Cryptographic Processors—A Survey

ROSS ANDERSON, MIKE BOND, JOLYON CLULOW, AND SERGEI SKOROBOGATOV

Invited Paper

Tamper-resistant cryptographic processors are becoming the standard way to enforce data-usage policies. Their origins lie with military cipher machines and PIN processing in banking payment networks, expanding in the 1990s into embedded applications: token vending machines for prepayment electricity and mobile phone credit. Major applications such as GSM mobile phone identification and pay TV set-top boxes have pushed low-cost cryptoprocessors toward ubiquity. In the last five years, dedicated crypto chips have been embedded in devices such as game console accessories and printer ink cartridges, to control product and accessory aftermarkets. The "Trusted Computing" initiative will soon embed cryptoprocessors in PCs so they can identify each other remotely. This paper surveys the range of applications of tamper-resistant hardware and the array of attack and defense mechanisms which have evolved in the tamper-resistance arms race.

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

I. Introduction

The combination of cryptography and tamper-resistance first appeared in military applications such as securing communications links, and the spread of automated teller machine (ATM) networks brought the technology into the commercial mainstream. Devices used for protecting ATM networks were subsequently adapted for a much wider range of applications.

A typical high-end cryptoprocessor is a physically tamper-resistant embedded processor which communicates with a conventional computer 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 implements the device's security policy. We discuss security APIs in Section V.

The 1990s saw cryptoprocessors gaining widespread use: protecting Secure Socket Layer (SSL) keys used by webservers and defending proprietary software and algorithms from theft by employees; low-cost cryptoprocessors such as smart cards and secure microcontrollers also became commonplace. A whole host of embedded applications 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 smart cards 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 toward tamper-resistant processors to enforce arbitrary policies on the way content is processed. The range of possible applications is incredible, and—to some observers—disturbing [10]. The entertainment industry in particular seeks further revenue 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 Section V describes logical attacks on the security APIs of cryptoprocessors. Sections VI and VII look at policy issues and draw conclusions.

II. Applications

A. ATM Security

ATMs were the "killer application" that got cryptography into wide use outside of military and diplomatic circles and

Manuscript received October 22, 2004; revised January 3, 2005.

The authors are with the Computer Laboratory, University of Cambridge, Cambridge CB3 0FD, U.K. (e-mail: ross.anderson@cl.cam.ac.uk; mike.bond@cl.cam.ac.uk; jolyon.clulow@cl.cam.ac.uk; sergei.skorobogatov@cl.cam.ac.uk).

Digital Object Identifier 10.1109/JPROC.2005.862423

remain a high-volume use for tamper-resistant hardware. In the 1970s, IBM developed a system for authenticating customers to ATMs. Bank customers were issued with PINs, computed from their account numbers using a secret key, the *PIN derivation key*. References [3], [4], and [37] describe the early 3614 ATM series and its accompanying back-end processor, the 3848. This was the first commercial *hardware security module* (HSM), as stand-alone cryptoprocessors have come to be known in the financial sector. HSMs controlled access to the keys, and also kept PINs secret in transit through the network. They are still used during all stages of PIN management, including *acquisition* of PINs at the ATMs; *verification* at the card issuing bank; and also during *generation* processes, e.g., at card issuing sites.

The HSM's API is designed to allow legitimate operations on PINs and derivation keys, such as verifying a an encrypted trial PIN from an ATM, sending a clear PIN to a special printer to be sent to the customer, or enforcing dual control policies on key material access. In addition, it must prevent abuse by malicious employees; for instance, it must not allow wholesale discovery of the PIN for every bank account.

B. Other Electronic Payment Schemes

The movement by European banks to replace magnetic stripe cards with smart cards has spread HSM-based cryptography from ATM networks into point-of-sale (POS) systems. Bank cryptoprocessors now support communications between banks and merchants, and authenticate cards presented at POS terminals. The merchant devices may contain low-cost cryptoprocessors, and there is now a third location for a cryptoprocessor—on the customer's card. These "chip and PIN" smart cards store keys, certificates, and customer PINs, and enable mutual authentication between the smart card, the terminal, and the issuing bank through the EMV API [34]. HSMs are also an integral part of the back-end systems at banks which process these transactions, preventing insiders from exploiting their positions.

The concerns raised by Internet banking and payment can be assuaged in part by using cryptoprocessors. The challenge is to create trust in the user's home computing platform: TC for PCs (fitting every computer with a cryptoprocessor) and disconnected cryptoprocessor authorization devices (such as the RSA SecurID) are some of the possible answers. The latter effectively exports a security API to the PC through the user.

"Digital cash" and other futuristic schemes use HSMs at trusted third parties to 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. A combination of trusted execution for code and ingenious cryptography gives useful properties, such as causing double-spending to automatically reveal a violator's identity to the issuer [23], [25].

C. Trusted Computing

"Trusted Computing" is an umbrella term for new technologies designed to embed cryptoprocessors in current computing platforms, including PCs and PDAs. The Trusted Computing Group (TCG) industry consortium designed the Trusted Platform Module (TPM) cryptoprocessor, which aims to build an island of trust within the desktop PC [67], as well as on other platforms. The first-generation TPM (already deployed in IBM laptops) was designed for key storage and passive measurement of the machine state. The next-generation TPM will support desktop computers, for instance, for secure boot. It will also play a part in Microsoft's plans for a trust in their Vista operating system (formerly named Longhorn/Palladium/NGSCB) [50]. This TPM, in combination with a software microkernel which it validates, will form a virtual cryptoprocessor that 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 better memory protection, it aims to prevent one program from interfering with another.

The major application of TC is DRM: the control of the distribution and use of data. 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; thus, the vendor gets better assurance that the song or movie will remain under its control. 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. Some vendors believe that more effective technical DRM mechanisms are the best way to protect their revenue streams. They may also support alternative marketing strategies, such as subscription services for listening to music. (IBM's Enhanced Media Management System has an optional secure hardware component [39] 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, but can also be applied to electronic documents and e-mail. For example, Microsoft's Information Rights Management (IRM) technology supports restrictions on documents such as limiting access to named machines or self-destruction after a chosen period. 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.

