in hardware security modules that closely control their use; there is also a central database of extant bills maintained on a server operated by a trusted third party [43]. In order to sell an oil cargo twice, a crook would have to subvert this third party and also defeat the HSM.

*Key-Loading Sites* – Applications which deploy thousands of cryptoprocessors, such as pay-TV smartcards and prepayment token vending machines, require a trustworthy means of initialising them before delivery. Generic manufacturing processes are often followed by a key-loading phase, which may happen at the premises of a bank or a specialist security contractor. This operator may have a hierarchy of HSMs, or in the case of smartcard-to-smartcard protocols, a large rack of smartcards in readers, to supervise the initialisation and personalisation of new devices. Some interesting failures have occurred where the system operator has tried to save money on this operation. In one case, a pay-TV operator used PC software to personalise smartcards; this software entered the public domain when one of the PCs from the control centre was sold second-hand and its disk contents were undeleted by a curious purchaser. In other cases, weak private keys have been issued to bank smartcards by badly-designed card personalisation systems.

## III. Taxonomy of Cryptoprocessors and Attacks

Early military cipher machines may be seen at the NSA Museum, Fort George G. Meade, Maryland; the tamper-resistance mechanisms extend to protective detonation mechanisms that destroy the device if it is improperly opened. The earliest civilian cryptoprocessor known to us is the IBM 3848, an encryption device that functioned as a mainframe peripheral. It was contained in a steel cabinet, with switches that zeroised the memory containing cryptographic keys whenever the cabinet was opened.

Top-of-the-range cryptoprocessors nowadays have generally followed the second line of development. An example is the IBM 4758 (figure 1), the 3848's descendant, which has a cryptographic engine surrounded by a multilayer tamper-sensing mesh (figure 2). This is constantly monitored by the engine, which erases its key material and renders itself inoperable if a tampering attempt is detected. Rather than occupying a whole cabinet in a machine room, the 4758 comes as a standard PCI card for mounting in a server. A rich literature documents the design, development and validation of the 4758 [31], [38], [63], [64], [66], [67], and its security architecture [32].

Fig. 1. IBM 4758-001

At the low-cost end of the market, cryptoprocessors are often implemented in microcontrollers. Many engineers are familiar with these cheap, standard components. The cheapest microcontrollers cannot perform public-key cryptography in a reasonable time, but no matter: many applications such as remote key entry use shared-key cryptographic algorithms such as triple-DES and AES, whose computational cost is low. A more serious problem is that the read-protect mechanisms in low-cost microcontrollers are not really designed to withstand skilled and determined attack.

A middle market has therefore emerged of single-chip products that have been hardened in various ways against attack. These products include smartcards, and the TPM chips specified by the Trusted Computing Group for use in PCs. Before deciding whether an application can use a low-cost microcontroller, or needs a smartcard-grade component or even a high-end tamper-responding device, it is necessary to understand something about the technology of attacks and defences. We will discuss the attacks on microcontrollers, and the measures that can be adopted to thwart them, in more detail in the next section.

When analyzing the security of a cryptoprocessor, it can be useful to perform a systematic review of the *attack surface* – the set of physical, electrical and logical interfaces that are exposed to a potential opponent. This leads us to divide attacks into four classes.

*Invasive attacks* involve direct electrical access to the internal components of the cryptoprocessor. For example, the attacker may open a hole in the passivation layer of a microcontroller chip and place a microprobing needle on a bus line in order to capture a signal.

*Semi-invasive attacks* involve access to the device, but without damaging the passivation layer of the chip or making electrical contact other than with the authorised interface. For example, the attacker may use a laser beam to ionise a transistor and thus change the state of the flip-flop that holds the device's protection state.
*Local non-invasive attacks* involve close observation or manipulation of the device's operation. An example is *power analysis*: measuring the current drawn by the processor with high precision, and correlating this with the computations being performed by the device in order to deduce the value of cryptographic keys.
*Remote attacks* involve observation or manipulation of the device's normal input and output. Examples include timing analysis, cryptanalysis, protocol analysis and attacks on application programming interfaces.

Each of these types of attack may also be either *active* or *passive*. In the latter the attacker works with the device as it is operated normally, while in the former the attacker may manipulate the device, its inputs or its environment so as to induce abnormal operation. An active invasive attack may involve injecting signals physically into the device using probing needles; an active semi-invasive attack may use a laser or strong electromagnetic pulse to cause aberrant behaviour of some internal component of the chip; an active local noninvasive attack may manipulate power line voltages or clock frequencies to cause partial failure of the device under test; while an active remote noninvasive attacker might feed a cryptoprocessor with carefully-chosen sequences of transactions in order to cause behaviour that was not anticipated by the device's designer.

Fig. 2. An IBM 4758-001 part potted in urethane, showing membrane and interior (courtesy F. Stajano)

The high-level summary of these attacks is that, by spending more money on a better cryptoprocessor, you can greatly diminish and perhaps even eliminate the first three classes. All of these are local, in that the opponent needs to obtain unsupervised access to the device; so in an application where the cryptoprocessor can be physically guarded, or perhaps where its owner's incentive is to protect its secrets rather than try to extract them, you may be able to use a cheaper cryptoprocessor or even ignore these attacks completely. But many attacks in the fourth, remote, class are independent of the quality of the cryptoprocessor hardware. It does not matter how much you spend on device-level protection, if the transaction set which you implement on it can be manipulated in such a way as to break your security policy (Smith – a designer of the IBM 4758 – reflects upon the disparity between its state-of-the-art hardware and firmware and the financial API it usually implements, which has turned out to be the weakest link, in [68]).

In the next section we will describe the local attacks to which budget cryptoprocessors may be vulnerable – the invasive, semi-invasive and local non-invasive attacks. The following section will then focus on remote attacks, and in particular on API attacks.

## IV. Local Attacks

Fifteen years ago, devices such as microcontrollers and smartcards offered little resistance to a capable motivated opponent. Protection typically consisted of a read-only bit that was set after programming; this bit could often be reset by simple tricks such as *glitching* [46] – inserting a transient into the power supply, perhaps by reducing the supply voltage to zero for a few microseconds – or illuminating it with UV light. Even so, microcontrollers gave better protection than circuit boards with discrete components, whose interconnects could be observed directly with an oscilloscope. At that time, there were few valuable applications and thus few serious attackers. As late as 1994, a senior industry figure explained at the Cardis conference that there was simply no demand for better security.

That changed once smartcards started to host applications such as pay-TV. Various people set out to reverse-engineer pay-TV smartcards: some were pirates who wanted to forge cards, while others wanted to put key material on the Internet so that people could watch for free. (This sometimes involved idealism, but in at least one case a pay-TV operator was sued for hacking a rival's card and anonymously publishing their key [24].) In addition, microcontrollers were used in accessory-control applications such as games cartridges, where the business model involved subsidising consoles from sales of software, and the security devices in the cartridges were there to ensure that accessory vendors paid the manufacturer a royalty. This created a strong business incentive for a vendor's competitors and aftermarket suppliers to reverse-engineer its security chips – which was legal in most relevant jurisdictions.

This triggered an arms race between attack and defence. In the mid-90s, attackers invested in invasive attack methods, using probing stations; then in the late 1990s there was a wave of attacks using non-invasive techniques such as power analysis; and in the early 2000s, a number of semi-invasive attacks were developed, of which the best known involve optical probing. Chipmakers have responded by developing new families of microcontrollers and smartcards with a number of defensive features.

### A. Invasive attacks

The earliest tamper-resistant processors – the hardware security modules used by banks from the 1980s – were very vulnerable to physical attack. The protection mechanisms relied on steel boxes with lid switches and, sometimes, fur-

ther sensors such as photodiodes and seismometers. An attacker familiar with these mechanisms could drill his way in. A worse problem was that maintenance visits were needed every year or two to replace the battery that preserved cryptographic keys while the machine was powered off. It was trivial for a maintenance technician to disable the tamper-responding circuit on one visit, and then extract the keys on his next visit. Modern HSMs therefore use more complex designs. The IBM 4758, for example, has its electronic components encased in a tamper-sensing membrane that is itself potted in a compound that's difficult to cut or drill cleanly, and the batteries are outside this security perimeter (see figures 1 and 2).

As well as tamper-sensing membranes, a number of other tricks are used: custom-pressure air gaps, stressed glass components, and even shaped charges in military devices which will destroy the memory chip yet are so small that they do not result in gas release from the package. Weingart surveys such anti-tamper technologies in [71]. Such designs are by now fairly mature, and there is a certification program (FIPS 140) which vendors can follow to assure their customers of a certain level of tamper-proofness. FIPS is discussed in section VII.

