The Design and Analysis of Cryptographic Application Programming Interfaces for Security Devices

Version: 4.0

Issue Date: 2003-01-17

The Design and Analysis of Cryptographic Application Programming Interfaces for Security Devices

by

Jolyon Clulow

A dissertation
presented to the University of Natal, Durban
in fulfilment of the
requirements for the degree of
Master of Science
in
Mathematics.

Durban, South Africa, 2003

Abstract

The significance of security devices that protect the numerous transactions, which take place in today's distributed virtual environment, cannot be underestimated. The importance of such devices will increase as our society continues to evolve into a cashless electronic society.

The continuing increase in the number and size of electronic transactions, the advances in the technology utilized and the growing sophistication of the adversary have led to significant resources being invested in the evaluation and analysis of security devices. There has been a transformation of the traditional security analysis from one focused on mathematical primitives and physical engineering solutions to a holistic approach that seeks to protect against subtle interactions between the cryptographic, logical and physical aspects of such devices that can collude to compromise the security thereof.

In the above setting, this dissertation investigates the electronic interface to security devices (i.e., the application programming interface or API) as a source of vulnerabilities. A number of innovative attacks are presented with significant implications for our financial transaction systems (e.g. ATM networks) that challenge previous assertions of their security, integrity and robustness. Finally, some design solutions and criteria are presented.

Dedication

To my parents.

Acknowledgements

I am greatly indebted to my supervisor Professor H.C. “Henda” Swart for her guidance, enthusiasm, encouragement and, most importantly, her genuine friendship. It has been a real honour to work with a person of her stature.

This dissertation has been shaped by the years spent working in the Crypto Business Units of Nanoteq (Pty) Ltd and subsequently Prism Payment Technologies (Pty) Ltd (Prism acquired the division from Nanoteq). I am grateful to Dave Ritten, David Joubert and Richard Scott for the various roles they played in my personal development and their friendship. (You almost succeeded in turning me into a true engineer ☺.) I would also like to thank the rest of my colleagues and friends in the team over the years, including

Anna Maciel
Anton Strydom
Avinash Samlall
Billy Bharath
Brando Harilal
Brenda Math
Carl Meijer
Charles Louw
Colleen Thomas
Collin Chetty
Craig Kirk
Curtis Kemraj
Damian Budd
Dave Ritten
David Joubert
Douglas Patterson
Eugene Aliphon
Gary Rolfe
Gerhard Claassen
Giovanni Gallus
Jackie Tluczek
Joy Hillis
Jyri Hamalainen
Karl Richardson
Kishore Ramparthab
Laven Naidoo
Leon Fouche
Louise Tromp
Mark Currie
Mark Strydom
Matthew Hayhurst
Mose Jali
Patricia Mthembu
Peter Norbury
Radha Moodley
Richard Pitchers
Richard Scott
Rod Ladwig
Selvan Naidoo
Shane Nicholas
Shawn O'Neill
Sipho Ndlovu
Trevor Davel
Veni Pragasam
Werner Theron

Table of Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction and Literature Review

1.1. Introduction

Security devices are becoming increasingly pervasive in our modern society. Indeed, on closer examination of our interaction with both the physical and electronic world, one discovers numerous instances where we make use of electronic or virtual security devices (e.g. access tokens, electronic keys, gate remotes, subscriber identification modules (SIM cards), bank cards, credit cards and debit cards). Traditional security techniques have been augmented (and often completely replaced) by virtual methods. For example, traditional authentication by means of a written signature or face-to-face contact has evolved to include the use of digital signatures and biometrics. These changes are almost transparent, perhaps owing to the fact that the mechanisms employed often closely resemble the historical methods that we are accustomed to (e.g. an electronic key is conceptually identical to a physical key).

Perhaps the biggest change is the demise of our physical identities and the rise of our virtual personas. In our current Internet connected reality we can transact without a physical presence and through a distributed mechanism involving an eclectic collection of role players. These advances bring with them new security considerations, dangers and demands. Primarily the concern centres on our ability to authenticate and authorize a transaction, while at the same time ensuring the secrecy of the information (secrets) with which we achieve this. In the distributed scenario involving the use of infrastructure over which one has no control, one is particularly concerned about issues of trust in addition to the normal security requirements. The need to guarantee security in a hostile or untrustworthy environment is often also addressed through the use of security devices.

This work is predominantly concerned with the security of such devices – more specifically the *logical* security thereof. In order to reach the point were we can analyse the logical security in the general case and specific instances, we must first gain an understanding as to the design and workings of security devices.

1.2. Types and Examples of Security Devices

There is a fairly wide range of devices that could be described as security devices. These range from the low end, low cost ICC (Integrated Chip Card or Smart card) and smart buttons, to the more powerful and costly, secure coprocessors and tamper responding security modules.

Example 1. Access control
The Dallas iButton is typically used as an access control device to control entrance to physical locations such as buildings and offices. It possesses a mechanism to authenticate itself to the control software, which can then authorize access to the bearer. In a sense, it is a key that opens a lock by electronic, as opposed to physical, means. A typical operating procedure would dictate that a token (button) only be issued to authorized people and the control system be configured to limit what access can be achieved with the button. The user is expected to take reasonable precautions to safeguard the device against accidental loss or theft. Theft of the device is an issue until the user can report it to the control system administrator. It is common practice for users to attach the buttons to their key rings (more evidence that they are perceived as conceptually equivalent) and hence the buttons are potentially transported, and exposed, to a large number of environments. The button represents a 'difficult challenge' to an attacker, who would attempt to steal or duplicate it. Thus, the security design requirements are to ensure that the secrets used to authenticate the device remain secure inside the device.

For our purposes, we shall limit our discussions to two categories of security devices, namely Integrated Chip Cards (ICCs), more commonly referred to as smart cards, and secure crypto coprocessors.

1.2.1. Secure Coprocessors

A secure coprocessor can be known by a variety of names including:

- Tamper Resistant/Responding Security Module (TRSM),
- Crypto Accelerator,
- Network Security Processor (NSP),
- Host Security Module (HSM), and
- Hardware Security Module (HSM).

Essentially these devices provide a secure, trusted environment to perform sensitive operations. The exception is the crypto accelerator, which justifies its use purely on a performance basis by offloading the complex, time consuming cryptographic operations from the host system. These devices offer a degree of physical protection by being able to detect and respond to malicious attempts to recover key material or sensitive data. Typical measures include the use of a physical tamper envelope or membrane to detect physical intrusion, sensors for temperature and radiation as well as power supply monitoring and filtering. A detected tamper attempt causes the erasure of protected data. A thorough discussion on the physical security requirements is presented later.

Early work in building robust general-purpose computational environments for storing secrets was done by [Pa94, WC87, WWAP91]. Yee [Ye94] addressed the distributed security problem through the use of secure crypto coprocessors to protect the privacy of the data they process. Related work examined the use of such devices in e-commerce and related applications [TY93, YT95]. There are several commercially produced secure coprocessors [AEP, ATALLA, IBM, ERACOM, NCIPHER, PRISM, THALES, UTIMACO] with related marketing material available from the respective websites. This typically includes only a statement that the device is 'secure', justified by means of a security certification or simply an unsubstantiated statement that it is in compliance with the certification requirements. Unfortunately, there is little manufacturer-supplied documentation about the design and analysis of the devices. This is probably due to the commercial nature of these products and the dual concerns of protecting intellectual property and limiting the knowledge available to an attacker. The exception is the series of papers published in open academic literature by the design team members of the IBM 4758 secure coprocessor [Sm96, SM97, SA98, SPW98, SW99, DPSL99, SPWA99]. These covered topics such as the design of the hardware, software and cryptographic architecture, the formal methods used to verify the design and their experience in validating the design and implementation as part of a certification process.