D. Public-Key Cryptoprocessors

The arrival of public-key technology spawned a range of new secure communications protocols. SSL 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 browsers authenticate a certificates chain that relates domain names to SSL protocol public keys. The user relies on this chain to be sure that she is communicating directly with the correct webserver.

The computation cost of crypto in webservers drove the development of the current generation of public-key cryptoprocessors. A server farm may also enforce a simple *nonexport* 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.

E. Other Applications

Prepayment electricity meters: HSMs also play an important role in prepayment electricity meter systems [6], particularly at token vending stations. Here, the HSM can limit the loss if a vending machine is stolen or misused by mantaining a credit counter which will prevent further dispensing of credit tokens once the balance is exhausted.

Military applications: Some World War II, military cipher machines were capture-resistant: thermite charges destroyed the mechanism if the devices were opened incorrectly. The damages caused during the Cold War by traitors (in particular by the Walker family) convinced the National Security Agency (NSA) that cipher machines should, as far as possible, also resist dishonest insiders. Modern military cipher machines use classified algorithms in tamper-resistant chips and transport initial key material in special *crypto ignition keys*.

III. Taxonomy of Cryptoprocessors and Attacks

Early military cipher machines may be seen at the NSA Museum, Fort George Meade, MD; 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, a mainframe peripheral. It was contained in a steel cabinet, with switches that zeroized the memory containing the keys whenever the cabinet was opened.

Top-of-the-range cryptoprocessors nowadays include the IBM 4758 (Fig. 1), the 3848's descendant, which has a cryptographic engine surrounded by a multilayer tamper-sensing mesh (Fig. 2). This is constantly monitored by the engine, which erases its key material if a tampering attempt is detected. Rather than occupying a whole cabinet as the 3848 did, the 4758 comes as a standard PCI card for mounting in a server. A rich literature documents its design, development, and validation [32], [33], [60], [61], [63], [64].

At the low 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 use shared-key cryptographic algorithms such as AES whose computational cost is low. A more serious problem is that the read-protect mechanisms in low-cost microcontrollers are not designed to withstand skilled and determined attack.

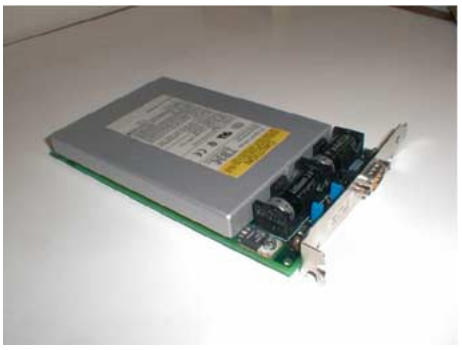
Fig. 1. IBM 4758-001.

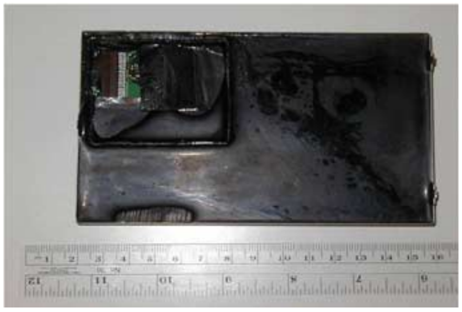
Fig. 2. An IBM 4758-001 part potted in urethane, showing membrane and interior (courtesy F. Stajano).

A middle market has therefore emerged of single-chip products that have been hardened in various ways against attack. These products include smart cards and the TPM chips specified by the TCG for use in PCs. Before deciding whether an application can use a low-cost microcontroller, or needs a smart card 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.

1) *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.
2) *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 authorized interface. For example, the attacker may use a laser beam to ionize a transistor and thus change the state of the flip–flop that holds the device's protection state.
3) *Local noninvasive 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.

4) *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 be either *active* or *passive*. In passive attacks the attacker works with the device as it is operated normally, while an active attack involes manipulating the device, its inputs or its environment so as to induce abnormal operation.

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 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 [65]).

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

IV. Local Attacks

Fifteen years ago, microcontrollers and smart cards offered little resistance to a capable motivated opponent. Protection typically consisted of a read-only bit that was set after programming; it could be reset by *glitching* [43]—inserting a transient into the power supply or clock signal—or illuminating it with UV light. Even so, microcontrollers gave better protection than discrete component circuits, observable directly with an oscilloscope. At that time, there were few valuable applications and thus few serious attackers.

That changed once smart cards started to host applications such as pay TV. Pirates wanted to forge and sell cards, while others wanted to publish key material 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 the contents [24].) Microcontrollers were also introduced for accessory control: for example, games consoles were subsidized from sales of software and accessories, as the security devices ensured that cartridge and accessory vendors paid the appropriate royalty. A vendor's competitors thus had a strong incentive to reverse-engineer its security chips—legal in most relevant jurisdictions. Thus, the arms race between attack and defense began.

In the mid-1990s, attackers invested in invasive attack methods, using probing stations; the late 1990s saw noninvasive attack techniques such as power analysis. Semi-invasive attacks were developed in the early 2000s, and the best known involve optical probing. Chipmakers have responded by developing new microcontroller families and smart cards with additional defensive features.

A. Invasive Attacks

The earliest tamper-resistant processors—the HSMs used by banks from the 1980s—were very vulnerable to physical attack. The protection mechanisms relied on steel boxes with lid switches and, sometimes, further sensors such as photodiodes and seismometers. An attacker familiar with these mechanisms could drill his way in. Furthermore, maintenance engineers with regular access could disable tamper-response on one visit, and attack on the next. Modern HSMs therefore use more complex designs. The IBM 4758, for example, has its components encased in a tamper-sensing membrane that is itself potted in a compound that is difficult to cut or drill cleanly (see Figs. 1 and 2).