Of more interest are the technologies relating to making single-chip devices resistant to invasive attack. Probing chip internals directly had always been an option for semiconductor makers. By the mid-1990s, the availability of second-hand semiconductor test equipment such as manual probing stations made invasive attacks practical for a much wider class of attackers.

A typical probing station consists of a microscope with an objective working distance of several millimeters, mounted on a low-vibration platform containing micromanipulators that can be used to place probes on to the device under test. The microscope will typically also be fitted with a laser, with which small holes can be drilled in the chip's passivation layer. These holes allow electrical contact by the probes, and indeed stabilise them in position. The modus operandi is to probe the device's internal bus, so that both program and data can be read out. Many neat tricks have been developed by practitioners; for example, placing a grounded probe on the clock line to the instruction latch, so that the same instruction is executed repeatedly and the program counter is thus incremented continuously, causing the entire memory contents to become available in sequence on the bus. The standard reference on microprobing is [46].

Since the late 1990s, smartcard vendors have made invasive attacks very much more difficult. First, high-grade cards typically have a sensor mesh implemented in the top metal layer (figure 3), consisting of a serpentine pattern of sensor, ground and power lines: if the sensor line is broken, or shorted to ground or power, the device self-destructs.

Second, technological progress is steadily increasing the cost of an attack. Ten years ago it was possible to use a laser cutter and a simple probing station to get access to any point on the chip surface, and it was possible to navigate the chip's surface visually. Modern deep-submicron

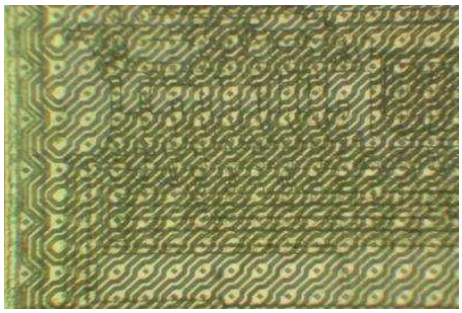

Fig. 3. Top metal sensor mesh on ST16 smartcard

semiconductor chips call for more sophisticated and expensive probing technologies, which exclude most potential attackers.

For example, the structure of the Microchip PIC16F877 microcontroller can be easily observed and reverse engineered under a microscope (figure 4). The second metal layer and polysilicon layer can still be seen, although buried under the top metal layer. This is possible because each subsequent layer in the fabrication process follows the shape of the previous layer. Under a microscope, the observer sees not only the highest layer but also edges that reveal the structure of the deeper layers.

In 0.5 $\mu$m and smaller technologies, as in the later Microchip PIC16F877A microcontroller (figure 5), each layer but the last one is planarised using chemical-mechanical polishing before applying the next layer. As a result, the top metal layer does not show any trace of the deeper layers. The only way to reveal the structure of the deeper layers is by removing the top metal layers either mechanically or chemically.

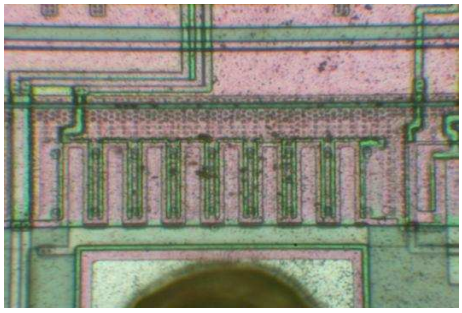

Fig. 4. PIC16F877 built with old technology (0.9-micron)

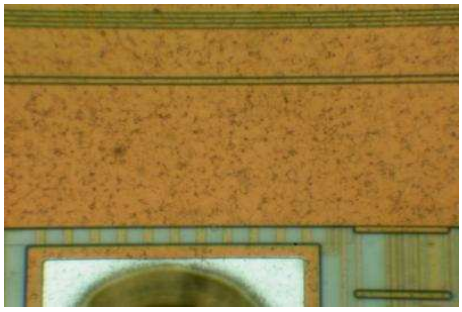

Fig. 5. PIC16F877A built with new CMP technology (0.5-micron)

Even once the chip is sufficiently well-understood that the attacker knows where to tap, making a useful electrical contact is not trivial. A Focussed Ion Beam (FIB) workstation may be needed to drill down to the appropriate component and bring out a contact to a bonding pad. Even so, the added capacitative load may be such that the modification causes the chip to fail. There are also defences that can be used – refractory passivation layers, for example, that are slow to penetrate with the FIB or cause excessive charge buildup, which can be used to trigger an alarm.

### *B. Local non-invasive attacks*

In the mid-1990s, the cryptoprocessor industry received a rude shock when Kocher revealed that many products could be easily compromised by timing attacks. An operation such as the calculation of a digital signature involves many modular multiplications, and the time taken by an efficient implementation depends strongly on the input values. With suitable analysis, the time taken by a cryptoprocessor to perform such operations leaks the value of its private key [44].

The industry had hardly had time to digest this news and devise countermeasures before Kocher delivered an even nastier shock, with the publication of *Differential Power analysis* (DPA) in 1998 [45]. DPA works by analysing the current drawn by the processor, which varies depending on the data being processed. In general, the field of *emission security* (EMSEC) is about ways in which electronic devices and systems can be attacked by observing or inducing information leakage by electromagnetic means, and about defences against such attacks.

Emission security first came to the attention of the military when crosstalk between telegraph circuits was discovered during a British army expedition to the Nile and Suakin in 1884–5. It became a serious issue during World War 1, when it was found that the earth-return telephones used at the front could be eavesdropped by the other side at a distance of hundreds of yards. Nowadays, substantial sums are spent on 'tempest' protection of military electronic equipment – that is, ensuring that it does not leak useful information by means of stray RF, power-line, or other emissions. A survey of emission security can be found in [3].

Electromagnetic attacks can be either active or passive. We already mentioned one active attack – the glitch attacks whereby microcontrollers' security bits could often be reset by inserting a suitable transient into the power line. Clock glitches were also effective with some processors: doubling the clock frequency for a few microseconds would cause some, but not all, instructions to fail. Thus it could be possible to modify the device's control flow – for example, by stepping over the branch instruction following a failed password check.

Differential power analysis is, by contrast, a passive attack. The idea is to measure the power consumption of a smartcard chip while it does a number of cryptographic computations (typically several hundred) with different data. We then guess the value of some bit in a cryptographic computation – perhaps a bit of the key, or an intermediate value in the encryption – and then sort our power traces into two piles, depending on whether our target bit combined with the input or output data in question would have activated some circuit in the processor or not. For example, we might suspect that the carry function used a measurable amount of power, and sort our traces into two piles depending on whether, after the first round of a cipher, our guessed key value would, when combined with the observed data, cause the carry bit to be set. We then check our guess by looking to see whether the two piles are statistically different.

This simple signal-processing trick turned out to be devastatingly effective, in that key material could be rapidly extracted from most of the smartcards then on the market, and by attackers with no detailed knowledge of how the cryptography was implemented. Previously, non-invasive attacks had either been discovered by luck, as with glitch attacks, or following expensive reverse engineering of the chip, typically involving a full-scale invasive attack. However, DPA empowered a moderately clueful graduate student to build a device to extract crypto keys from smartcards. The implications were grim. For example, if smartcards were widely deployed in point-of-sale applications, one might expect the Mafia to build and deploy key-stealing terminals. Now, false-terminal attacks have been a chronic problem for the magnetic-strip bank-card industry since the early 1990s, and one of the strong selling points of smartcards had been the prospect of eliminating them. Now, it seemed, smartcards were not all that much harder to copy than the magnetic-strip cards they were intended to replace.

A lot of effort was invested around the turn of the century in devising defenses against power analysis. This is not entirely straightforward, as a measure that makes power analysis harder may make timing analysis easier: there are several variants on the non-invasive attack theme that must be dealt with simultaneously. The possible defenses include the following.

*Hardware measures:* Noise generators have been tried but tend to just force the attacker to collect more samples. Randomisation is better: for example, insert a no operation instruction (NOP) at random in the instruction stream with probability 1 in 64. Another possibility is balanced logic: we discuss this in the next section. The current research frontier is design-time validation: developing tools that enable a chip maker to simulate power analysis and other emsec attacks early in the design stage, so that changes can be made before the chip is fabricated.

*Crypto library measures:* Much effort has been put into devising randomized ways of implementing crypto algorithms. For example, if one is performing an RSA decryption, $m = c^d \pmod N$ where $d$ is the secret decryption exponent, once can split $d$ into two components, $d_1 + d_2 = d$ and compute $m = c^{d_1} c^{d_2} \pmod N$. That at least is the theory; in practice one would probably have to do a bit more than this, and randomising ciphers that have less mathematical structure than RSA can be hard work.