1.2.2. A Simple Generalized Architecture

A simple design for a secure coprocessor consists of the following basic components:

- communications interface
- central processing unit (CPU)
- crypto accelerators (including noise generating circuits)
- general purpose memory
- memory for secrets

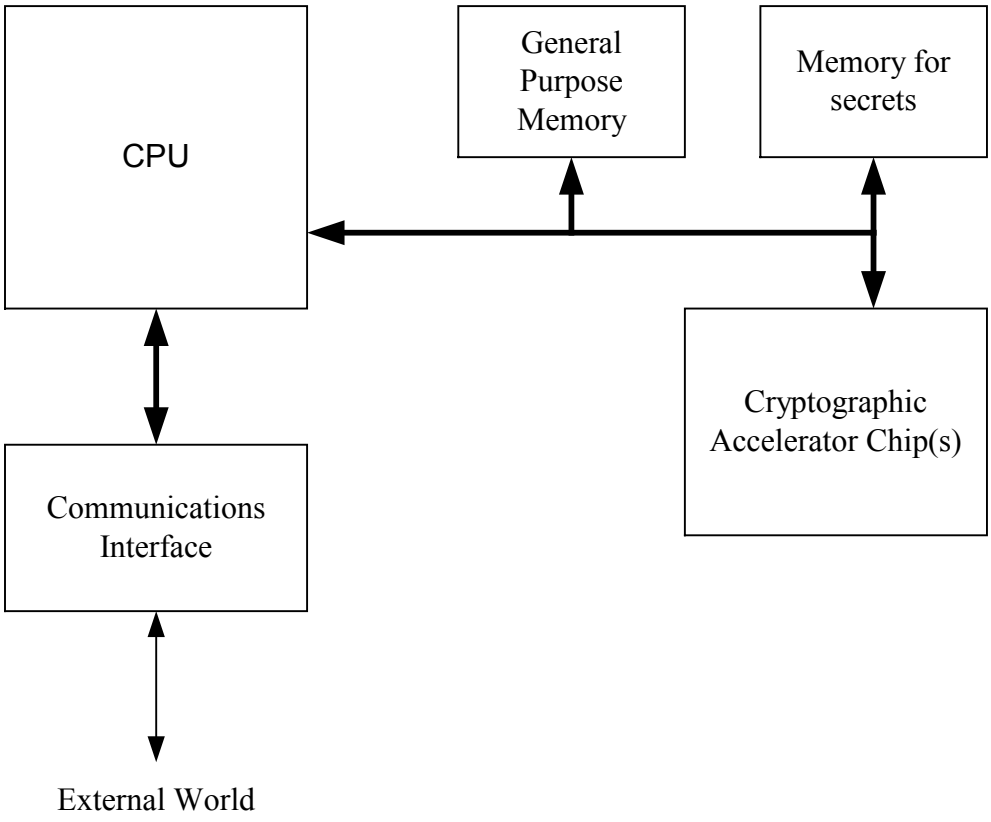

Figure 1-1: A simple design for a secure coprocessor

The CPU controls the general operation of the device. It communicates with the external world through the communications interface and makes use of memory to store program code, permanent information and temporary variables. Being a security device, the coprocessor is likely to contain some form of crypto accelerators, either as custom designed and built chips that implement the crypto or security algorithms in hardware or more general-purpose processors with optimised code implementations of the algorithms. The use of cryptographic chips is largely driven by performance requirements and the fact that crypto algorithms are typically computationally intensive. Hardware generation of noise for the seeding of random number generation may be required owing to the fact that software noise is at best pseudorandom. Designing a noise source that generates genuinely random noise can be challenging task in itself. In addition, there may be a special area of memory used for storing secrets and other sensitive data, which may require special characteristics such as the ability to actively detect secrets while on standby power. While this description implies a hardware solution (i.e., a physical device), one can conceive of a software only solution that executes on some host system.

1.2.3. Smart Cards

A smart card or ICC can be viewed as a low cost hardware security module and typically has less processing power and memory. Over the past decades they have increased from having 8-bit cores to fully fledged 32-bit CPUs with 128 KB of electrically erasable programmable memory (EEPROM) and a smaller amount of RAM. They have a simple interface containing power, ground and a single communications line typically operating at speeds of 9600bps to 14400bps.

1.3. Towards Defining the Security Goals

It is an axiom of security engineering that a system is only as strong as its weakest link. Despite this fact, it can be argued that real world systems typically fail to consider completely all aspects of security. Often a designer will focus on those areas about which he is knowledgeable or comfortable. This is perhaps a natural phenomenon. After all, one cannot consider topics or issues about which one is not aware. Towards the goal of describing a suitable security target, [We00] provides a useful and descriptive subdivision of security concerns relating to secure devices as follows:

- *physical security*, a barrier placed around a computing system to deter unauthorized physical access to the computing system itself,
- *logical security*, the mechanisms by which operating systems and other software prevent unauthorized access to data,
- *environmental security*, the protection the system receives by virtue of location such as guards, cameras, badge readers, access policies, etc.

This division of security can be represented diagrammatically as depicted in Figure 1-2.

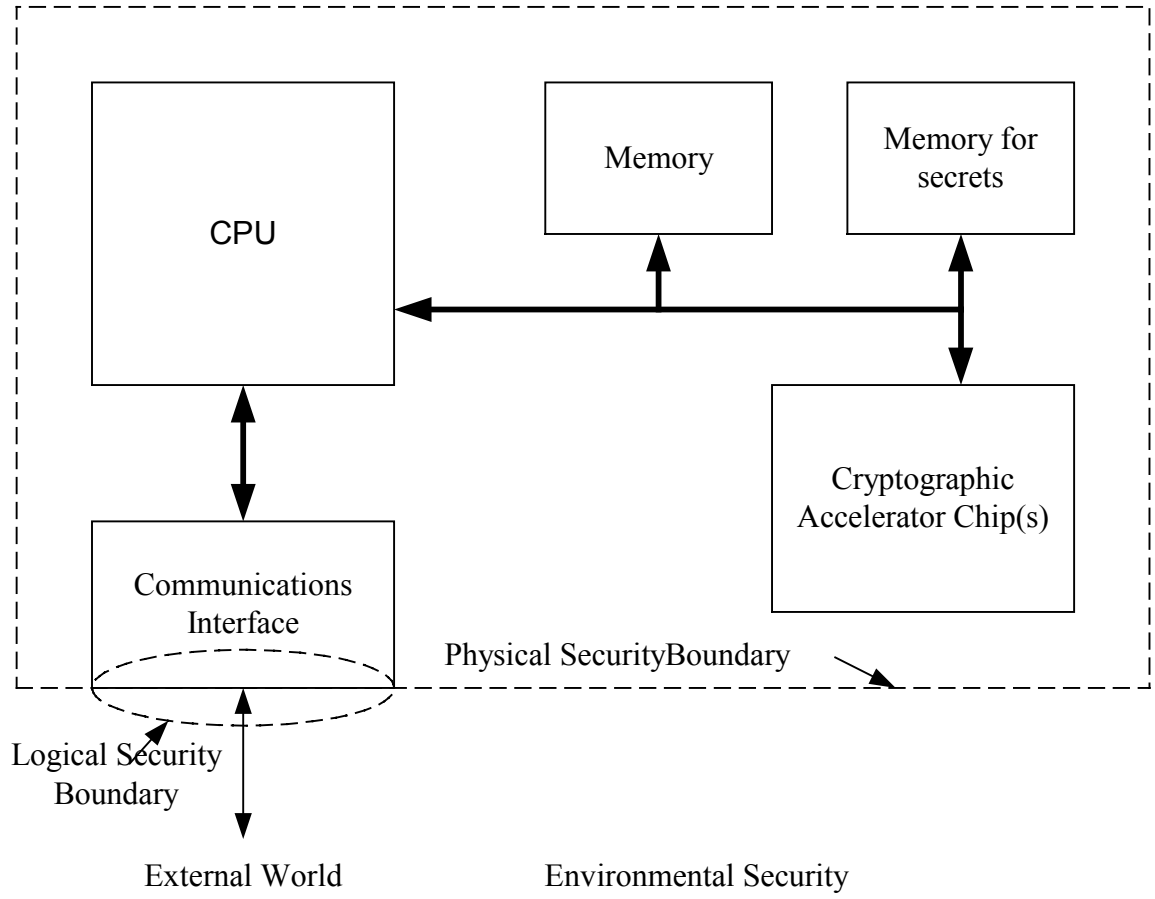

Figure 1-2: Security boundaries

1.4. Physical Security

1.4.1. Description and Goal

Physical security is the perhaps most obvious criterion when analysing the security of a physical device and yet (as described later) is perhaps the hardest to achieve and prove. Let us consider the attack model that physical security is defending against. Firstly one must assume that the attacker has access to the device as well as the ability to mount an attack (else there is no need for physical security). Normally one assumes that the attacker has unlimited time and opportunity to perform the attack and permits the attacker the luxury of transporting the device to his laboratory where he has unlimited access to specialized equipment. This is of course particularly true for low cost, mass produced and portable devices such as smart cards. One can parameterise attacks based on time, opportunity as well as cost. In reality, the absolute cost must be reasonable particularly when one compares the cost against the potential rewards from defeating the targeted security mechanism. Ideally, from the designer's perspective, the costs of mounting the attack should outweigh the

potential rewards. It, therefore, makes sense that weaker, more vulnerable devices should contain less valuable secrets, which represent only a limited potential exposure.