Other tricks are occasionally used: custom-pressure air gaps, stressed glass components, and even shaped charges in military devices which will destroy the memory chip. Weingart surveys these anti-tamper technologies in [68]. Such designs are by now fairly mature; vendors can follow the FIPS 140 certification program to assure customers of a certain level of tamper-proofness (see Section VI).

Of more interest are the technologies relating to making single-chip devices resistant to invasive attack. By the mid-1990s, the availability of secondhand 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 a long working distance objective, mounted on a low-vibration platform with micromanipulators to place probes on the sample. It may also have a laser, to drill small holes in the chip's passivation layer. These holes allow electrical contact by the probes, and stabilize them in position. The *modus operandi* is to probe the device's internal bus, so that both program and data can be read out. Ingenious tricks minimize the probing required. For example, placing a single grounded probe on the clock line to the instruction latch causes the same instruction to be executed repeatedly. The program counter is thus incremented continuously, thus dumping the entire memory contents to the bus. The standard reference on microprobing is [43].

Invasive attacks have become much more difficult since the late 1990s. High-grade cards typically have a sensor mesh in the top metal layer (Fig. 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.

Also, the move to deep-submicrometer feature size is driving up the sophistication and cost of probing techniques.

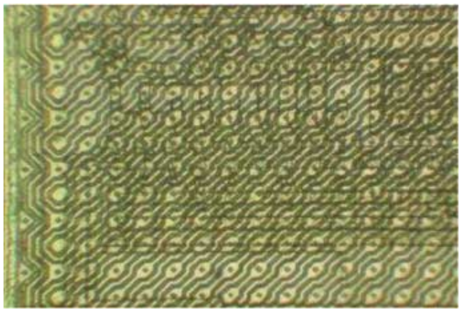

Fig. 3. Top metal sensor mesh on ST16 smart card.

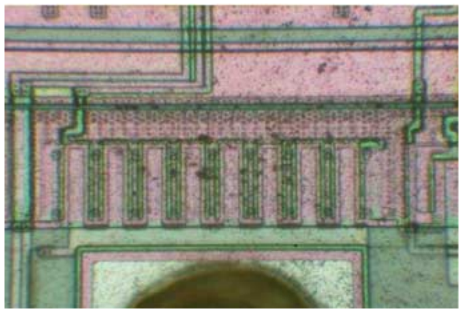

Fig. 4. PIC16F877 built with old technology (0.9 μm).

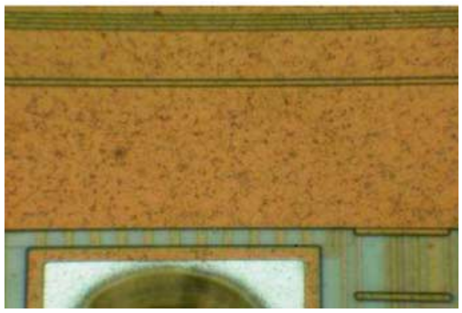

Fig. 5. PIC16F877A built with new CMP technology (0.5 μm).

Consider the structure of the Microchip PIC16F877 microcontroller, easily observable and under a microscope (Fig. 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 from deeper layers.

In 0.5-μm and smaller technologies, as in the later Microchip PIC16F877A microcontroller (Fig. 5), each layer but the last one is planarized using chemical–mechanical polishing before applying the next layer. Thus, the top metal layer does not show any trace of the deeper layers' structure, which can only be revealed by removing the higher layers mechanically or chemically.

Even once the chip is sufficiently well-understood that the attacker knows where to tap, making a useful electrical contact is not trivial. A focused ion beam (FIB) workstation may be needed to drill down to the appropriate component and bring out a contact to a bonding pad. However, the added capacitative load may cause the chip to fail. Defences such as refractory passivation layers, for example, that are slow to penetrate with the FIB or cause excessive charge buildup, can be used to trigger an alarm.

B. *Local Noninvasive Attacks*

In the mid-1990s, the cryptoprocessor industry received a rude shock when Kocher revealed the power of *timing attacks*. For example, an RSA operation involves many modular multiplications, and the time taken by an efficient implementation depends strongly on the input values. Suitable analysis of the time taken to compute the result can leak the private key [41]. Kocher delivered an even nastier shock shortly after, with the publication of *Differential Power Analysis* (DPA) in '98 [42]. DPA analyzes the current drawn, which for many processors is data-dependent.

These techniques showed that cryptoprocessor designers must consider *emission security* (EMSEC): the attack and defense mechanisms governing information leakage by electromagnetic means, surveyed in [3]. Military circles were aware of EMSEC since discovery of crosstalk between telegraph lines on the British army expedition to the Nile and Suakin in 1884–1885. During World War I, earth-return telephones used at the front could be eavesdropped by the other side at a distance of hundreds of yards—which became a serious vulnerability. Nowadays "Tempest" emissions standards govern military equipment design.

EMSEC attacks can be either active or passive. Active glitching attacks (such as described earlier) 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.

DPA is, by contrast, a passive attack: only the power consumption of a device is measured while it performs cryptographic computations (typically several hundred) with different data. We then guess the value of some bit in the computation (a key bit, or an intermediate value) and sort our power traces into two piles, depending on whether our target bit would have activated some circuit in the processor. For example, the carry function may use a measurable amount of power, and we thus sort our traces on whether, after the first round of a cipher, our guessed key value would set the carry bit, when combined with the observed data. We check our guess by observing whether the two piles are statistically different.

This simple signal-processing trick was devastatingly effective. Key material could be rapidly extracted from most of the smart cards then on the market, without any detailed knowledge of a particular algorithm's implementation. Unlike previous noninvasive attacks, DPA did not require a full-scale invasive attack to find the vulnerability—it was within the scope of a single moderately clueful graduate student. The consequences of this attack cut even deeper as electronic payment systems were already being designed with smart card components.

The turn of the century saw much research in power analysis defense. The challenge was to create defences which did not trade off resistance against power analysis with ease of timing analysis. The possible defenses include the following.