*Protocol measures:* If one has the luxury of designing the entire system, rather than having to build more secure components for a system whose design is already fixed, then it can be extremely effective to randomise all the encryption operations and protocol messages, so that the opponent never knows both the plaintext and the ciphertext of any cipher operation. If keys can be constantly changed too, then this makes any form of cryptanalysis – including power analysis – harder.

However, just as these defences push up the bar, advances in attack technology let the opponent reach higher. The use of randomised NOP instructions, for example, may be countered if the attacker can write software to align the traces he collects; correlation analysis can then be performed as before. Perhaps the most radical advance, though, is Differential Electromagnetic Analysis, by Samyde and Quisquater [58]. In this, a small coil or other magnetic sensor is brought to the very surface of the chip itself. The attacker sees not just the gross power signal, which is a composite of the power drawn by all of the circuits in the chip, but a local signal correlated with the power drawn by some target circuit. This can yield significantly more information. It also brings us to the topic of semi-invasive attacks.

### C. Semi-invasive attacks

Between the invasive and non-invasive attacks lies a third class of local attacks, which we christened *semi-invasive attacks* [60]. These are attacks that involve access to the chip surface, but which do not require penetration of the passivation layer or direct electrical contact to the chip internals.

The earliest semi-invasive attacks involved the use of UV light to reset the protection bit on microcontrollers, so that memory contents could be read out [8]. Another early development was the LIVA / LECIVA semiconductor-testing technology developed by Sandia, in which illumination of non-conducting CMOS transistors causes measurable current leakage [1]. Boneh and others also speculated that revealing faults might be induced by subjecting smartcards to pulses of microwave radiation [22]. No-one appears to have made this work, but together with the glitching attacks reported in [7] it helped motivate researchers to spend some time thinking about *fault analysis* – how suitably engineered faults could cause interesting failures of cryptographic mechanisms. For example, one can extract crypto key material by successively setting key bits to 0 [14]; and in concrete systems, one can often bypass crypto altogether, for example by interfering with the processor's control flow so that cryptographic computations are not performed, or their results ignored [7].

Practical semi-invasive attacks emerged a few years ago when we pioneered optical probing techniques to inject security faults into digital circuits. The idea is that illuminating a target transistor causes it to conduct, thereby inducing a transient fault. Such an attack can be carried out with simple, low-cost equipment: we demonstrated it by mounting a photographer's flash gun, bought second-hand for $30, on the camera port of our microscope (figure 6). Using this, we were able to set or reset any individual bit of SRAM in a microcontroller [60].

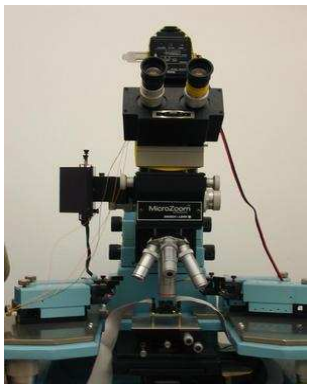


Fig. 6. Low-cost probing workstation and photoflash

Much better results can be obtained using laser probing equipment. In addition to setting RAM contents to desired values, laser probing can be adapted to read out other memory technologies, such as Flash and EEPROM, and to interfere with control logic directly. Detailed information on semi-invasive attacks is presented in [62].

As well as performing fault analysis, laser probes can be used for passive read-out of memory contents. Here, the trick is to illuminate the memory cell transistors one at a time. The resulting ionization causes a measurable increase in leakage current if the transistor is switched off. Such attacks do not necessarily require expensive gas lasers, but may be carried out using adapted laser pointers [59].

Defense mechanisms include defensive programming techniques to detect active attacks and perform some suitable alarm function, such as erasing key material. Opaque top-metal grids and shields also raise the bar for attackers in time and equipment cost – the attacker may now have to go through the rear of the chip, which will typically involve ion-etching equipment to thin the device and an infra-red laser to penetrate the silicon substrate.

No single low-cost defence technology can protect a smartcard against attack. Existing high-end techniques, such as top-layer metal shielding, bus encryption and randomised cryptography, make attacks more complicated; but a sophisticated attacker can defeat metal shielding by using infrared light from the rear side or X rays, while bus encryption can be defeated by attacking registers directly. The effective level of protection depends on how well the defense mechanisms work together. Often they do not, as the hardware, the operating system, the crypto library and the application are created by different teams, who minimize information sharing on the grounds of 'security'.

### D. How smartcards differ from commodity microcontrollers

So, with a growing demand for better protection, chip manufacturers started to come up with smartcard products that are specifically designed to protect against most of the known attacks. This has resulted in a sharp bifurcation in the market, between the low-cost microcontrollers

offering rudimentary protection and higher-cost smartcards in which serious security effort has been invested.

Their protection features now typically include internal voltage sensors to protect against under- and over-voltages used in power glitch attacks; clock frequency sensors to prevent attackers slowing down the clock frequency for static analysis and also from raising it for clock-glitch attacks; top-metal sensor meshes mentioned above; internal bus hardware encryption to make data analysis more difficult; and light sensors to prevent an opened chip from functioning. Software access to internal memory is often restricted by passwords, so that simple hacks to read out all the memory on the bus are no longer available.

Another improvement worth mentioning is a move in the late 1990s from the standard building-block structures as in figure 7, where one can identify under an optical microscope the chip components such as the CPU instruction decoder, register file, ALU and I/O circuits, to a randomised ASIC-like logic design. This is called 'glue logic' and it is now widely used in smartcards (figure 8). Glue logic makes it virtually impossible to find signals manually for physical probing. An attacker needs automated tools such as the automated laser probe described above; in extremis, she may have to reverse engineer the entire circuit from micrographs.

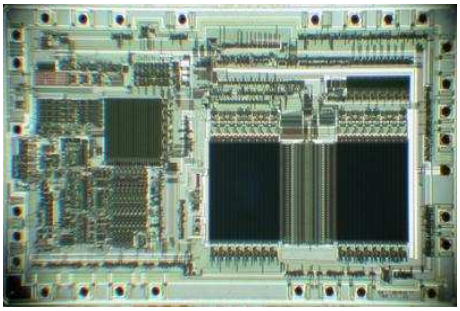

Fig. 7. MC68HC705PA microcontroller with clearly distinguishable blocks

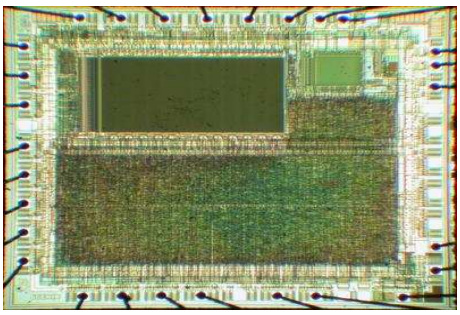

Fig. 8. SX28 microcontroller with 'glue logic' design

Attacking smartcards, especially recently designed ones, is now an expensive and time-consuming task. Only well-equipped laboratories with highly-qualified engineers generally succeed. However, progress in attack technology constantly forces chipmakers to improve their products. For example, the recently discovered optical fault-injection attacks [60] revealed many problems in some designs as they let an attacker change the state of any transistor on the chip. An attacker can thus set up a chip on a motorised stage and use fault injection to perform an automated search for components such as the register file which can then be studied in detail and attacked. Advances in nanotechnology are bringing still more useful tools. An example is scanning capacitance microscopy, which lets us measure the threshold level of each individual transistor, and thus determine the contents of memory technologies previously believed to be secure against probing, such as voltage threshold ROM.

An interesting possible future defense is the use of error-detecting logic. In the EU-funded G3Card project, we built prototype processors using self-timed dual-rail logic, with an inbuilt alarm function. Each bit was signalled on two wires, with (0,1) signaling zero, (1,0) signaling one and (1,1) signaling 'alarm'. Circuit components were designed so that any single-transistor failure would result in alarm propagation, thus making optical fault induction attacks much harder, and also so that the power consumed by a computation was independent of the logical values of the input and output. In a collaboration involving three universities, two smartcard OEMs and a specialist crypto company, we built three prototype processors and demonstrated the feasibility of the technology. However it is expensive, taking about three times as many gates as conventional logic designs [11], [12].

The full-size hardware security module still has several critical advantages over the smartcard. First, it can contain large capacitors to filter data-dependent signals thoroughly from its external connections. Second, it is big enough for a tamper-sensing barrier. Third, and perhaps most important, it has an internal power supply. This enables it to monitor the tamper-sensing barrier constantly and destroy key material if a penetration attempt is detected. The smartcard, by contrast, is unpowered most of the time, so the attacker can work happily away on one location on the chip without causing harm to key material stored at another location. There have been some ingenious attempts to solve this problem – for example, by coating the chip in a chaotic matrix of magnetic fibres, whose emergent characteristics store a master key with which the memory contents are encrypted. However, as far as we are aware, no-one has managed to bring such an idea to market as a working product.