So what must be detected? Basically, any attempt to perform some unauthorized physical or electronic action to the device, in order to recover its secrets, is referred to as a tamper and must be prevented. [We00] includes a survey of the technologies available to both the attacker and defender that forms the basis for this discussion. The holy grail of a physical attack would be to gain access to the memory (or other chips), that contain secrets, and to read the values from these components. In order to access the internal circuitry, one may first have to remove several layers of covering material. Depending on the type of covering used, this can be achieved by manual methods (such as scraping with a knife), or by mechanical means (including the use of water, lasers and chemicals). Once exposed, the data in the memory can be easily recovered from the device by attaching probes. A more sophisticated attack is the use of energy probes (electron beams, ion beams or focused beams of light), which can read and write the contents of semiconductor storage and can thus recover secrets from a variety of semiconductor devices.

This represents a first pass description of the threats posed to physical security. It allows us to start identifying the threats and specifying appropriate defences. Any defence we envisage could itself be attacked and compromised. Hence, the development of the security requirements, in this sense, is an iterative process. It is a truism that the ever-advancing skills and increasing knowledge of adversaries will inevitably lead to the discovery of new weaknesses both of the systems we protect, as well as the mechanisms we use to do so. This condemns security engineering to be a continuing evolving discipline.

Physical security is often described in terms of its properties in the event of a tamper or attempted tamper (a physical attempt to circumvent physical protections and extract sensitive information or perform some action). These are:

- *tamper evidence,*
- *tamper resistance,*
- *tamper detection* and,
- *tamper response.*

However this is not a complete list of the security requirements. In addition, a device must

- not *leak* any secrets (or information or data),
- not allow the *modification* of secrets (or information or data), and

- be able to provide its functionality when expected (*availability* of service).

All of the above must be ensured throughout the lifecycle of the device including periods when the device is not under normal power.

1.4.2. Tamper Evidence

The goal of tamper evidence is to ensure that concrete evidence is left behind when a tamper occurs. For example, technologies include

- brittle packages which crack or shatter upon an attempted penetration, leaving evidence,
- bleeding paint, where paint of one colour is mixed with micro-balloons containing paint of a contrasting colour which rupture when the surface is scratched, resulting in visible 'bleeding', and
- seals and other similar 'trip wiring'.

1.4.3. Tamper Resistance

The obvious approach to gain resistance against penetration is the use of a protective boundary or hard exterior. Often the device is 'potted' inside an enclosure in a hard resistant resin or epoxy type compound with the intention that an attempt to remove either the enclosure or potting material will cause damage to the circuitry. In order to protect against a chemical attack that removes the resin, aluminium powder can be added to the potting compound. A solvent capable of dissolving the aluminium will corrode the underlying components or circuitry, rendering it useless. One remains unconvinced that an attacker would not be able to remove any protective layer with enough effort and patience. Thus tamper resistance works best with detection and response.

1.4.4. Tamper Detection

Typically, tamper detection is achieved through the use of an enclosing protective boundary, which is damaged in some way by an attempted penetration and in the process provides a signal or trigger. Early designs included winding wire around the device and then potting in a hard compound. The wire is monitored for continuity by the device (i.e., whether the wire is short circuited or open circuited). An attacker attempting to machine through the potting compound would break the wire. The device would detect the resulting open circuit. The continuity approach suffers from the

weakness that an attacker can short any two points on the wire (using another piece) and is then free to break the original wire between these two points. This leads to the development of resistive networks where, as the name suggests, the resistance of the wire is monitored for changes or variations outside a prescribed range. The latest in this line is the use of a flexible printed circuit sensor (often called a tamper foil). Here interleaving resistive networks are printed or silk screened onto a flexible foil that can then be folded and wrapped to create a nearly completely sealed package. Several layers can be printed onto the foil creating a three dimensional barrier. Often the foil and potting material are made from chemically similar materials in order that they bond together, becoming indistinguishable to the attacker.

1.4.5. Tamper Response

The obvious response to the detection of a tamper is the deletion of any secrets or sensitive information inside the device. For volatile memory, removing power or clamping the device to ground would naively seem to destroy the contents. One could perform a more active erasure by overwriting the existing contents of memory. This is required for non-volatile forms of memory such as EEPROM (electrically erasable, programmable read only memory).

Unfortunately, it transpires that this is not sufficient to destroy the secrets. Peter Gutmann in two intriguing papers [Gu96, Gu01] observes that "Contrary to conventional wisdom, volatile semiconductor memory does not entirely lose its contents when power is removed. Both static (SRAM) and dynamic (DRAM) memory retains some information…" In the case of SRAM, storing constant data over a prolonged period of time has the effect of altering the preferred power up state to the same as the state representing the stored data. [AK96, An01] document a case where the master key for a Visa security module was 'burnt' into the SRAM and found to be almost intact on power up. The error in bits was in the order of between 5 and 10% with the recoverability being aided by the parity of the DES key. It is not only areas such as RAM and non-volatile memory cells that are affected by data remanence problems, but also other aspects of semi-conductor devices through which secrets pass. This is due to a variety of observable physical effects including, but not limited to, hot-carrier effects (which change the characteristics of the semi-conductors in the device) and electromigration (which physically alters the device itself).

This leads to additional design criteria, aimed at avoiding retention effects, such as ensuring that no memory cell holds information longer than a set period of time, and processes aimed at healing the

cells. Perhaps the most practical solution is periodically moving the data or, alternatively, inverting it in storage.

The erasure of data is dependant on other external factors too. The period taken for certain types of memory to lose state when power is removed or clamped is affected by factors such as temperature. The lower the temperature, the longer the state persists. This leads to the attack strategy of lowering the temperature of the device before attempting a physical penetration, hopefully extending the window of opportunity available for the attacker to reach the memory, and reapplying power before its contents become irrecoverable. Thus, the detection strategy needs to include the monitoring of factors such as temperature to ensure the device is only exposed to an acceptable range. A similar concern is the use of X-rays or other radiation to burn the contents of memory into the device.

1.4.6. Leaking Secrets

The prehistory of emission security (EmSec) is traced back by Anderson [An01] to as early as World War I, when field telephone lines could be monitored using the phenomenon of 'cross talk' (i.e., interference caused by the energy from one telephone conversation invading another by electrostatic or electromagnetic coupling). This was achievable at a range of 100 to 300 yards depending on the communications technique (i.e., voice or Morse code) and was within the range of the enemy in the opposing trench. By the late 50's and early 60's, unintentional emissions were being used to compromise cryptographic equipment. In his memoirs [Wr87], former MI5 agent Peter Wright details an attempt to monitor French communication. The initial attempts to break the French diplomatic cipher failed. However, it was noticed that a faint secondary signal was carried along with the normal enciphered traffic. It turned out that this was the plaintext that had somehow leaked through the cipher machine! The prevention of 'compromising emissions' became an important (and classified) subject within the defence industry, leading to the creation of the TEMPEST program and requirements for TEMPEST shielded devices within the government and defence industries. Further information can be found on [Mc]. Due to the highly classified nature of the TEMPEST program, the topic disappeared from public literature. It made a stunning comeback when in 1985 a Dutch researcher Wim van Eck, published an article [Ec85] on how it was possible to reconstruct the image displayed on a Video Display Unit (VDU) at distance. This was achievable using simple, readily available equipment. The subject remains significant, especially with the advent of the digital age and the PC. Anderson and Kuhn [KA98] showed how compromising emissions from a PC could be made better or worse by software. Recent contributions from Kuhn [Ku02] detailed how, by monitoring the intensity of even diffuse reflections off a variety of sources (e.g. walls), the original image from a

display can be recovered. [LU01] described how status light emitting diodes (LEDs) on communications equipment leak the data being transmitted.

Over recent years, interest has increased in side channel attacks (defined in [KSWH00] as cryptanalysis using implementation data) against security devices. This has been for a number of reasons. Firstly, the discovery of a number of novel attacks including timing attacks [Ko96], Power Analysis [KJJ98, KJJ99], Fault Analysis [BDL97, BS97] and Electromagnetic Analysis [KJJ99, GMO01, RR01] captured the imagination of the cryptographic and security engineering community. Secondly, the attacks are characterized by their practical nature and their ease of implementation. This is in contrast to the often-theoretical nature of cryptography and cryptanalysis of algorithms. Thirdly, the increasing use and popularity of security devices provides a lucrative field in terms of funding and availability of devices to analyse. And finally, since the devices already exist and are used in real world systems, results tend to have significant real world implications.