1) *Hardware measures*: Randomization of execution can help, for example, inserting an NOP at random in the instruction stream with probability 1 in 64; random precharging of the bus can also help. Balanced logic is a further option, discussed in the next section. The current research frontier is design-time validation: developing tools that enable a chip maker to identify EMSEC attacks early in the design stage, so that changes can be made before the chip is fabricated.
2) *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 $m = c^{d_1} c^{d_2} \pmod N$. In practice one has to do rather more than this, and randomizing ciphers other than RSA can be hard work.
3) *Protocol measures*: If the designer controls the protocol as well as the platform, an effective measure is to randomize all encryption operations and protocol messages; thus the opponent never knows both plaintext and ciphertext of any cipher operation. Regular key changes make any form of cryptanalysis—including power analysis—harder.

Of course the attack technology has advanced too: randomized NOP instructions, for example, may be countered if the attacker writes software to realign and correlate the collected traces. Perhaps the most radical advance, though, is Differential Electromagnetic Analysis, by Samyde and Quisquater [55]. Here, a small coil or magnetic sensor is brought to the surface of the chip itself. The attacker thus sees a local signal correlated with the power drawn from a subcomponent of the chip, yielding significantly more information.

C. Semi-Invasive Attacks

Between the invasive and noninvasive attacks lies a third class of local attacks, which we christened *semi-invasive attacks* [57]. These attacks involve access to the chip surface, but do not require penetration of the passivation layer, or direct electrical contact with the chip. The earliest semi-invasive attacks used UV light to reset the protection bit on microcontrollers, so that the memory contents could be read out [8]. Another early development was the LIVA/LECIVA semiconductor-testing technology developed by Sandia, in which illumination of nonconducting CMOS transistors causes measurable current leakage [1]. Boneh and others also speculated that revealing faults might be induced by subjecting smart cards 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].

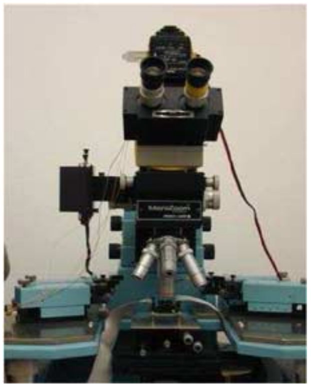
Fig. 6. Low-cost probing workstation and photoflash.

Practical semi-invasive attacks emerged a few years ago when we pioneered optical probing to inject security faults into digital circuits. By illuminating a target transistor it is induced to conduct, creating a transient fault. We need only simple, low-cost equipment: a photographer's flash gun mounted on the camera port of our microscope (Fig. 6) suffices. Using this, we were able to set or reset any individual bit of SRAM in a microcontroller [57].

Laser probing equipment yields even better results. RAM contents can be set to desired values, other memory technologies such as Flash and EEPROM can be read, and one can interfere with control logic directly. Detailed information on semi-invasive attacks is presented in [59]. Laser probes can also be used passively. If we 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, and thus we can read the memory contents. Laser pointers can even be used in place of solid-state lasers [56].

Defensive programming techniques may resist by performing some suitable alarm function, on attack detection. 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 may require ion-etching equipment to thin the device and an infrared laser to penetrate the silicon substrate.

D. How Smart Cards Differ From Commodity Microcontrollers

The growing demand for better protection has yielded smart cards specifically designed to protect against most of the known attacks. Thus, there is a sharp bifurcation in the market between low-cost microcontrollers offering rudimentary protection and higher cost smart cards in which serious security effort has been invested. High-end protection features now include internal voltage sensors to protect against power glitch attacks; clock frequency sensors to prevent attackers slowing down the clock for static analysis or raising it for clock-glitch attacks; top-metal sensor meshes mentioned above; internal bus hardware encryption to make

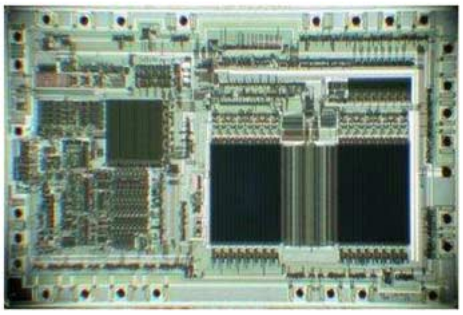

Fig. 7. MC68HC705PA microcontroller with clearly distinguishable blocks.

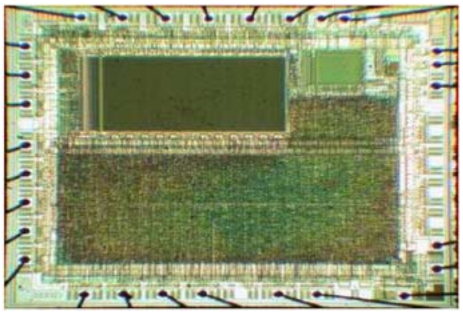

Fig. 8. SX28 microcontroller with "glue logic" design.

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 no longer work.

Standard building-block structures as in Fig. 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, are now increasingly randomized, yielding a tangible improvement. The resulting "glue logic" (Fig. 8) makes it virtually impossible to find signals manually for physical probing. An attacker would need automated probing tools or, *in extremis*, have to reverse engineer the entire circuit from micrographs.

Attacking smart cards, especially recently designed ones, is now an expensive and time-consuming task, possible only at well-equipped laboratories with highly qualified engineers. However, progress in lower cost attack technology constantly forces chipmakers to improve their products. For example, the recently discovered optical fault-injection attacks [57] revealed many problems in some designs as they let an attacker change the state of any transistor on the chip. Nanotechnology advances bring still more useful tools. For example, scanning capacitance microscopy lets us measure the threshold level of each individual transistor, allowing data recovery from memory technology 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 smart card 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 HSM still has several critical advantages over the smart card: it can contain large capacitors to filter I/O and power channels, and it can contain a tamper-sensing barrier. In particular, it can contain an internal power supply that can monitor the tamper-sensing barrier constantly, and actively destroy key material if a penetration attempt is detected. There have been some ingenious attempts to develop chip-level tamper reaction which does not require power—for example, by coating the chip in a chaotic matrix of magnetic fibers, whose emergent characteristics store a master key with which the memory contents are encrypted, but there exist no deployed products with this technology yet.