## V. Remote attacks

The last type of attack is the remote attack. There are several kinds of attack that are independent of the distance between the attacker and the cryptoprocessor; the attacker can be on the other side of the world. She merely needs access to the encrypted traffic. Again, these attacks can be passive or active; in the former case, the encrypted transaction stream is observed and analysed, while in the latter the attacker can insert transactions and observe the responses.

Two well-known types of remote attack are cryptanalysis

and protocol analysis. In the former, the attacker exploits design flaws in crypto primitives such as encryption algorithms, hash functions and digital signature schemes; in the latter, she uses flaws in the protocols in which these primitives are used. There is a large literature on both types of analysis; see for example [3]. They are not specific to cryptographic processors and we will not consider them further here.

There is however another kind of remote attack that is specific to cryptographic processors. This is *API analysis*. This attack technology was developed only in the last few years, and most cryptographic processors have turned out to have at least one API vulnerability. Although patches are issued by manufacturers, API flaws continue to be found.

We will illustrate API analysis with reference to the HSMs used to protect PINs for bank ATMs. API vulnerabilities are found in other HSM applications too, such as token, vending, but PIN security provides a good example of the ideas that underlie API attacks.

### A. What is a Security API?

A security API is "an Application Programming Interface that uses cryptography to enforce a security policy on the interactions between two entities" – essentially the top-level software component of a cryptoprocessor, which governs its interaction with the outside world. It differs from a *cryptographic API* in that as well as providing cryptographic services (such as DES, RSA, PKCS padding), it also enforces *policy* on the interactions. The crypto API designer relies on the attached host to make policy decisions about key use, while the security API designer is more pessimistic, and makes few assumptions about the outside world. He would usually assume that the host with which the cryptoprocessor interacts is under the control of the opponent.

### B. Early API Attacks

Longley and Rigby made early analyses of "key management schemes" in the late 80s, [48], [57], [49], [50] working with the HSM vendor Eracom. They described analysis techniques and failures of the devices of their time, which we would nowadays call cryptoprocessors. These devices enforced very little policy on the usage of keys, save that they should not leave the box. In particular, Longley and Rigby identified the importance of separating different key types. The products and APIs they analysed were not specified.

Anderson later brought the design and use of cryptoprocessors in banking into the open in "Why Cryptosystems Fail" [4]. He focused on the known failure modes of ATM banking systems, including several procedural and technical failures in the use of HSMs. A cryptographic binding error was typical of the failures described:

*"One large UK bank even wrote the encrypted PIN to the card strip. It took the criminal fraternity fifteen years to figure out that you could change the account number on your own card's magnetic strip to that of your target, and then use it with your own PIN to loot his account."*

That paper stopped short of describing what we would nowadays call an API attack. Another paper, "Low Cost Attacks on Tamper Resistant Devices" [7], describes the deliberate addition of a dangerous transaction to a security API.

Common banking practice was to calculate customer PINs by encrypting the customer's Primary Account Number (PAN) with a secret key, then converting the resulting ciphertext into a four digit number. If a customer changed her PIN, the bank stored an offset between old and new PINs in their master database. For example, an original PIN of `3566` changed to `3690`, yielded an offset `0134`, which would be subtracted during verification.

One bank wished to restructure their customer PANs. Unfortunately, changing the PAN would change the original PIN issued to customers, and the bank did not wish to force all its customers to accept new PINs. The bank commissioned a transaction to adjust all the stored offsets in the database, so that they could change PANs without forcing PIN re-issue. The designers produced a transaction of the following form, warning that it was dangerous and should only be used to perform the batch conversion, then removed from the API. $U \longrightarrow C$ represents the user's command to the cryptoprocessor, and $C \longrightarrow U$ the response.

$U \longrightarrow C$ : `old PAN, new PAN, offset`
$C \longrightarrow U$ : `new offset`

The warnings were forgotten, and the transaction was never disabled, and year or so later, a programmer spotted how this transaction might be abused. If he fed in his own account number as the `new PAN`, the command would duly calculate and return the difference between any customer's issued PIN and his own original PIN! At the time, this was characterised as a protocol failure.

In "The Correctness of Crypto Transaction Sets" [2] (2000), Anderson remarked that while such failures pertained to a *single* bad transaction, they raised the harder question: "So how can you be sure that there isn't some chain of 17 transactions which will leak a clear key?". The idea of an API attack was born as ***an unexpected sequence of transactions which would trick a security module into revealing a secret in a manner contrary to the device's security policy***.

### C. The Visa Security Module

Shortly afterwards, an attack on the 'VISA Security Module' (VSM) was discovered by Anderson. The VSM was one of the earliest financial HSM designs, used to facilitate PIN processing in networks of bank ATMs. The VSM had to inter-operate with offline ATMs (due to poor telephone network reliability in some countries), so when banks set up new ATMs, they needed a way to securely transfer the *PIN derivation keys* from the VSM to the ATMs. The VSM used a system of dual control: two service engineers would each take one *component* of a master key to the ATM, and enter it in. Once both components were

entered, the ATM could combine the components using the XOR function. The resulting 'Terminal Master Key' (TMK) would be shared with the VSM and could be used for communicating all the other keys. A transaction was first run twice at the VSM to generate the components:

**(Generate Component) x2**

$$\begin{array}{lcll} C & \longrightarrow & Printer & : TMK1 \\ C & \longrightarrow & U & : \{TMK1\}_{Km} \end{array}$$

$$\begin{array}{lcll} C & \longrightarrow & Printer & : TMK2 \\ C & \longrightarrow & U & : \{TMK2\}_{Km} \end{array}$$

The VSM only had very limited internal storage, yet there might be many different ATMs it needed to hold keys for. The paradigm of working with encrypted keys evolved: instead of keeping keys internally, the VSM only held a few master keys $(Km_1, ..., Km_n)$, and other keys were passed in as arguments to each transaction encrypted under one of these master keys (e.g. $K1_{Km}$). So, in response to the above transaction, the VSM returned an *encrypted copy* of the component to the host computer, encrypted under its master key, $Km$ (and also printed a clear copy onto a special sealed mailer for the service engineer). The VSM recreated the same key as the ATM using a command to combine two encrypted components together, shown in figure 9.

***Normal operation***

$$\begin{array}{lcll} U & \longrightarrow & C & : \{TMK1\}_{Km}, \{TMK2\}_{Km} \\ C & \longrightarrow & U & : \{TMK1 \oplus TMK2\}_{Km} \end{array}$$

***The attack***

$$\begin{array}{lcll} U & \longrightarrow & C & : \{TMK1\}_{Km}, \{TMK1\}_{Km} \\ C & \longrightarrow & U & : \{TMK1 \oplus TMK1\}_{Km} \end{array}$$

$$TMK1 \oplus TMK1 = 0$$

Fig. 9. The XOR to Null Key Attack

The crucial observation is this: if the same component is fed in twice, then thanks to the use of XOR, a known key (of binary zeroes) will result. This known key could then be used to export the PIN Derivation Key (PDK) in the clear. Bond later examined the VSM transaction set and found that there were even more vulnerabilities, due to inadequate key separation. The *Terminal Master Keys* used to send other keys to ATMs, and the *PIN Derivation Keys* used to calculate customer PINs were considered by the VSM to be keys of the same type (which it expressed by storing them encrypted with the same master key, here called $Km$). Two example transactions using these keys are shown below. $PDK1$ is a PIN derivation key, and $TMK1$ is a terminal master key.

The first transaction encrypts a customer PAN with the PIN derivation key, but sends the PIN to a secure printer (for subsequent mailing to the customer); the second encrypts the PDK under a TMK belonging to an ATM. Though they perform quite different functions which are not connected, their inputs arguments are encrypted with the same master key.

**(Print PIN Mailer)**

$$\begin{array}{lcll} U & \longrightarrow & C & : PAN, \{PDK1\}_{Km} \\ C & \longrightarrow & Printer & : \{PAN\}_{PDK1} \end{array}$$

**(Send PDK to ATM)**

$$\begin{array}{ll} U \longrightarrow C: & \{PDK1\}_{Km}, \{TMK1\}_{Km} \\ C \longrightarrow U: & \{PDK1\}_{TMK1} \end{array}$$

However, the designers *did* recognise a clear difference between 'Terminal Communications' keys (TCs) and PIN derivation keys or TMKs. $TC1$ is a terminal communications key, and $Km2$ is a second master key that was used to encrypt keys of this type, keeping them separate from the rest. They were kept separate because terminal communications keys were not considered to be as valuable as PIN derivation keys – and there needed to be a transaction to enter a chosen TC key.