1.4.6.1. Timing Attacks

In November 1995, Paul Kocher [Ko96] described an attack that exploited the correlation between the time taken to perform an operation and data involved in that operation. We describe the original timing attack against RSA. We briefly revise the notation and operation of the RSA public key cipher. The modulus $N = p \cdot q$ is the product of two large primes. A public exponent e is selected with the property that it be relatively prime to $\phi(n) = (p-1) \cdot (q-1)$ (i.e., $\gcd(e, \phi(n)) = 1$). Finally the private exponent d is calculated as $e \cdot d \equiv 1 \bmod \phi(n)$. A message ($M$) is encrypted under the public exponent as $C = M^e \bmod N$. The message M is signed (or decrypted) as $S = M^d \bmod N$.

It is a well-known fact that the running time of an exponentiation is dependent upon not only the lengths of the exponent and modulus but also the Hamming weight. For example, it is common to choose $2^{16}+1$ or 3 as public exponents for speed considerations [MOV96]. This can be clearly seen from a perfunctory analysis of the right-to-left square and multiply algorithm for modular exponentiation.

INPUT: M ; N ; $d = (d_{n-1} d_{n-2} \ldots d_1 d_0)$

OUTPUT: $S = M^d \bmod N$

1. $A \leftarrow 1$, $S \leftarrow M$

2. for $j = 0 \ldots n-1$ do
    3. if $d_j = 1$ then $A \leftarrow A \cdot S \bmod N$
    4. $S \leftarrow S^2 \bmod N$
5. return A

Algorithm 1: The right-to-left square and multiply algorithm

The loop need only be repeated j times where d_j is the first nonzero bit of d when scanning from left to right (i.e., execution time dependent on the length of d). Furthermore, the multiplication step inside the loop is only executed when d_j is nonzero (i.e. execution time dependant on Hamming weight of d). The length of the modulus affects the time taken for a modular multiplication as do the values of M and intermediary values of S and A.

Given that the loop consists of either one to two modular multiplications depending on the value of the associated bit of d, if it were possible for an attacker to measure the execution time of each loop, then it should be trivial to identify the value of d. A short loop would indicate the associated bit of d is 0 while a long loop would indicate a 1. It is not immediately obvious how that timing information might be measured (other later attacks including power and electromagnetic analysis do provide means).

Kocher developed a method to use the total time taken to perform the exponentiation to deduce the bits of the exponent. The attack proceeds as follows. A series of k messages $M_i \in Z_N, i = 1 \ldots k$ are submitted to the device for signing under the private key d. For each signature, the associated time $T_i, i = 1 \ldots k$ time taken for the operation is recorded.

INPUT: $M_i ; N ; d = \left(d_{n-1} d_{n-2} \ldots d_1 d_0\right)$

OUTPUT: $S_i = M_i^d \bmod N$

1. $A_{i,0} \leftarrow 1, S_{i,0} \leftarrow M_i$
2. for $j = 0 \ldots n-1$ do
    3. if $d_j = 1$ then $A_{i,j+1} \leftarrow A_{i,j} \cdot S_{i,j} \bmod N$, else $A_{i,j+1} \leftarrow A_{i,j}$

4. $S_{i,j+1} \leftarrow S_{i,j}{}^2 \bmod N$

5. return $A_{i,j+1}$

Algorithm 2: The parameterised right-to-left square and multiply algorithm

He observed that for some values of $A_{i,j}$ and $S_{i,j}$ the calculation $A_{i,j+1} \leftarrow A_{i,j} \cdot S_{i,j} \bmod N$ would be extremely slow – slow enough that it has a marked effect on the total exponentiation time. Since the attacker knows M_i, he can pre-calculate possible values of $A_{i,0}$, $S_{i,0}$ and determine whether the calculation $A_{i,1} \leftarrow A_{i,0} \cdot S_{i,0} \bmod N$ could be a particularly slow operation. If not, he chooses a new value for M_i until he obtains a suitable pair. He then times the exponentiation operation and if it is markedly slow deduces that $d_0 = 1$. Alternatively, if the exponentiation takes a normal length of time, it follows that the slow modular multiplication did not take place and hence $d_0 = 0$. Having identified d_0, the attacker repeats the process but this time looking for values of $A_{i,1}$ and $S_{i,1}$ that would result in a slow calculation of $A_{i,2} \leftarrow A_{i,1} \cdot S_{i,1} \bmod N$.

The attack can be treated as a signal detection problem where the signal is the timing variation due to the target bit and the noise results from measurement inaccuracies and timing variations due to unknown exponent bits. Each timing observation consists of $T = e + \sum_{i=0}^{n-1} t_i$, where t_i is the time required for the multiplication and squaring steps for bit i and e includes measurement error, loop overhead, etc. For a given guess for the first j bits of d (i.e., $d_0, d_1, \ldots d_{j-1}$), the attacker can calculate $\sum_{i=0}^{j-1} t_i$ for each message M. If the j bits are correct, then subtracting from T yields $e + \sum_{i=0}^{n-1} t_i - \sum_{i=0}^{j-1} t_i = e + \sum_{i=j}^{n-1} t_i$. Since the modular multiplication times are effectively independent from each other and from the measurement error, the variance of $e + \sum_{i=j}^{n-1} t_i$ over all of the observed samples is expected to be $Var(e) + (n-j) \cdot Var(t)$. However if only the first $k < j$ bits are correct, $e + \sum_{i=j}^{n-1} t_i +$ the expected variance will be $Var(e) + (n + j - 2k) \cdot Var(t)$. It can clearly be seen that a correctly emulated iteration has the effect of decreasing the variance by $Var(t)$. However an incorrect guess for an exponent bit causes subsequent rounds to have an increase in variance of $Var(t)$.

The different resulting effects on the total variance that correct and incorrect guesses for exponent bits have, demonstrates an error correcting property of the attack. After an incorrect exponent bit guess, the correlation between timings and subsequent exponent bits gets weaker. The attacker simply interprets his reduced ability to identify subsequent bits as an indicator that his previous guess was in fact incorrect. This property can be used to reduce the number of timing samples required.

In [DKLMQW98] 200,000 to 300,000 timing measurements of the modular exponentiation algorithm were required in order to recover a 512-bit secret RSA decryption exponent. The analysis stage took just a few minutes. The authors warned, "the timing attack represents an important threat against cryptosystems". This attack was then improved by a factor of 50 [SKQ01] with an improved mathematical model and error correction scheme. Timing attacks have now been extended to RSA using the Chinese Remainder Theorem [Sc00] with the additional requirement that the attacker must be able to choose the messages that get signed. Indeed, virtually any operation that takes a variable amount of time is potentially vulnerable to such an attack. Public key and signatures schemes such as ECC, RSA and El Gamal have algebraic operations that often run in non-constant time. Timing attacks have also been applied to symmetric ciphers. Attacks against Rijndael [KQ99] and IDEA [KSWH00] exploit the use of multiplication in their encryption process. Bit rotations using a shift and conditional bit wrap around can leak information in RC5 [HH98] and DES [HK99] implementations.

There exist several strategies to defend against a timing attack. First we can ensure that all operations run in a constant time which is a non-trivial task to achieve and will degrade system performance. Secondly we can spike the operation with random timing delays. This has the effect of increasing the number of timing measurements required, hopefully pushing the attack into the realms of impracticality. It too carries a performance penalty. The best solution is the 'blinding' of the ciphertext. Instead of signing the original message $M \in Z_N$, a random number $r \in Z_N^*$ is generated and the message $M' = r^e \cdot M \bmod N$ is signed instead, yielding S'. The true signature is calculated as $r^{-1} \cdot S' = r^{-1} \cdot r^{e \cdot d} \cdot M^d = M^d$. The method removes the attacker's knowledge of the number M' used in the modular exponentiation.

1.4.6.2. Power Analysis

Power Analysis was introduced by Kocher, Jaffe and Jun in [KJJ98, KJJ99] as a side channel attack, which exploited the power consumption of a cryptographic device. It received widespread attention

largely due to the very real threat it posed to smart cards. It is categorized into two types of attacks, namely simple and differential power analysis, SPA and DPA, respectively.

Simple Power Analysis is effectively a visual analysis of a plot of the power consumption of the device during operation. Consider the Figure 1-3 below. This graph published in [KJJ99] shows the power consumption during a DES operation. Key features of the algorithm such as the 16 rounds are clearly evident.