V. Remote Attacks

The last type of attack is the remote attack—independent of the distance between the attacker and the cryptoprocessor. The attacker merely needs access to the API of the device. Remote attacks can be passive—the encrypted transaction stream is observed and analyzed—or active, where 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 the 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].

An extension of protocol analysis is *API analysis*, which is specific to cryptoprocessors. 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.

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 extends a *cryptographic API* by enforcing *policy* on the interactions as well as providing cryptographic services.

A. *Introducing API Attacks*

The design and use of cryptoprocessors came into the open in "Why Cryptosystems Fail" [4]. Anderson focused on the known failure modes of ATM banking systems. For instance, he described a large U.K. bank that failed to bind the encrypted PIN to the account stored on the magnetic stripe. The criminal fraternity later discovered that by changing the account number on your own card you could use your own PIN to loot another's account. The 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 bind PINs to customer account numbers (PANs) through encryption with a secret key and the addition of a public offset. Thus, when one bank restructured their PANs, they commissioned a transaction to recalculate the stored offsets in the database, transparent to the customer. As a side effect, this transaction would produce the offset between any pair of PINs. An astute programmer later noticed he could calculate the PINs of others in relation to his personal PIN. At the time, this was characterized 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, and 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*.

B. Pure API Attacks

Shortly afterwards, an attack on the "VISA Security Module" (VSM) was discovered. The VSM financial security module had to interoperate 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

$$\textbf{(Generate Component) x2}$$

$$\begin{array}{ll} C \longrightarrow \text{Printer} & : \text{TMK1} \\ C \longrightarrow \quad U & : \{\text{TMK1}\}_{\text{Km}} \\ C \longrightarrow \text{Printer} & : \text{TMK2} \\ C \longrightarrow \quad U & : \{\text{TMK2}\}_{\text{Km}}. \end{array}$$

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 $(\text{Km}_1, \ldots, \text{Km}_n)$, and other keys were passed in as arguments to each transaction encrypted under one of these master keys (e.g., $\{K1\}_{\text{Km}}$). 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

$$\textbf{The attack}$$

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

$$\text{TMK1} \oplus \text{TMK1} = 0.$$

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 found a key separation attack on the VSM. PDKs and Terminal Master Keys (TMKs, the master keys held at ATMs) were considered to have the same type, and thus were encrypted with the same master key Km. There existed a command for the clear entry of a Terminal Communication key (TC) and a command to send these TC keys to an ATM that resulted in the TC key being encrypted under a TMK ($\{\text{TC}\}_{\text{TMK}}$). If these commands were abused and a PAN subsitiued for a clear TC key while a PDK was substituted for a TMK, then one obtained a PAN encrypted under a PDK ($\{\text{PAN}\}_{\text{PDK1}}$).

$$\textbf{(Enter clear TC Key)}$$

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

$$\textbf{(Send TC Key to ATM)}$$

$$\begin{array}{ll} U \longrightarrow C: & \{\text{PAN}\}_{\text{Km2}}, \{\text{PDK1}\}_{\text{Km}} \\ C \longrightarrow U: & \{\text{PAN}\}_{\text{PDK1}}. \end{array}$$

Of course, $\{\text{PAN}\}_{\text{PDK1}}$ 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?

C. Cryptographic API 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 [48], [49]; the detailed reference is [38]). 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

$$\begin{aligned} &(\textbf{Generate Check Value}) \\ U \longrightarrow C: &\quad \{\text{TMK1}\}_{\text{Km}} \\ C \longrightarrow U: &\quad \{0\,000\,000\,000\,000\,000\}_{\text{TMK1}}. \end{aligned}$$

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 that 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 [52]. 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. Time-memory tradeoffs and parallel key search are well known [30], [31]; however, this was the first real-world application of these techniques to extract keys.

Once DES became 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. "Replicate" keys with both halves the same gave legacy support for DES, as two of the 3DES encryptions cancel $(E(K1, D(K1, E(K1, P))) = E(K1, P))$. This led to an attack we discovered on the CCA, which did not bind the halves of these 3DES keys together. Exchanging halves of two known replicate keys (X and Y) would then yield a full 3DES key, which could be used to export other intrinsically valuable keys

$$\begin{aligned} &(\textbf{Generate Replicate})\ \textbf{x2} \\ C \longrightarrow U: &\quad \{X\}_{\text{Km}\oplus\text{left}}, \{X\}_{\text{Km}\oplus\text{right}} \\ C \longrightarrow U: &\quad \{Y\}_{\text{Km}\oplus\text{left}}, \{Y\}_{\text{Km}\oplus\text{right}} \\ \text{Known key}: &\quad \{X\}_{\text{Km}\oplus\text{left}}, \{Y\}_{\text{Km}\oplus\text{right}}. \end{aligned}$$

The key binding attack reduced the CCA's 3DES from 2^{112} to 2^{57} (only twice as good as single DES), and the time-memory tradeoff reduced further to about 2^{42}. We describe multiple completions of similar attacks using this technique in [2], [15], [20], and [26]–[28], as well as a number of other pure and cryptographic API attacks.

D. *Information Leakage Attacks*

The collection of pure API attacks relate to weaknesses in the *key management architecture* of HSMs rather than the application specific functionality. Closer examination of PIN processing yielded discovery of information leakage at the API level [28].

PIN format attack: Consider the following classic attack against the ISO-0 PIN block encryption standard. ISO-0 formats a PIN as 16 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 $\{P \oplus A\}_{\text{PEK}}$ where PEK is the key, P the formatted PIN buffer, and A the account number

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

Whenever the encrypted PIN is verified or reencrypted 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 calls 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.

At first glance, the integrity check seems sensible 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. 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 digits of the PIN and to determine the exact value from the set of $\{P, P \oplus 1\}$ [28].