**(Enter clear TC Key)**

$$\begin{array}{ll} U \longrightarrow C: & TC1 \\ C \longrightarrow U: & \{TC1\}_{Km2} \end{array}$$

TCs needed to be communicated to ATMs in the same way as PIN derivation keys, so there was a command that worked in a very similar way, encrypting the chosen TC under a chosen TMK corresponding to a particular ATM.

**(Send TC Key to ATM)**

$$\begin{array}{ll} U \longrightarrow C: & \{TC1\}_{Km2}, \{TMK1\}_{Km} \\ C \longrightarrow U: & \{TC1\}_{TMK1} \end{array}$$

However, Bond spotted that when these two transactions were used together, given the lack of separation between PIN derivation keys and TMKs, there was a simple attack. It was to enter in a customer PAN, claiming it to be a TC key, and substitute a PIN derivation key for a TMK in the "send to ATM" transaction.

**(Enter clear TC Key)**

$$\begin{array}{l} U \longrightarrow C: PAN \\ C \longrightarrow U: \{PAN\}_{Km2} \end{array}$$

**(Send TC Key to ATM)**

$$\begin{array}{l} U \longrightarrow C: \{PAN\}_{Km2}, \{PDK1\}_{Km} \\ C \longrightarrow U: \{PAN\}_{PDK1} \end{array}$$

Of course, $\{PAN\}_{PDK1}$ is simply the customer's PIN. Just like the 'XOR to Null Key Attack', this vulnerability had gone unnoticed for over a decade. How many more attacks were waiting to be found?

### *D. Cryptographic Attacks*

Further systematic exploration of the VSM API, and IBM's Common Cryptographic Architecture (CCA) for the IBM 4758 yielded new attack techniques (an overview of the CCA is in [51], [52]; the detailed reference is [37]). Bond

observed that both the CCA and the VSM had transactions to generate 'check values' for keys – a key identifier calculated by encrypting a fixed string under the key. These check values were used to detect typing mistakes during key entry and for debugging purposes. The typical check value implementation was to encrypt a block of binary zeroes.

**(Generate Check Value)**

$$\begin{aligned} U &\longrightarrow C: \quad \{TMK1\}_{Km} \\ C &\longrightarrow U: \quad \{0000000000000000\}_{TMK1} \end{aligned}$$

Due to the commonly prevailing external storage design, a user could generate an almost unlimited number of conventional keys of a particular type. The designers were aware check values could be used as known plaintext for a brute force search to find a key, but considered search of the 56-bit DES key space too expensive. Many estimates have been made of the cost of DES cracking [54]. However, due to the HSM architecture, any one of a large set of keys would suffice to use as a stepping stone to extract valuable keys. A parallel search for keys was thus possible. The algorithm was as follows:
1. Generate a large number of terminal master keys, and collect the check value of each.
2. Store all the check values in a hash table
3. Repeatedly guess a key, encrypt the test pattern with it, and compare the resulting check value against all the stored check values by looking it up in the hash table.

With a $2^{56}$ keyspace, and $2^{16}$ target keys, a target key should be found with roughly $2^{56}/2^{16} = 2^{40}$ guesses. This is called the 'meet-in-the-middle' attack, with reference to the meeting of effort spent by the HSM generating keys and effort spent by the brute force search checking keys. Time-memory trade-offs such as this had already been described several decades ago, for example in an attack against 2DES proposed by Diffie and Hellman [30]; the idea of parallel search for multiple keys was also not new (Desmedt describes parallel keysearch in [29]). However, this was the first application to HSMs, and it was a practical use of a technique previously considered rather theoretical. We compromised many older HSM architectures with this technique; [20] and [2] have more details.

### *E. 3DES Key Binding Attack*

In the late 90s, once DES became too vulnerable to brute force, financial APIs started replacing it with triple-DES (3DES). IBM's CCA was extended to support two-key 3DES keys, but stored each half separately, encrypted under the master key. A different *variant* of the master key was used for the left and right halves – achieved by XORing constants representing the types *left* and *right* with the master key $Km$.

**(Encrypt)**

$$\begin{aligned} U &\longrightarrow C \quad : \{KL\}_{Km \oplus left}, \{KR\}_{Km \oplus right}, data \\ C &\longrightarrow U \quad : \{data\}_{KL|KR} \end{aligned}$$

The CCA supported single DES in a special legacy mode: a 'replicate' 3DES key could be generated, with both halves the same. 3DES is encryption with $K1$, followed by decryption with $K2$, then encryption with $K1$, so if $K1 = K2$ then $E(K1, D(K1, E(K1, data))) = E(K1, data)$, and a replicate key performs exactly as a single DES key.

**(Generate Replicate)**

$$C \longrightarrow U : \{X\}_{Km \oplus left}, \{X\}_{Km \oplus right}$$

The flaw was that the two halves of 3DES keys were not bound together with each other properly, only separated into left and right. A large set of replicate keys could be generated and two cracked using a parallel key search. Swapping the halves of two replicate keys would then yield a known true 3DES key, which could be used to export other intrinsically valuable keys.

**(Generate Replicate) x2**

$$\begin{aligned} C &\longrightarrow U: \quad \{X\}_{Km \oplus left}, \{X\}_{Km \oplus right} \\ C &\longrightarrow U: \quad \{Y\}_{Km \oplus left}, \{Y\}_{Km \oplus right} \end{aligned}$$

$$\text{Known key}: \{X\}_{Km \oplus left}, \{Y\}_{Km \oplus right}$$

This key binding attack reduced the CCA's 3DES from $2^{112}$ to $2^{57}$ (only twice as good as single DES). We describe multiple completions of an attack using this technique in [15], [20], [26].

### *F. Implementation Faults*

Implentation level faults also have a part to play in the vulnerability of security APIs. Reference [20] describes a serious failure in IBM's CCA check value generation command, which does not properly check the type associated with keys, permitting creation of a check value, for instance, over a sub-component of a key. Further implementation errors in checking of rules for exporting more secure keys encrypted under less secure keys, and in internal caching algorithms are reported in IBMs version history and update pages [39], [40], [41].

## VI. Security APIs – Information Leakage Attacks

The previous collection of API attacks is well summarised in [15], but relate to weaknesses in the *key management architecture* of financial HSMs rather than the specific PIN processing command set. Clulow examined the specific financial functionality closely, discovering information leakage attacks at API level. These are detailed further in his MSc thesis "The Design and Analysis of Cryptographic APIs for Security Devices" [27].

### *A. PIN Format Attack*

Reference [27] describes a classic attack against the ISO-0 PIN block encryption standard. ISO-0 formats a PIN as sixteen hexadecimal digits containing PIN, control information and padding. This buffer is then XORed with the customer's account number before encryption under the appropriate key. We represent this as $\{P \oplus A\}_{PEK}$ where

$PEK$ is the key, $P$ the formatted PIN buffer and $A$ the account number.

$$\begin{array}{ll} U \longrightarrow C: & X, \quad \{P \oplus A\}_{PEK} \\ C \longrightarrow U: & ((P \oplus A) \oplus X) < 10 \end{array}$$

Fig. 10. The PIN Integrity Check Protocol

Whenever the encrypted PIN is verified or re-encrypted under a different key, it must be decrypted, so the formatting process is reversed, which requires the user to submit the customer's account number, denoted $X$. The PIN is extracted and an integrity check is applied. Since each digit of the PIN is meant to be a decimal digit in the range `0` to `9`, the check simply tests that each hexadecimal PIN digit extracted from the decrypted buffer is less than ten (Atalla call this a "PIN Sanity Check" [13]). Should this test fail, it means that either the account number, key or encrypted PIN is incorrect or corrupted.

| | | $P \oplus A$ | | | | |
|---|---|---|---|---|---|---|
| | | 0,1 | 2, 3 | 4, 5 | 6, 7 | 8, 9 |
| | 0,1 | Pass | Pass | Pass | Pass | Pass |
| | 2,3 | Pass | Pass | Pass | Pass | Fail |
| | 4,5 | Pass | Pass | Pass | Pass | Fail |
| $X$ | 6,7 | Pass | Pass | Pass | Pass | Fail |
| | 8,9 | Pass | Fail | Fail | Fail | Pass |
| | A,B | Fail | Pass | Fail | Fail | Pass |
| | C,D | Fail | Fail | Pass | Fail | Pass |
| | E,F | Fail | Fail | Fail | Pass | Pass |

Fig. 11. Identifying a PIN using the PIN Integrity Check

At first glance, the integrity check seems to have enough, and does indeed detect unintended errors. However repeated execution of this protocol with different values of the claimed account number, $X$, quickly leads to the identification of the PIN. This can clearly be seen from figure 11. A given value of $P \oplus A$ results in a unique pattern of passes and fails, indentifying each PIN digit down to the set $\{P, P \oplus 1\}$.