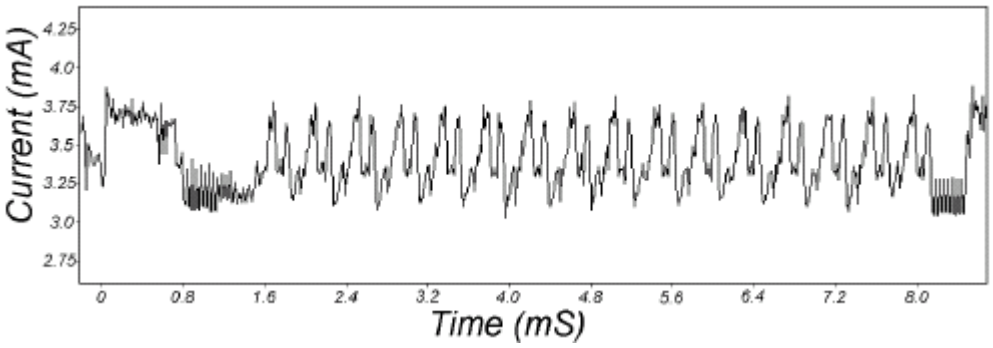

Figure 1-3: A SPA trace showing an entire DES operation.

Consider the partial trace of an RSA signature operation shown in Figure 1-4 (reproduced from [Mu01]). Clearly identifiable are the power spikes during either a squaring or multiplication operation. In fact the multiplication operations are distinguishable from the squaring operations owing to a wider spike as a result of extra register loads (a multiplication operation has 2 operands and a modulus compared to the squaring operation's single operand and modulus). Thus we can identify the following sequence of operations – squaring, squaring, squaring, multiplication, squaring, multiplication, squaring, multiplication and finally squaring. Realizing that this is obviously an implementation of the square-and-multiple algorithm where the multiplication operation is conditional on the value of the associated exponent bit, we can identify the exponent as containing the bit sequence {1, 1, 1, 0, 0}.

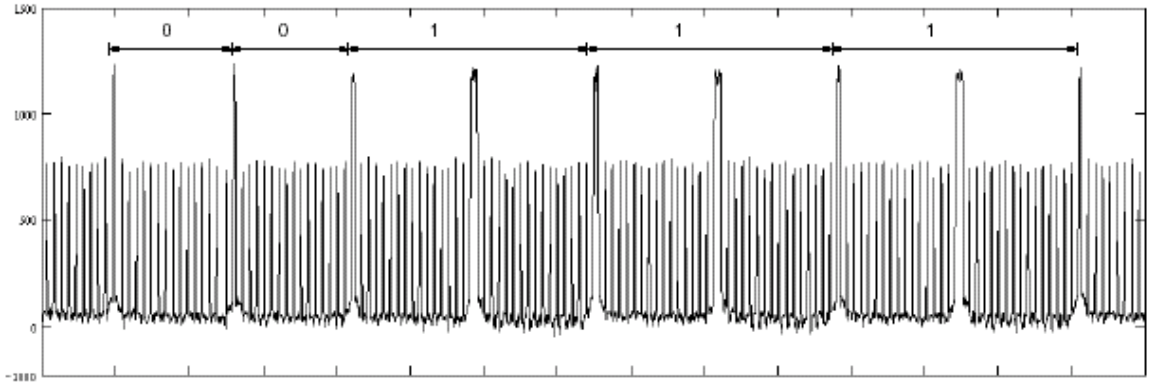

Figure 1-4: A SPA trace showing part of a RSA signature operation.

In contrast to SPA, which makes use of large-scale power variations due to instruction sequence, DPA relies of smaller, subtler effects owing to the actual data values being manipulated. This carries the challenge that these effects can be easily lost in amongst system noise and sampling error. We thus require the use of statistical methods.

As an illustrative example we use a cipher consisting of repeated rounds of a keyed function defined as $x_0 = f_0(x, k_0), x_i = f_i(x_{i-1}, k_i)$. We submit a large number of values x_j to the function. Consider the final round $y_j = f_n(x_{j,n-1}, k_n)$. Let $X_1, X_1, \ldots, X_k$ be a partition on the set of $n-1^{\text{th}}$ round inputs $\{x_{j,n-1}\}$ such that $X_k = \{x_{j,n-1} \mid x_{j,n-1} = k\}$. We know that the power consumption over time will differ depending on the value of the data. That is to say that the average power traces of X_i will exhibit a power consumption profile related to the input data. However a random data set will 'average' the effects of different input data.

The contrasting power consumption between two partitions (or sets) can be shown by effectively subtracting the respective average power traces. Common sources of power consumption between the two will cancel each other out leaving the differences highlighted as (positive or negative) spikes. In particular we expect two partitions to show a marked difference over the n^{th} round due to the difference between input data to that round. We would not expect such behaviour from two random sets. We thus have a technique for identifying whether two given sets of power traces belong to a partition or are random sets.

This leads to a method to search for the final round key k_n. We iterate through all possible values of k_n. For each value $k_{n,k}$ we can calculate the data input to the final round by $x_{j,n-1,k} = f_n^{-1}(x_{j,n}, k_{n,k})$

which allows us to partition the traces and determine the average trace for each partition. We then calculate the differences between the average traces between partitions. If our guess $k_{n,k}$ is incorrect, then the calculated $x_{j,n-1,k}$ are not the true values and our partitions based on these values are essentially random sets. In that case, we expect to see no marked differences between the average traces. If we have guessed correctly and $k_{n,k}$ is the true value for k_n then the partitions are likewise correct. Thus we expect to see data dependent differences between the average traces visible as spikes.

The graph Figure 1-5 below (reproduced from [KJJ99]) shows a reference power consumption followed by three traces. The first trace is a correct one and with clear evidence of positive and negative spikes resulting from different data. The last two traces are incorrect traces and exhibit no significant spikes.

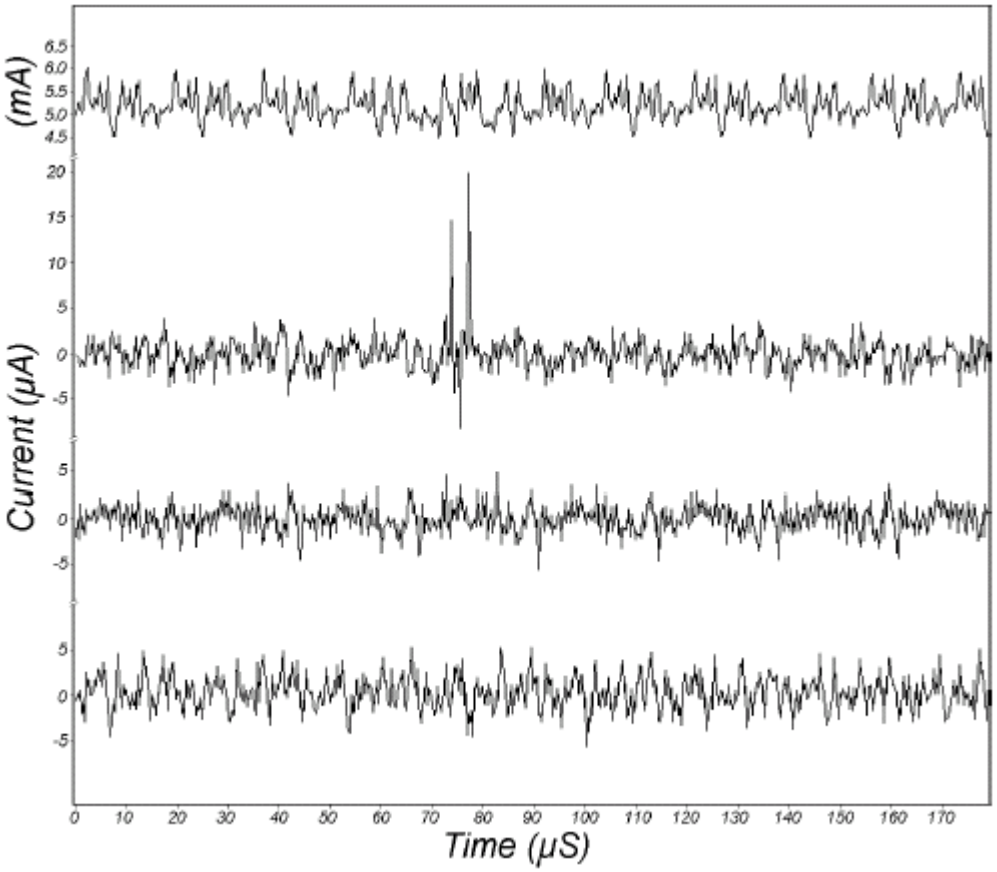

Figure 1-5: A reference trace followed by three averaged traces.