Differential protocol attack: The decimalization table attack [5], [17], [20], [28] exploits the way that PINs are generated and verified. The PAN is encrypted with a 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 decimalization table ($dec1$), usually having the specific value shown below:

Hexadecimal Digit	0123	4567	89AB	CDEF
Mapped Decimal Digit	0123	4567	8901	2345

In this example, the hex character `0` is mapped onto `0`, while `A` is also mapped onto `0`. This $dectab$ is written as `0123456789012345`. 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

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

The problem is that the decimalization 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 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., $\{A\}_{\text{PDK}}$) does not contain a zero hexadecimal digit in the first four hexadecimal digits, then the 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 ($offset$ = customer PIN − generated PIN $(mod\ 10)$). This offset is stored in the bank's master database. Verification with offsets proceeds as follows:

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

This function remains vulnerable to the previous attack. Given an valid encrypted PIN block and its corresponding account, we can again manipulate the decimalization table to learn the PIN. Suppose we modify the usual decimalization table from `0123456789012345` to `1123456789012345`, while keeping the other inputs A, $\{P \oplus A\}_{\text{PEK}}$ and *offset* constant. If the PINs do not match and the verification fails, we learn that $\{A\}_{\text{PDK}}$ 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.

E. *Statistical Attacks*

An unfortunate consequence of the decimalization 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 improve the quality of an outsider attack on a German banking scheme in [44]. We consider the risks of an insider exploiting such a weakness: suppose the previous PIN generation command 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 decimalization 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

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

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 are known, coupled with the knowledge of the frequency of occurence of each block, we can identify a particular block (and thus all blocks) uniquely.

ISO-0 attack: In fact, the ISO-0 format above leaks information regardless of the algorithm used to generate PINs, or the uniformity of PINs, due to the limited range of possible values. Consider the possible encrypted PIN blocks for a fixed PAN— $\{P \oplus A\}_{\text{PEK}}$. Since PINs must be decimal, we will observe an incomplete set of possible hexadecimal values $P \oplus A$, regardless of the choice of A. Thus, comparing whole sets of blocks across different A we can determine information about P [18].

VI. 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. Although the keys remain secret, 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 multiparty 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 originally secure; 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?

Evaluation and certification: One response is to look for evaluations by third parties. There are two schemes under which cryptoprocessors are certified—FIPS 140 [35], run by the U.S. National Institute of Standards and Technology, and the Common Criteria [29], 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 modeling entailed in validating the 4758's firmware in [60]–[62].

The Common Criteria are used to evaluate products according to "protection profiles"—formalized 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.

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 1980s, automating the analysis of "key management systems" (essentially the security APIs of the day) initially using expert systems and later PROLOG [45]–[47], [54], 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.

We have gone on to reexamine 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 [53], [66], [69]. This is the subject of ongoing research in a Cambridge-MIT Institute research program. Meanwhile, recent work by Ganapathy *et al.* [36] on automated discovery of API-level vulnerabilities has modeled type-casting attacks [21] on the IBM CCA. We describe some of the challenges facing the protcol analysis community in extending their tools to analyze security APIs in [19].

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 e-mails 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? Competition and trade policy issues [9] arise in the support of restrictive business models. For instance, the tiny cryptoprocessors embedded in ink cartridges to authenticate them to the printer as coming from the same company enables printer vendors to subsidize the sales of printers by charging more for ink. Such usage is controversial: the EU has responded with an environmental directive requiring all ink cartridges to be refillable by 2007.

The spread of TC 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 [51].

VII. Conclusion

We have surveyed cryptoprocessors and their applications. Low-cost and midcost cryptoprocessors are the most rapidly developing area, with TC 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 here for the academic researcher.

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.

References

[1] C. Ajluni, "Two new imaging techniques promise to improve IC defect identification," *Electron. Des.,* vol. 43, no. 14, pp. 37–38, Jul. 1995.

[2] R. Anderson, "The correctness of crypto transaction sets," presented at the 8th Int. Workshop Security Protocols, Cambridge, U.K., 2000.

[3] ——, *Security Engineering—A Guide to Building Dependable Distributed Systems*. New York: Wiley, 2001.

[4] ——, "Why cryptosystems fail," *Commun. ACM* vol. 37, no. 11, pp. 32–40, Nov. 1994 [Online]. Available: http://www.cl.cam.ac.uk/users/rja14/wcf.html, earlier version.

[5] R. Anderson and M. Bond, "Protocol analysis, composability and computation," in *Computer Systems: Papers for Roger Needham*. Cambridge, U.K.: Microsoft Research, 2003, pp. 7–10.

[6] R. Anderson and S. Bezuidenhoudt, "On the reliability of electronic payment systems," *IEEE Trans. Softw. Eng.* vol. 22, no. 5, pp. 294–301, May 1996 [Online]. Available: http://www.cl.cam.ac.uk/ftp/users/rja14/meters.ps.gz

[7] R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," in *Security Protocols*. Heidelberg, Germany: Springer-Verlag, 1997, vol. 1361, Lecture Notes on Computer Science, pp. 125–136.

[8] ——, "Tamper resistance—A cautionary note," in *Proc. 2nd USENIX Workshop Electronic Commerce* Nov. 1996, pp. 1–11.

[9] R. Anderson, "Cryptography and competition policy," in *Economics of Information Security*, L. J. Camp and S. Lewis, Eds. Norwell, MA: Kluwer, 2004, pp. 35–52.

[10] ——, Trusted Computing FAQ. [Online]. Available: http://www.cl.cam.ac.uk/ rja14/tcpa-faq.html

[11] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor, "Improving smart card security using self-timed circuits," Asynch, 2002.

[12] S. Moore, R. Anderson, R. Mullins, G. Taylor, and J. Fournier, "Balanced self-checking asynchronous logic for smart card applications," *Microprocess. Microsyst. J.,* vol. 27, no. 9, pp. 421–430, Oct. 2003.

[13] HP/Atalla, "Atalla A8000NSP Banking Command Reference Manual," 2001.