In practice, the attack is slightly more sophisticated, requiring extra tricks to reveal all the digits of the PIN, and to determine the exact value from the set of $\{P, P \oplus 1\}$. Details can be found in [27].

### B. The Differential Protocol Attack

The decimalisation table attack is the subject of [17] and is also described in [27], [20], [5]. It concerns the way that PINs are generated and verified. The customer account number is encrypted with a PIN derivation key (PDK), yielding an essentially random string of 16 hexadecimal digits (one 64-bit DES block). The first four are selected, and are converted to decimal digits for use as the PIN. This mapping is done from a user-supplied function: a decimalisation table ($dec1$), usually having the specific value shown in figure 12. In this example, the hex character `0` is mapped onto `0`, while `A` is also mapped onto `0`. This *dectab* is written as `0123456789012345`.

| Hexadecimal Digit | `0123` | `4567` | `89AB` | `CDEF` |
|---|---|---|---|---|
| Mapped Decimal Digit | `0123` | `4567` | `8901` | `2345` |

Fig. 12. An Example Decimalisation Table

A PIN derived in such a method is never returned to the host in the clear but either sent to a secure printer, or returned encrypted under a PIN Encryption Key (PEK) for the purposes of later printing at a different site.

$$\begin{array}{ll} U \longrightarrow C: & A, \quad dec1 \\ C \longrightarrow U: & \{P \oplus A\}_{PEK} \text{ where } P = dec1(\{A\}_{PDK}) \end{array}$$

Fig. 13. Account Number Based PIN Generation

The problem is that the decimalisation table is open to malicious manipulation. If we set the table to all zeros (i.e., `0000000000000000`) then a PIN of '0000' must be generated and it is returned in encrypted form. We then repeat the call using a slightly modified table `1000000000000000`. If the result of the encryption of the account number (i.e., $\{A\}_{PDK}$) does not contain a zero hexadecimal digit in the first four hexadecimal digits, then PIN will be unchanged and the same encrypted PIN block will be returned. The technique can be similarly repeated with different tables to determine the constituent digits of the PIN. Since the method exploits comparison of repeated (slightly modified) runs of the same protocol, the term "Differential Protocol Analysis" was coined in [5] for this type of attack.

The precise PIN can be discovered by exploiting the HSM support for customer-chosen PINs. As already described, chosen PINs are stored as the offset between the generated PIN and the new chosen PIN ($offset = customerPIN - generatedPIN(mod10)$). This offset is stored in the bank's master database. Verification with offsets proceeds according to figure 14.

$$\begin{array}{ll} U \longrightarrow C: & \{P \oplus A\}_{PEK}, A \ , \ dec1 \ , \ offset \\ C \longrightarrow U: & true \quad \text{if} \quad P = dec1(\{A\}_{PDK}) + offset \end{array}$$

Fig. 14. The PIN Verification Protocol

The function shown in figure 14 remains vulnerable to the previous attack. Given an valid encrypted PIN block and its corresponding account, we can again manipulate the decimalisation table to learn the PIN. Suppose we modify the usual decimalisation table from `0123456789012345` to `1123456789012345`, while keeping the other inputs $A$, $\{P \oplus A\}_{PEK}$ and *offset* constant. If the PINs do not match and the verification fails, we learn that $\{A\}_{PDK}$ contains at least one instance of the modified digit. Owing to the simple arithmetic relationship between customer PINs, generated PINs and offsets, we determine which particular digits are affected by repeating the call with suitably selected values of *offset*. We thus learn both the value of the constituent digits of the PIN and their positions.

### C. Statistical Attacks on Encrypted Data

An unfortunate consequence of the decimalisation process is that it skews the otherwise uniform distribution of generated PINs, leading to the statistical attacks described in [18]. Kuhn uses this fact to imrove the quality of an outsider attack on a German banking scheme in [47]. We consider the risks of an insider exploiting such a weakness: suppose the PIN generation command in figure 15 is repeatedly executed, choosing many random PDKs (key conjuring enables this [15]), keeping $A$ and the *offset* constant. PINs will be created according to frequency distribution of digits in the decimalisation table, and this distribution will be visible in the corresponding frequency of encrypted PIN blocks returned. This in itself is not yet sufficient to uniquely identify a PIN as many will have the same expected frequency.

$$\begin{aligned} U \longrightarrow C: &\quad A, D, \mathit{offset} \\ C \longrightarrow U: &\quad \{P \oplus A\}_{PEK}, \\ &\quad \text{where} \quad P = D(\{A\}_{PDK}) + \mathit{offset} \end{aligned}$$

Fig. 15. Protocol for PIN Generation with Offsets

However, manipulating the *offset* parameter allows the encrypted PINs to be assembled into order. We generate an encrypted PIN for fixed input values of $A$ and $D$, and $PDK$, but vary the offset starting from `0000`. We increment the offset by 1 (mod 10) and repeat the call, thus obtaining the encrypted PIN block corresponding to an adjacent PIN in the space of possible PINs. Once the relationships between all encrypted PIN blocks is known, coupled with the knowledge of the frequency of occurence of each block, we can identify a particular block (and thus all blocks) uniquely.

### D. ISO-0 Collision Attack

In fact, the ISO-0 format described in VI A above leaks information regardless of the algorithm used to generate PINs, or the uniformity of PINs. Consider a fixed account number $A$ and decimalistaion table $D$: all possible encrypted PIN blocks – $\{P \oplus A\}_{PEK}$ – can be generated by manipulating offsets as described before. Since both PIN digits $P$ and account digits $A$ are decimal, we will observe an imcomplete set of possible hexadecimal values $P \oplus A$. Duplicating this process with a second account number $A'$, we obtain second incomplete set of encrypted PIN blocks.

The attacker can now compare the sets of encrypted PIN blocks resulting from the two runs. Both runs contain all the PINs. However, the two runs used different account numbers which affects how the set of PINs are encrypted and stored. Now the XOR of many PINs with the account number $A$ from the first run will match the XOR of other PINs and the different account number $A'$ from the second run. Whenever this occurs, the encrypted blocks will match as well (that is, have the same value). Conversely, each encrypted PIN block that exists in only one of the lists (without loss of generality, assume the list from the first run), corresponds to a value of $P \oplus A$ that is not achievable in the second run. This allows the attacker to determine a set of possible values for $P \oplus A$, and hence for $PIN$.

We illustrate this technique with a toy example using single digit PINs. The attacker has constructed the two lists of all the possible encrypted blocks for the accounts with PANs `7` and `0`, as shown in figure 16.

| A | P | (P⊕A) | encblock | A' | P | (P⊕A') | encblock |
|---|---|---|---|---|---|---|---|
| 7 | 0 | 7 | `2F2C` | 0 | 0 | 0 | `21A0` |
| 7 | 1 | 6 | `345A` | 0 | 1 | 1 | `73D2` |
| 7 | 2 | 5 | `0321` | 0 | 2 | 2 | `536A` |
| 7 | 3 | 4 | `FF3A` | 0 | 3 | 3 | `FA2A` |
| 7 | 4 | 3 | `FA2A` | 0 | 4 | 4 | `FF3A` |
| 7 | 5 | 2 | `536A` | 0 | 5 | 5 | `0321` |
| 7 | 6 | 1 | `73D2` | 0 | 6 | 6 | `345A` |
| 7 | 7 | 0 | `21A0` | 0 | 7 | 7 | `2F2C` |
| 7 | 8 | F | `AC42` | 0 | 8 | 8 | `4D0D` |
| 7 | 9 | E | `9A91` | 0 | 9 | 9 | `21CC` |

Fig. 16. Sets of encrypted all PIN blocks for accounts with PANs 7 and 0

The encrypted block `AC42` from the left hand list does not occur in the right hand list, and likewise for encblock `9A91`. Therefore, these blocks must have hexadecimal values that cannot result from XOR of the account number `0` with a decimal PIN digit: the two possibilities are `8` and `9`. This deduction has the same associated restrictions as that in section VI-A.

### E. PVV Clash Attack

VISA introduced a *Pin Verification Value* (PVV) method for PIN generation and verification which avoids attacks associated with decimalisation tables as it uses an unbiased random number source to generate the PINs themselves. However, the verification procedure has an interesting shortcoming.

To generate a PVV a *Transaction Security Parameter* (TSP) is first constructed, containing the account number, the generate PIN itself (and a generation key identifier). This TSP is then encrypted with the "PVV master key", and simplified to a four digit PIN Verification Value which can be stored in the clear at the bank's convenience, in a database or on an ATM card. The weakness is this: due to the short length of the PVV, and considering the encryption function as a random oracle, multiple TSPs will produce the same PVV as a result. 60% of accounts will have two or more correct PINs, while 0.5% of accounts have five or more correct PINs. With moderate effort, an attacker could observe and identify accounts with severe PIN space clashes.