Smart cards have been a favoured target of power analysis attacks [KJJ99, MDS99a, MDS99b]. There has been a significant amount of research [CG00, CJRR99, Me00a, Sh00] into techniques defend against such an attack. These defences can be loosely categorised as:

- Reducing signal size using methods such as a constant execution path, selecting operations that leak less information in their power consumption or by physically shielding the device. However even a heavily degraded signal may still be detectable.
- Introducing noise into the power consumption measurements.
- Using a leak tolerant design methodology.

While these mechanisms do act to thwart power analysis attacks they do not necessarily render them infeasible but rather merely increase experimental and computational workload. This was shown in [CCD00] in which hardware countermeasures, including random process interrupts and noisy power consumption, were investigated.

1.4.6.3. Electromagnetic Analysis (EMA)

EMA was a natural extension to the research into power analysis (PA). Indeed many of the original papers related to power analysis [KJJ99] noted that electromagnetic emissions could perhaps be similarly exploited. It was suggested by J.-J. Quisquater that Simple ElectroMagnetic Analysis (SEMA) and Differential ElectroMagnetic Analysis (DEMA) were the analogous methods to SPA and DPA. The paper [RR01] presented some preliminary results showing leakage of compromising information via EM emanations and concluded that EMA was more powerful than timing or power analysis attacks. In addition, the EM side channel resembles the power side channel in the nature of the information revealed. The first actual implemented attacks against smart cards were presented in [GMO01] which serves as our reference for further discussions.

To explain possible sources of EM signals, consider the operation of an inverter, the role of which is to convert a low (ground) signal into a high (+V) signal and vice versa. A CMOS (complementary metal-oxide silicon) logic inverter comprising of two complementary field-effect transistors (FETs) is shown in Figure 1-6 and can be viewed as a push-pull switch. Grounding the input signal *in* will turn the top transistor on and the bottom transistor off causing the output to be pulled high. Conversely, a high input signal *in* will turn the top transistor off and the bottom transistor on causing the output to be pulled to ground.

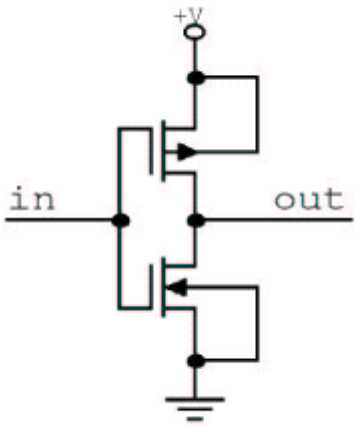

Figure 1-6: Elementary CMOS gate.

During a transition from 0 to 1 or vice versa, there is a short current pulse, which causes a sudden variation in the EM field surrounding the chip. Measuring these signals requires very small probes similar in dimension to the target devices. Since different areas radiate with different intensities (e.g. points near the CPU, data buses and power supply are generally more active), it is usual to measure local to areas that have strong EM emanations. When comparing EM and power curves, it was noted that while EM curves tend to be noisier, they feature sharper data signatures.

The tests of [GMO01] investigated three algorithms, alleged COMP128, DES and modular exponentiation. To effect a comparative study, EM and power signals were simultaneously acquired and analysed using the same software tools (i.e., the tools which performed the mathematical analysis for a DPA attack). In the case of DES and ACOMP128, both DPA and DEMA generated peaks for 'right guesses'. However, DEMA provided better peaks than DPA in terms of contrast and signal to noise ratios despite the more noisy measurements. Furthermore, DEMA had fewer and lower peaks for wrong guesses than DPA. Consequently DEMA requires a smaller sample set and can be considered experimentally at least as efficient as DPA, confirming these assertions in [RR01].

EMA's advantage is as a direct result of its capability of exploiting local information by changing the geographical point of measurement. This allows it to pinpoint the problematic areas that leak the greatest amount of information. This is in contrast to PA which typically allows only one point of measurement, namely the external supply to the card. However, the measurements required for PA are relatively simpler and easier to obtain particularly when compared with the more intimate access requirements of EMA.

1.4.7. Preventing Faults

1.4.7.1. Fault Analysis

Boneh, Demillo and Lipton introduced fault analysis [BDL97] in September 1996. They proposed a model in which errors were intentionally or accidentally introduced into a cryptographic operation or calculation. They then showed that, under such a model, it was possible to recover the private exponent of RSA key with only the result from a single erroneous signing operation as well as a good signature. A beautiful refinement by Leenstra created an elegant attack which only needed the faulty signature. Bao et al [BDHJNN98] described some variations of the attacks against RSA and other public key algorithms. Biham and Shamir [BS97] then demonstrated an attack against DES (and other block ciphers) based on the theory of differential cryptanalysis that they termed differential fault analysis. In [BMV00] the attacks were extended to elliptic curves.

Due to the speed advantages of RSA-CRT (nearly a factor of 4 speed improvement), many devices offer RSA-CRT support. Refreshing the basics of the CRT calculation of RSA. Let

$S_p = M^{d_p} \bmod p$

$S_q = M^{d_q} \bmod q$

where

$d_p = d \bmod p$

$d_q = d \bmod q$

The result is obtained by combining S_p and S_q using the Chinese Remainder Theorem as

$S = a_p \cdot S_p + a_q \cdot S_q$ for some predetermined values a_p, a_q where

$a_p = 1 \bmod p$, $a_p = 0 \bmod q$ and

$a_q = 0 \bmod p$, $a_q = 1 \bmod q$.

Suppose that an error occurs during the computation of S or more particularly during the calculation of either S_p or S_q. Without loss of generality, assume that the error occurs in S_q resulting in an incorrect value $\hat{S}_q \neq M^{d_q} \bmod q$. The resulting signature of M, which we denote $\hat{S}$, will be invalid.

Consider the value of $M - \hat{S}^e$.

$$M - \hat{S}^e \bmod p$$
$$= M - S_P{}^e \bmod p$$
$$= 0 \bmod p$$

$$M - \hat{S}^e \bmod q$$
$$= M - \hat{S}_q{}^e \bmod q$$
$$\neq 0 \bmod q$$

Thus p is a factor of $M - \hat{S}^e$ and q is not. Hence one need only calculate $\gcd\left(N, M - \hat{S}^e\right) = p$ in order to factor the modulus. Notice that attack has no requirements regarding the number, distribution or nature of the fault, but merely that a fault occurred.

1.4.7.2. Prevention

Since these attacks require the target to produce the output of faulty computation, it is an obvious strategy to prevent this. This leads us to methods of verifying the output of each cryptographic operation before outputting the results, discussed in [Sh97, YJ00]. One solution is to repeat the calculation a second time and confirm that the same answer is obtained in both cases. Alternatively, if possible, one can perform the inverse operation and verify that the result is the same as the initial input. Immediately one can expect a resulting loss in efficiency while dutifully performing the checks. The feasibility of inducing a fault is questioned in [Ma97], which remains the significant challenge for the adversary.

1.5. Logical Security

Previously we defined *logical security* as the mechanisms by which operating systems and other software prevent unauthorized access to data. This is a very broad topic and so we begin by listing some common components that may be present in a security design. These include:

- Algorithms
- Protocols
- Operating Systems and Applications

Since our focus is on the security and analysis of security devices, we shall concentrate on these areas as they affect security devices in a unique or particular manner. Thus, we shall largely ignore, for example, the traditional theoretical analysis of cryptographic algorithms.

1.5.1. Algorithms

Algorithms and their analysis form the heart of cryptography. They provide a set of techniques in order to achieve cryptographic goals of confidentiality, data integrity, authentication and non-repudiation. A treatment of this subject obviously exceeds the boundaries of this work. Fortunately there exist several excellent introductory and reference texts on the matter [St95, MOV96]. Any given practical security device will make use of a number of different algorithms. Normally one selects those algorithms that have been analysed by cryptographic experts for a number of years and have been established as official standards. This approach has a number of advantages, including compatibility, confidence, ease of development, legislated requirements and liability issues. Interoperability and compatibility with other system can only be ensured by strict adherence to a standard. By using an algorithm available in the public domain that has been exposed to years of analysis, one has increased confidence as to the correctness and security properties of the algorithm. It is highly likely that a reference implementation will be available in the form of software source code [Sc02, Da02] or at least the existence of reference test vectors to verify the implementation. This leads to a reduction in development time and lowering of costs. Many sectors and industries have legislated requirements relating to algorithm selection (e.g. the banking sector must satisfy ANSI Retail banking standards [ANSI X9.8, ANSI X9.17, ANSI X9.24]). Finally, by using an industry standard and accepted algorithm, one reduces liability in the event of a failure by demonstrating due diligence.