[14] E. Biham and A. Shamir, "Differential fault analysis: A new cryptanalytic attack on secret key cryptosystems," in *Advances in Cryptology—CRYPT0 '97*. Heidelberg, Germany: Springer-Verlag, 1997, vol. 1294, Lecture Notes in Computer Science, p. 513, .

[15] M. Bond, "Attacks on cryptoprocessor transaction sets," in *Cryptographic Hardware and Embedded Systems—CHES 2001*. Heidelberg, Germany: Springer-Verlag, vol. 2162, Lecture Notes in Computer Science, pp. 220–234.

[16] M. Bond and R. Anderson, "API-level attacks on embedded systems," *IEEE Computer,* vol. 34, no. 10, pp. 67–75, Oct. 2001.

[17] M. Bond and P. Zielinski, "Decimalization table attacks for PIN cracking," Univ. Cambridge Computer Laboratory, Tech. Rep. TR-560.

[18] M. Bond and J. Clulow, "Encrypted? Randomised? Compromised? (When cryptographically secured data is not secure)," presented at the Cryptographic Algorithms and Their Uses, Eracom Workshop 2004, Gold Coast, Qld., Australia.

[19] ——, "Extending security protocols analysis: new challenges," presented at the Automated Reasoning and Security Protocols Analysis (ARSPA) 2004, Cork, Ireland.

[20] M. Bond, "Understanding security APIs," Ph.D. dissertation, Univ. Cambridge, Cambridge, U.K., Jan. 2004.

[21] M. Bond*, A Chosen Key Difference Attack on Control Vectors* Nov. 1, 2000, unpublished.

[22] D. Boneh, R. A. Demillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in Advances in Cryptology—Eurocrypt '97 Heidelberg, Germany, Springer-Verlag, vol. 1294, Lecture Notes in Computer Science, pp. 37–51.

[23] S. Brands, "Untraceable off-line cash in wallet with observers," Crypto '93 Heidelberg, Germany, Springer-Verlag, vol. 773, Lecture Notes in Computer Science, pp. 302–318.

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

[25] D. Chaum, "Blind signatures for untraceable payments," in *Proc. Crypto '82* 1983, pp. 199–203.

[26] R. Clayton and M. Bond, "Experience using a low-cost FPGA design to crack DES keys," in *Cryptographic Hardware and Embedded Systems—CHES 2002*. Heidelberg, Germany: Springer-Verlag, vol. 2523, Lecture Notes in Computer Science, pp. 579–592.

[27] J. Clulow, "On the security of PKCS#11," in *Cryptographic Hardware and Embedded Systems—CHES 2003*. Heidelberg, Germany: Springer-Verlag, vol. 2779, Lecture Notes in Computer Science, pp. 411–425.

[28] ——, "The design and analysis of cryptographic APIs for security devices," M.Sc. thesis, University of Natal, Durban, South Africa, 2003.

[29] Common Criteria Evaluation Scheme [Online]. Available: http://www.commoncriteriaportal.org/

[30] Y. Desmedt, "Breaking one million DES keys," in *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. Sebastopol, CA: O'Reilly, 1987, ch. 9.

[31] W. Diffie and M. Hellman, "Exhaustive cryptanalysis of the NBS Data Encryption Standard," *Computer,* vol. 10, no. 6, pp. 74–84, Jun. 1977.

[32] J. Dyer, R. Perez, S. W. Smith, and M. Lindemann, "Application support for a high-performance, programmable secure coprocessor," presented at the 22nd Nat. Information Systems Security Conf., Arlington, VA, 1999.

[33] J. Dyer, M. Lindemann, R. Perez, R. Sailer, S. W. Smith, L. van Doorn, and S. Weingart, "Building the IBM 4758 secure coprocessor," *IEEE Computer,* vol. 34, no. 10, pp. 57–66, Oct. 2001.

[34] EMV 4.1, "Integrated circuit card specifications for payment systems" Jun. 2004 [Online]. Available: http://www.emvco.com

[35] *Security requirements for cryptographic modules*, NIST Federal Information Processing Standards 140-1, 140-2 [Online]. Available: http://csrc.nist.gov/cryptval/

[36] V. Ganapathy, S. A. Seshia, S. Jha, T. W. Reps, and R. E. Bryant, "Automatic discovery of API-level vulnerabilities" Univ. Wisconsin-Madison Computer Sciences, Tech. Rep. UW-CS-TR-1512, Jul. 2004 [Online]. Available: http://www.cs.wisc.edu/vg/writings/papers/tr1512.pdf

[37] IBM, "IBM 3614 consumer transaction facility implementation planning guide," IBM Doc. ZZ20-3789-1, 2nd ed., Dec. 1977.

[38] IBM, IBM 4758 PCI Cryptographic Coprocessor—CCA basic services reference and guide, Release 1.31 for the IBM 4758-001, 1999.

[39] IBM enhanced media management system [Online]. Available: http://www-306.ibm.com/software/data/emms/

[40] J. King, "Bolero—A practical application of trusted third party services," *Comput Fraud Secur. Bull.,* pp. 12–15, Jul. 1995.

[41] P. Kocher, "Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems," in *Advances in Cryptology—Crypto '96*. Heidelberg, Germany: Springer-Verlag, vol. 1109, Lecture Notes in Computer Science, pp. 104–113.

[42] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—Crypto '99*. Heidelberg, Germany: Springer-Verlag, vol. 1666, Lecture Notes in Computer Science, pp. 388–397.

[43] O. Kommerling and M. Kuhn, "Design principles for tamper-resistant smartcard processors," in *Proc. USENIX Workshop Smartcard Technology* 1999, pp. 9–20.

[44] M. Kuhn, "Probability theory for pickpockets—ec-PIN guessing" [Online]. Available: www.cl.cam.ac.uk/~mgk25/ec-pin-prob.pdf

[45] D. Longley and S. Rigby, "An automatic search for security flaws in key management," *Comput. Secur.,* vol. 11, pp. 75–89, Mar. 1992.