While in practice the attack is not the easiest way to compromise PINs, the existence of an unknown number of correct PINs for each bank account may have interesting legal implications [55].

## VII. Assurance and policy issues

The above section should have convinced the reader that the design of security APIs is an extremely hard problem.

If, on the one hand, we implement a simple crypto API that allows the server to call any cryptographic functions it pleases, then we have in effect put our keys at the disposal of the server. If the server gets hacked, then the physical protection that the cryptoprocessor offers to our keys may not buy us very much. Even although the keys remain in our custody, the opponent can use them as she pleases – so they are not in our effective custody.

If, on the other hand, we design an API that follows our application requirements closely – as the PIN-management API does – then we are in effect implementing a multi-party computation, many of whose inputs must be assumed to be under the control of a malicious adversary. We do not know how, in general, to devise such a computation so that it will use a secret safely, and not leak it in response to carefully chosen inputs. Things are made worse by the business pressures that cause ever more 'features' to be added. (The original designer of the VSM assured us that the device was secure when he sold it to VISA; it was the extra transactions added since then that made it insecure.)

Given this dilemma, how is a system designer to go about selecting a cryptoprocessor product, and designing a security API that does useful work?

### A. Evaluation and Certification

One response is to look for evaluations by third parties. There are two schemes under which cryptoprocessors are certified – FIPS 140 [34], run by the US National Institute of Standards and Technology, and the Common Criteria [28], operated by a number of member countries.

Both have drawbacks. FIPS looks only at the tamper-resistance of the hardware; the 4758 was certified to level 4, the highest available level, for hardware tamper-proofness while the CCA software supplied with it to almost all customers contained the fatal vulnerabilities described above. A FIPS evaluation, especially at level 4, is nonetheless of value to the discerning customer. Smith et al. describe the processes and modelling entailed in validating the 4758's firmware in [63], [64], [65].

The Common Criteria are used to evaluate products according to 'protection profiles' – formalised statements of the protection properties that the device is supposed to deliver. Often, inappropriate protection profiles are used that protect the wrong things. The associated politics, and some of the problems that can arise, are described in [3]. We have yet to see a convincing protection profile for a security API.

### B. Formal Analysis of Security APIs

The formal analysis of security APIs is in its infancy compared with the well-developed evaluation and assurance procedures for hardware and firmware. Longley and Rigby had some success during the late 80s automating the analysis of "key management systems" (essentially the security APIs of the day) initially using expert systems and later PROLOG [48], [57], [49], [50], working from protocol analysis experiments as a starting point.

Their tools searched for sequences of commands which could violate a security property, and focused on the use of heuristics to direct and restrict the search. If no attacks were found, a measure of assurance about API correctness could be obtained.

Bond, Clulow and colleagues have gone on to re-examine this approach exploiting the vastly increased computing power available a decade later, using more sophisticated modern protocol analysis tools, and generic tools such as theorem provers [72], [69], [56]. This is the subject of ongoing research in a Cambridge-MIT Institute research program. Meanwhile, recent work by Ganapathy et al. [35] on automated discovery of API-Level vulnerabilities has modelled type-casting attacks [21] on the IBM CCA. The authors describe some of the challenges facing the protcol analysis community in extending their tools to analyse security APIs in [19].

### C. Other policy issues

The use of cryptoprocessors is not free of controversy. For example, if TC/IRM mechanisms enable a company to cause all internal emails to become unreadable after 60 days, this may help the company weather the discovery process in litigation – but what if the company is the subject of a criminal investigation? Crypto policy was controversial in the mid-1990s; the widespread deployment of hardware-assisted strong cryptography that the Trusted Computing Group envisages may make it so once more.

A second bundle of issues has to do with competition and trade policy [9]. The fastest-growing applications of cryptoprocessors are in support of restrictive business models – as with the tiny cryptoprocessors embedded in ink cartridges to authenticate them to the printer as coming from the same company. This enables printer vendors to subsidise the sales of printers by charging more for ink, but is controversial – the European Union has responded with an environmental directive requiring all ink cartridges to be refillable by 2007.

The spread of 'trusted computing' mechanisms is likely to exacerbate these issues. Platforms containing virtual cryptoprocessors, which any developer can invoke, will tempt application writers to lock their customers in more tightly, to tie products together, to enforce incompatibility, and to experiment with all sorts of new business models. Many of these models may evoke consumer resistance, or litigation from competitors.

## VIII. Conclusions

We have surveyed cryptoprocessors and their applications. Low-cost and mid-cost cryptoprocessors are the most rapidly developing area, with 'trusted computing' likely to bring them into many mass-market platforms. The enforcement of novel business models is the most rapidly developing new application.

Possible attack technologies range from the use of semiconductor test equipment to access secret signals directly, through sophisticated statistical techniques building on the general principles of power and emissions analysis, and low-

cost versions of optical probing attacks once considered beyond the range of a moderately funded adversary. Logical attacks – on the security API of a cryptoprocessor – have seen very rapid development in recent years, and are likely to remain the weak spot of most high-end systems.

There are many interesting topics for the academic researcher. In particular, security API attacks are still poorly understood, and there may be considerable scope for applying formal techniques to their analysis.

Finally, it must not be forgotten that many cryptoprocessor applications can be controversial, and particularly those applications that seek to establish or enforce restrictive business models. Designers must therefore bear in mind one final kind of attack. This is the legal attack – in which a judge tells you to hand over the keys to your competitor, or go to jail.

## Acknowledgments

The authors would like to acknowledge the funding of the Cambridge-MIT Institute and the Cecil Renaud Educational and Charitable Trust.

## References

[1] C. Ajluni, "Two New Imaging Techniques Promise To Improve IC Defect Identification", Electronic Design, Vol. 43(14), July 1995, pp. 37-38

[2] R. Anderson, "The Correctness of Crypto Transaction Sets", 8th International Workshop on Security Protocols, Cambridge, UK, April 2000

[3] R. Anderson, "Security Engineering – a Guide to Building Dependable Distributed Systems", Wiley (2001) ISBN 0-471-38922-6

[4] R. Anderson, "Why Cryptosystems Fail" in Communications of the ACM vol 37 no 11 (November 1994) pp 32-40; earlier version at http://www.cl.cam.ac.uk/users/rja14/wcf.html

[5] R. Anderson, M. Bond, "Protocol Analysis, Composability and Computation", Computer Systems: Papers for Roger Needham, Jan 2003

[6] R. Anderson, S. Bezuidenhoudt, "On the Reliability of Electronic Payment Systems", in IEEE Transactions on Software Engineering vol 22 no 5 (May 1996) pp 294-301; http://www.cl.cam.ac.uk/ftp/users/rja14/meters.ps.gz

[7] R. Anderson, M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices", in Security Protocols (Proceedings of the 5th International Workshop (1997) Springer LNCS vol 1361 pp 125–136

[8] R. Anderson, M. Kuhn, "Tamper Resistance – A Cautionary Note", 2nd USENIX Workshop on Electronic Commerce, Nov 1996, pp 1-11, ISBN 1-880446-83-9

[9] R. Anderson, "Cryptography and Competition Policy", Economics of Information Security, ed. LJ Camp, S Lewis, Kluwer 2004, pp. 35-52

[10] R. Anderson, "Trusted Computing FAQ", http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html

[11] R. Anderson, S. Moore, P. Cunningham, R. Mullins, G. Taylor, "Improving Smart Card Security using Self-timed Circuits", Asynch 2002 (to appear)

[12] R. Anderson, S. Moore, R. Mullins, G. Taylor, J. Fournier, "Balanced Self-Checking Asynchronous Logic for Smart Card Applications", Microprocessors and Microsystems (to appear)

[13] Atalla A8000NSP Banking Command Reference Manual, HP-Atalla, Mar 2001

[14] E. Biham, A. Shamir, "Differential Fault Analysis: A New Cryptanalytic Attack on Secret Key Cryptosystems", 1997, Springer LNCS 1294, pp. 513

[15] M. Bond, "Attacks on Cryptoprocessor Transaction Sets", CHES 2001, Springer LNCS 2162, pp. 220-234

[16] M. Bond, R. Anderson, "API-Level Attacks on Embedded Systems", IEEE Computer, Oct 2001, Vol 34 No. 10, pp. 67-75

[17] M. Bond, P. Zielinski, "Decimalisation Table Attacks for PIN Cracking", University of Cambridge Computer Laboratory Technical Report TR-560

[18] M. Bond, J. Clulow, "Encrypted? Randomised? Compromised? (When Cryptographically Secured Data is Not Secure)", Cryptographic Algorithms and Their Uses, Eracom Workshop 2004, Queensland Australia

[19] M. Bond, J. Clulow, "Extending Security Protocols Analysis: New Challenges", Automated Reasoning and Security Protocols Analysis (ARSPA) 2004, Cork, Ireland

[20] M. Bond, "Understanding Security APIs", PhD Thesis, University of Cambridge, Jan 2004

[21] M. Bond, "A Chosen Key Difference Attack on Control Vectors", 1st November 2000, Unpublished.