1.5.2. Protocols

Protocols describe the interaction between different entities. Similar to algorithms, protocols have been the focus of much research over the years. The security of a protocol can be analysed using formal methods. Perhaps the most cited paper on the topic was that on the topic of BAN logic developed by Burrows, Abadi and Needham [BAN89]. An interesting extension to formal methods is the use of theorem provers or model checkers to automate the proof generation and verify the correctness of the model.

As with algorithms, the design and justification of a secure protocol is not the focus of this work. This is due to a number of reasons. Firstly, it may be (and commonly is) the intention of the API to provide building blocks with which to construct a desired protocol. Also, there exist many established protocols providing a fairly comprehensive spectrum of functionality (e.g. The STS (site-to-site) protocol for exchanging keys [MOV96]). A designer can therefore merely select a standard protocol with the desired functionality. This also aids interoperability. The challenge lies in designing API level functions from which the desired protocol can be created, but that at the same time do not compromise the protocol or the device.

1.5.3. Operating Systems and Applications

The role of operating systems (OS) in security devices can sometimes be limited. For example many low-end devices (e.g. smart cards) do not have an operating system, although there have been recent developments in the field. Unlike a desktop machine or server where an OS must provide a wide variety of functionality, a security device represents a more closed system. For example, a given build of OS runs on exactly one hardware platform (i.e., the security device) for which all components are known and fixed. Their role in the crypto coprocessor is largely resource management including memory management (heap and stack) and potentially providing a multithreaded environment.

1.6. Environmental Security

Environmental security is the least interesting aspect of security from an analysis perspective. The goal here is to limit an attacker's opportunity to initiate an attack by creating layers on hindrance. Typical measures include physical access control through the use of identification and authorization key cards, closed circuit television cameras to monitor and guards to police the area. While these are all valid risk management techniques, they are not necessarily applicable to security devices operating in distributed and hostile environments.

The one exception is the operational procedure detailing the secure administration and use of a device. Our ultimate goal is the confidence that a correctly installed and configured device guarantees security regardless of its environment. This is particularly true of a tamper resistant and responding device for which its physical security mechanisms are designed to prevent an adversary gaining an advantage by having complete and total control over the environment (and hence being able to mount a physical attack).

We make a distinction between two classes of people who interact with a security device. The first group is the administrators of the device (sometimes called the security officers). These are the individuals who install and configure and authorize the establishment of master and operational keys. The second group is the users who simply make use of those functions and services presented by the device. Clearly the administrators have a certain amount of power over a device that could be misused to achieve nefarious ends. To prevent a single administrator from compromising the system, the principle of dual control is enforced. The requirement is that at least two administrators must agree to allow a function before that function is available. Thus at least two security officers must collude in order to circumvent the security offered by the device and so the single point of failure is addressed.

1.7. A New View of Security

The traditional categorization of security that has been presented suffers from two weaknesses. Firstly, it treats the various 'areas' of security as discrete entities without interaction. This is a throwback to the traditional approach of analysis which advocates a divide and conquer approach. Each facet of the security puzzle was identified and then individually targeted. As the field has matured (particularly in the case of security devices), there has been a steady increase in vulnerabilities or attacks arising from the interaction of one or more of the traditional categories. Obviously, a more holistic approach is required, that takes into account these subtle interactions. Secondly, the traditional categorisation does not necessarily give one an intuitive or clear overview of the security puzzle. We thus present a new representation of this security puzzle.

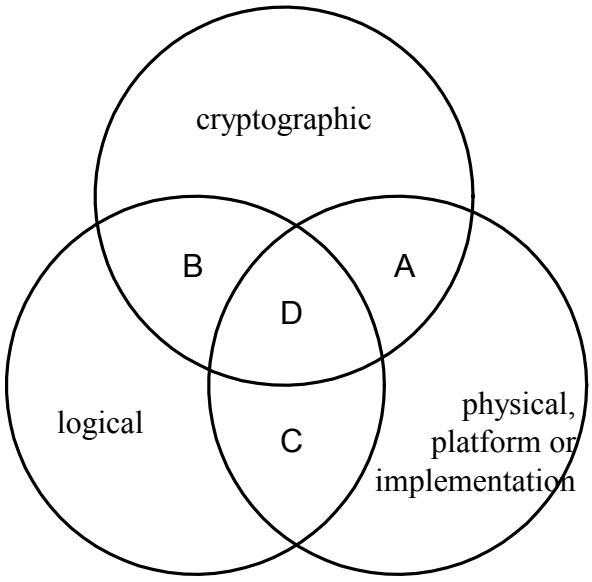

Figure 1-7: A new representation of the security considerations

The diagram indicates the interaction between three distinct areas of security, namely *cryptographic*, *logical* and *physical*. The cryptographic circle contains the mathematical aspects of cryptology. An attack within this circle is a traditional cryptanalysis that defeats a mathematical algorithm. The physical circle refers to an implementation of a security device. As such, it is not restricted to only devices with physical form, but also electronic (such as a software only solution). An attack here can be a physical attack or an implementation error (such as a software bug). The logical area refers primarily to the application programming interface (or any interface for that matter) which exposes the functionality of solution.

Consider now the intersections of the circles. The area denoted by 'A' is the intersection between cryptographic and physical domains. It is within this area that we find many of the attacks described thus far including the timing attack, power analysis, electromagnetic analysis and fault analysis. These attacks exploit the interaction between the cryptographic algorithm and the implementation thereof. For example, the timing attack correlates timing characteristics of the physical operations with the mathematical structure of an encryption algorithm. The solutions or defences to these attacks also draw from both domains.

The area 'B' marks the intersection between the logical and cryptographic domains. Here we find a growing number of API attacks which provide methods to circumvent, overcome or manipulate the cryptographic algorithms. Most of the subsequent work in this dissertation involves analysing this interplay.

Area 'C' has not been studied extensively. We would expect to see possible weaknesses in interaction between the API and management of the implementation or physical device. Other possible weaknesses would be the compromising of communication channels and device identification.

One can further extend the model including operational considerations. Since interaction with the security device is through the interfaces, it only intersects with the logical domain.

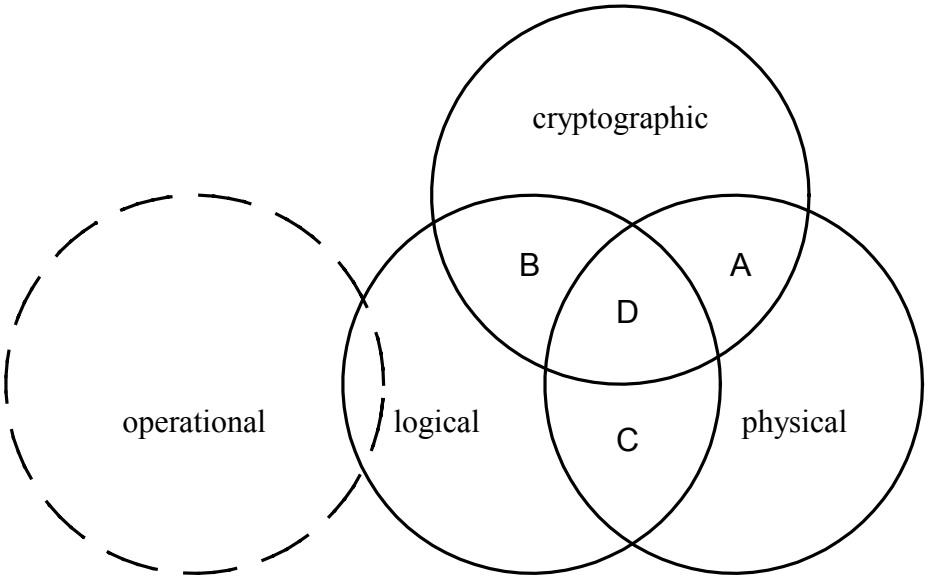

Figure 1-8: Extended representation of the security considerations

2. The Standard Cryptographic API

We begin by describing a standard cryptographic API that implements the normal functionality that is required of a security device. While it is a challenge in itself to make devices, hailing from different vendors, work together, we shall seek to extract the common functionality provided. We do not profess to be describing the best possible prototype. The intention here is to develop for further analysis an API that is generally representative of the current state of the field. When there is some uncertainty or flexibility in a decision, we favour the most common industry practice determined by taking a consensus of the CCA, Thales-Zaxus-Racal, Compaq-Atalla and PKCS#11 APIs listed below.