[46] D. Longley, "Expert systems applied to the analysis of key management schemes," *Comput. Secur.,* vol. 6, no. 1, pp. 54–67, Feb. 1987.

[47] D. Longley and S. Vasudevan, "Effects of key generators on the automatic search for flaws in key management schemes," *Comput. Secur.,* vol. 13, no. 4, pp. 335–347, 1994.

[48] S. M. Matyas, "Key handling with control vectors," *IBM Syst. J.,* vol. 30, no. 2, pp. 151–174, 1991.

[49] S. M. Matyas, A. V. Le, and D. G. Abraham, "A key management scheme based on control vectors," *IBM Syst. J.,* vol. 30, no. 2, pp. 175–191, 1991.

[50] "Microsoft next-generation secure computing base (NGSCB)" [Online]. Available: http://www.microsoft.com/resources/ngscb/default.mspx

[51] A. M. Odlyzko*, The Internet and 19th century railways* 2003, unpublished.

[52] P. van Oorschot and M. Wiener, "Parallel collision search with applications to hash functions and discrete logarithms," in *Proc. 2nd ACM Conf. Computer and Communications Security* 1994, pp. 210–218.

[53] Otter—An automated deduction system [Online]. Available: http://www-unix.mcs.anl.gov/AR/otter

[54] S. Rigby, "Key management in secure data networks," M.Sc. thesis, Queensland Inst. Technol., Brisbane, Qld., Australia, 1987.

[55] D. Samyde and J. Quisquater, "Electromagnetic analysis (EMA): Measures and counter-measures for smartcards," in *Smart Card Programming and Security*. Heidelberg, Germany: Springer-Verlag, 2001, vol. 2140, Lecture Notes in Computer Science, pp. 200–210.

[56] D. Samyde, S. Skorobogatov, R. Anderson, and J. Quisquater, "On a new way to read data from memory," in *Proc. 1st Int. IEEE Security in Storage Workshop (SISW 2002)* pp. 65–69.

[57] S. Skorobogatov and R. Anderson, "Optical fault induction attacks," in *Cryptographic Hardware and Embedded Systems Workshop, CHES 2002*. Heidelberg, Germany: Springer-Verlag, vol. 2523, Lecture Notes in Computer Science, pp. 2–12.

[58] S. Skorobogotov, "Low temperature data remanence in static RAM" Univ. Cambridge Computer Laboratory, Tech. Rep. TR-536 [Online]. Available: http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-536.pdf

[59] ——, "Semi-invasive attacks—A new approach to hardware security analysis," Ph.D. dissertation, Univ. Cambridge, Cambridge, U.K., Sep. 2004.

[60] S. W. Smith and V. Austel, "Trusting trusted hardware: Toward a formal model for programmable secure coprocessors," presented at the 3rd USENIX Workshop Electronic Commerce, Boston, MA, 1998.

[61] S. W. Smith, R. Perez, S. H. Weingart, and V. Austel, "Validating a high-performance, programmable secure coprocessor," in *Proc. 22nd Nat. Information Systems Security Conf.* 1999 [Online]. Available: http://csrc.ncsl.nist.gov/nissc/1999/proceeding/papers/p16.pdf

[62] S. W. Smith, "Outbound authentication for programmable secure coprocessors," in *Computer Security—ESORICS 2002*. Heidelberg, Germany: Springer-Verlag, 2002, vol. 2502, Lecture Notes in Computer Science, pp. 72–89.

[63] S. W. Smith, E. R. Palmer, and S. Weingart, "Using a high-performance, programmable secure coprocessor," in *Financial Cryptography*. Heidelberg, Germany: Springer-Verlag, 1998, vol. 1465, Lecture Notes in Computer Science, pp. 73–89.

[64] S. W. Smith and S. Weingart, "Building a high-performance, programmable secure coprocessor," *Comput. Netw. (Special Issue on Computer Network Security),* vol. 31, pp. 831–860, Apr. 1999.

[65] S. W. Smith, "Fairy dust, secrets and the real world," *IEEE Security Privacy,* vol. 1, no. 1, pp. 89–93, Jan/Feb. 2003.

[66] "SPASS—An Automated theorem prover for first-order logic with equality" [Online]. Available: http://spass.mpi-sb.mpg.de

[67] Trusted Computing Group, "Trusted platform module specifications v1.2" [Online]. Available: https://www.trustedcomputinggroup.org

[68] S. Weingart, "Physical security devices for computer subsystems: A survey of attacks and defenses," in *Cryptographic Hardware and Embedded Systems—CHES 2000*. Heidelberg, Germany: Springer-Verlag, 2000, vol. 1965, Lecture Notes in Computer Science, pp. 302–317.

[69] P. Youn, "The analysis of cryptographic APIs using formal methods," M.S. thesis, Massachusetts Inst. Technol, Cambridge, Jun. 2004.

Ross Anderson received the BA, MA and PhD degrees from the University of Cambridge, Cambridge, U.K., in 1978, 1982, and 1995, respectively.

He is Professor of Security Engineering at the University of Cambridge. His research interests include cryptographic processors, peer-to-peer systems, security usability, and security economics.

Mike Bond is a Research Associate in the security group at the University of Cambridge Computer Laboratory, Cambridge, U.K. His research interests include the design and analysis of security APIs—the software interfaces to hardware security modules. His recent work has focused in particular on APIs for Trusted Computing and financial transaction security.

Jolyon Clulow is currently working toward the Ph.D. degree in the security group at the Computer Laboratory, University of Cambridge, Cambridge, U.K. His research interests include security APIs, secure coprocessors, financial security, and cryptography.

Sergei Skorobogatov received the M.Sc. degree in physics from Moscow Engineering Physics Institute (Technical University), Russia, in 1997 and the Ph.D. degree in computer science from the University of Cambridge, Cambridge, U.K., in 2005.

He is currently a Research Associate at the University of Cambridge, where he has held research positions since 2000. His research interests include hardware security evaluation and improvement in microcontrollers and FPGAs, implementation and use of optical fault injection attacks on semiconductor devices.