[22] D. Boneh, R. A. Demillo, R. J. Lipton, "On the importance of checking cryptographic protocols for faults", Advances in Cryptology – Eurocrypt '97, Springer LNCS 1294, pp. 37-51

[23] S. Brands, "Untraceable Off-line Cash in Wallet with Observers", Crypto '93, Springer LNCS 773, pp.302-318

[24] CanalPlus vs. NDS, Allegations that NDS lab in Israel cracked CanalPlus key and leaked data to internet, March 2002

[25] D. Chaum, "Blind Signatures for Untraceable Payments", Crypto '82, Plenum Press (1983), pp. 199-203

[26] R. Clayton, M. Bond, "Experience Using a Low-Cost FPGA Design to Crack DES Keys", CHES Workshop 2002, San Francisco, Springer LNCS 2523, pp. 579-592

[27] J. Clulow, "The Design and Analysis of Cryptographic APIs for Security Devices", MSc Thesis, University of Natal, SA, 2003

[28] Common Criteria Evaluation Scheme, http://www.commoncriteriaportal.org/

[29] Y. Desmedt, "An Exhaustive Key Search Machine Breaking One Million DES Keys", Eurocrypt, 1987

[30] W. Diffie and M. Hellman, "Exhaustive cryptanalysis of the NBS Data Encryption Standard" , Computer vol.10 no.6 (June 1977) pp. 74-84

[31] J. Dyer, R. Perez, S.W. Smith, M. Lindemann, "Application Support for a High-Performance, Programmable Secure Coprocessor", 22nd National Information Systems Security Conference, Oct 1999

[32] J. Dyer, M. Lindemann, R. Perez, R. Sailer, S.W. Smith, L. van Doorn, S. Weingart, "Building the IBM 4758 Secure Coprocessor", IEEE Computer, Oct 2001, Vol 34 pp. 57-66

[33] EMV 4.1, Integrated Circuit Card Specifications for Payment Systems, June 2004, available from http://www.emvco.com

[34] "Security Requirements for Cryptographic Modules" NIST Federal Information Processing Standard 140-1 and 140-2, http://csrc.nist.gov/cryptval/

[35] V. Ganapathy, S. A. Seshia, S. Jha, T. W. Reps, R. E. Bryant, "Automatic Discovery of API-Level Vulnerabilities", UW-Madison Computer Sciences Technical Report UW-CS-TR-1512, Jul 2004, http://www.cs.wisc.edu/~vg/writings/papers/tr1512.pdf

[36] IBM 3614 Consumer Transaction Facility Implementation Planning Guide, IBM document ZZ20-3789-1, Second edition, December 1977

[37] IBM, 'IBM 4758 PCI Cryptographic Coprocessor – CCA Basic Services Reference and Guide, Release 1.31 for the IBM 4758-001'

[38] IBM Cryptocards Homepage for the 4758 http://www.ibm.com/security/cryptocards/

[39] IBM Inc.: CCA Version 2.41. (5 Feb 2002) http://www-3.ibm.com/security/cryptocards/html/release241.shtml

[40] IBM Inc.: CCA Version 2.42. (Jul 2004) http://www-3.ibm.com/security/cryptocards/html/release242.shtml

[41] IBM Inc.: Version history of CCA Version 2.41, IBM 4758 PCI Cryptographic Coprocessor CCA Basic Services Reference and Guide for the IBM 4758-002. IBM, pg xv (Feb 2002)

[42] "IBM Enhanced Media Management System", http://www-306.ibm.com/software/data/emms/

[43] J King, "Bolero — a practical application of trusted third party services", in *Computer Fraud and Security Bulletin*, July '95 pp. 12-15

[44] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", Advances in Cryptology – Crypto '96, Springer LNCS 1109, pp. 104-113

[45] P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis", Advances in Cryptology – Crypto '99, Springer LNCS 1666, pp. 388-397

[46] O. Kommerling, M. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors", USENIX Workshop on Smartcard Technology, 1999, ISBM 1-880446-34-0, pp. 9-20
[47] M. Kuhn, "Probability Theory for Pickpockets – ec-PIN Guessing", available from http://www.cl.cam.ac.uk/~mgk25/
[48] D. Longley, S. Rigby, "An Automatic Search for Security Flaws in Key Management", Computers & Security, March 1992, vol 11, pp. 75-89
[49] D. Longley "Expert Systems Applied to the Analysis of Key Management Schemes", Computers & Security, Feb 1987, Vol 6, No 1, pp. 54-67
[50] D. Longley, S. Vasudevan, "Effects of Key Generators on the Automatic Search for Flaws in Key Management Schemes", Computers & Security, 1994, Vol 13 No 4, pp. 335-347
[51] S.M. Matyas, "Key Handling with Control Vectors", IBM Systems Journal v. 30 n. 2, 1991, pp. 151-174
[52] S.M. Matyas, A.V. Le, D.G. Abraham, "A Key Management Scheme Based on Control Vectors", IBM Systems Journal v. 30 n. 2, 1991, pp. 175-191
[53] Microsoft Next-Generation Secure Computing Base (NGSCB), http://www.microsoft.com/resources/ngscb/default.mspx
[54] P. van Oorschot, M. Wiener, "Parallel Collision Search With Applications to Hash Functions and Discrete Logarithms", Second ACM Conference on Computer and Communications Security, ISBN 0-89791-732-4, pp. 210-218
[55] T. Opel, Private communication. Details available at http://www.kreditkartendiebe.de/
[56] Otter – An Automated Deduction System, http://www-unix.mcs.anl.gov/AR/otter/
[57] S. Rigby, "Key Management in Secure Data Networks", MSc Thesis, Queensland Institute of Technology, Australia, 1987
[58] D. Samyde, J. Quisquater, "Electromagnetic Analysis (EMA): Measures and Counter-Measures for Smartcards", E-Smart 2001, Springer LNCS 2140, pp. 200-210
[59] D. Samyde, S. Skorobogatov, R. Anderson, J. Quisquater, "On a New Way to Read Data from Memory", SISW2002 – First International IEEE Security in Storage Workshop
[60] S. Skorobogatov, R. Anderson, "Optical Fault Induction Attacks", Cryptographic Hardware and Embedded Systems Workshop, CHES 2002, Springer LNCS 2523, pp. 2-12
[61] S. Skorobogotov, "Low temperature data remanence in static RAM", University of Cambridge Computer Laboratory Technical Report TR-536 http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-536.pdf
[62] S. Skorobogatov, "Semi-Invasive Attacks – A New Approach to Hardware Security Analysis", PhD Thesis, University of Cambridge, September 2004
[63] S.W. Smith, V. Austel, "Trusting Trusted Hardware: Towards a Formal Model for Programmable Secure Coprocessors", 3rd USENIX Workshop on Electronic Commerce, Aug 1998
[64] S.W. Smith, R. Perez, S.H. Weingart, V. Austel, "Validating a High-Performance, Programmable Secure Coprocessor", 22nd National Information Systems Security Conference, Oct 1999
[65] S.W. Smith, "Outbound Authentication for Programmable Secure Coprocessors", 7th European Symposium on Research in Computer Security, Springer LNCS 2502, pp. 72-89
[66] S.W. Smith, E.R. Palmer, S. Weingart, "Using a High-Performance, Programmable Secure Coprocessor", 2nd International Conference on Financial Cryptography, Feb 1998, Springer LNCS 1465
[67] S.W. Smith, S. Weingart, "Building a High-Performance, Programmable Secure Coprocessor", Computer Networks (Special Issue on Computer Network Security) Apr 1999, Vol 31 pp. 831-860
[68] S.W Smith, "Fairy Dust, Secrets and the Real World", IEEE Security and Privacy, Jan/Feb 2003, pp. 89-93
[69] SPASS – An Automated Theorem Prover for First-Order Logic with Equality http://spass.mpi-sb.mpg.de/
[70] Trusted Computing Group, Trusted Platform Module Specifications v1.2, available from https://www.trustedcomputinggroup.org
[71] S. Weingart, "Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses", CHES 2000, Springer LNCS 1965, pp. 302-317
[72] P. Youn, "The Analysis of Cryptographic APIs Using Formal Methods", Massachusetts Institute of Technology, June 2004