2.1. Our Reference APIs

As our choice of reference APIs, we have selected four common and commercially significant APIs.

2.1.1. The Common Cryptographic Architecture (CCA) from IBM

IBM has a long history in the design of security devices for a wide range of applications. It is the de facto standard in mainframe security and was the first company to develop a FIPS 140-1 Level 4 certified device in the 4758, a programmable, high speed, secure coprocessor. It has developed a number of APIs through the years including the Transaction Security Services (TSS) API and later the Integrated Cryptographic Services Facility (ICSF) API which provides the callable security services for OS/390 mainframes. The 4758 was released with a modified version of ICSF called the Common Cryptographic Architecture (CCA) API.

2.1.2. The Thales-Zaxus-Racal API

The Racal brand is an established name in the international market for security modules for retail financial institutions and dominates the market outside the US. Although acquired by Zaxus and later by Thales, they have been slow to change marketing details (such as product names) due to the very strong branding and brand recognition.

2.1.3. The Compaq-Atalla API

Atalla's security processors, referred to as Network Security Processors (NSPs), dominate the US market for banking and financial security devices. The range of Atalla security devices includes A8000, A9000, A10000E, and the A10000/PCI card and are designed to provide privacy, integrity, and performance in Automated Teller Machine (ATM), Electronic Funds Transfer (EFT), and Point

Of Sale (POS) networks. They can also be used for other types of applications that require Data Encryption Standard (DES) or triple DES (3DES) support. Public key (PKI) cryptography is not supported in this product.

2.1.4. Public Key Cryptography Standard (PKCS) #11 from RSA

PKCS #11 was developed by RSA Labs in cooperation with representatives of industry, academia and government to provide a standard to allow interoperability and compatibility between vendor devices and implementations. As a result, it is an extremely well known and popular API. Many important, existing APIs and protocols have been built upon PKCS#11 (e.g. SSL). Notable products including Mozilla (the Netscape browser) and SSL hardware accelerators from nCipher, IBM, Thales, Rainbow, AEP/Baltimore, etc support or make use of the API.

The design goal of PKCS#11 was to provide a standard interface between applications and (portable) cryptographic devices. It is designed to allow resource sharing (a many-to-many relationship between applications and devices). It is not intended to be a general interface to cryptographic operations or security services but could be used to build such services, operations or suitable APIs.

2.2. The Basic Functionality

2.2.1. Key Tokens and Formats

Keys are the most critical detail of any cryptographic system. They are input parameters to most of the cryptographic functions in the API (with the exception of some un-keyed functions such as hashing or message digests). Either the keys can be passed in with every call [IBM] or alternatively they can be stored internal to the device [PKCS#11] and referenced by some index or handle. Some devices provide both options [THALES]. Internal storage carries a limitation with respect to the space available in memory to store a potentially large number of keys. The former approach requires that the keys be encrypted under another key when outside the physical boundaries of the device to protect their secrecy. Thus with each function call, the encrypted key (or key token) is supplied to the device which decrypts the key and extracts the clear value for use in the function. A master key (normally a double or triple length DES key) is stored internal to the device that is used to encrypt the other keys.

We differentiate between two types of keys known as *operational* and *external* keys. An *operational* key is a working key that can be used in the system as is (including keys encrypted under the master

key). An *external* key is typically a key that is to be imported into the system (or has been exported from the system). In a sense it is thus external to the system and requires a conversion (or import) process to turn it into a working key. Typically it is encrypted under a key encrypting key (i.e., key exchange). A key encrypting key could be a symmetric or asymmetric key. We are thus interested in the encryption of keys both under the master key (as an operational key) and under a key encrypting-key (external key).

We refer to an encrypted key token (or key block) as $T = e_{MK}(k)$ (i.e., the encryption of the key k under the master key MK). Some of the reference APIs do not include any additional information in the key block [ATALLA, THALES]. The IBM token is the most complicated, including additional key information and verification value (a simple error detecting code).

Offset	**Length in Bytes**	**Meaning**
00	1	X'01' (a flag that indicates an internal token)
01	1	Reserved, binary zero
02	2	Master key verification pattern
04	1	The version number (X'03')
05	1	Reserved, binary zero
06	1	Flag byte 1
07	1	Reserved, binary zero
08-15	8	Reserved, binary zero
16-23	8	The single length operational (master key encrypted) key or the left half of a double length operational key
24-31	8	Null, or the right half of an operational key
32-39	8	The control vector base
40-47	8	Null, or the control vector base for the right half of a double length key
48-59	12	Reserved, binary zero
60-63	4	The token-validation value

Figure 2-1: CCA DES Internal Key Token

As can be seen from above, the halves of a double length key are encrypted separately. That is to say that the double length key $k = \langle k_{left}, k_{right} \rangle$ is encrypted as $T = \langle T_{left}, T_{right} \rangle = \langle e_{MK}(k_{left}), e_{MK}(k_{right}) \rangle$.

2.2.2. Key Separation

Key separation is a mechanism, which enforces the use of a given key as intended. The well known attack scenario is supplying an encrypted external key token and the associated key encrypting key to a standard data decrypt call, which would result in the clear key being returned.

For example, let k be the clear value of the encrypted token $T = e_{MK}(k)$. We can 'use' this token without knowing the value thereof. If there is no separation between key encrypting keys and data decrypting keys we can use k in either call. Let k_{target} be an unknown key that we wish to recover. We export k_{target} under k yielding the external token $T_{target} = e_k(k_{target})$. We then supply T_{target} as encrypted data to a *data* decryption call using the same key k. This returns the cleartext $P = d_k(T_{target})$ which is $P = d_k(T_{target}) = d_k(e_k(k_{target})) = k_{target}$. Hence the clear value of the target key k_{target} is obtained.

This is an attack on the correctness of the transaction set and demonstrates the necessity to 'separate' key encrypting keys from data decrypting keys. There are two common methods for achieving this. The first is to use different master keys for encrypting different types of keys. Incorrectly using an encrypted key would result in the decryption of the key under the incorrect master key, yielding a 'random' result. This may be detected if parity checking of keys is enforced or enabled. The second method involves the creation of a variant of the master keys based on the type of key.

Perhaps the most widely used system, is IBM's control vector method. A unique control vector (CV) is associated with each type of key. To encrypt or decrypt the key, the master key variant is created by exclusive-oring the master key with the control vector. This operation is represented as $T = \langle T_{left}, T_{right} \rangle = \langle e_{MK \oplus CV_{left}}(k_{left}), e_{MK \oplus CV_{right}}(k_{right}) \rangle$.

Again, using an encrypted key as incorrect type, results in a 'random' result. The control vector can encode additional information about the key including the length of the key and properties such as whether the key is exportable.

Typical classes of keys include:

- Encryption/Decryption Keys (also called DATA, ENCRYPT or DECRYPT keys)

- MAC Generation/Verification Keys (also called MAC_GEN, MAC_VER, DATA and DATA_MAC keys)
- Key Encrypting/Decrypting Keys (also called EXPORTER and IMPORTER keys)

The IBM API strictly enforces the 'direction' of the key in the sense that an Encryption Key has a different control vector to a Decryption Key and cannot be used to decrypt data.

2.2.3. Key Management

2.2.3.1. Key Generation

Many high-end security devices have hardware noise sources for the generation of random data. Due to the challenges of designing a truly random hardware source, the harvested noise is often then used to seed a pseudo random number generation algorithm. The resulting output can be used to create keys.

The basic key generation function specifies the type and length of the key to be generated. Some APIs have restrictions on the length of certain types of keys (for example specifying that a key encrypting key must be a double length key). For compatibility with single length key legacy systems, sometimes the option to generate a double length key with repeated halves is available (as this is effectively a single length key).

```
Key Generate
{
  //inputs

  Type,                    //key type
  Length,                  //either a single, double to triple length key
  Key Token Template,      //

  //outputs

  Error Code,              //error code
  Generated Key Token
}
```

2.2.3.2. Key Derivation

Key derivation results in the creating of a new key from a base key. In doing so, it provides two functions. Firstly, it can provide an alternative to the key generation function in the sense that it generates a key (using a derivation algorithm as opposed to using random data). Secondly, it can create, for each transaction, a unique or new key that is related to the base key and an additional piece

of information (e.g. a transaction counter). Two common derivation algorithms are the VISA Derived Unique Key Per Transaction (DUKPT)[ANSI X9.24] and offsetting [ANSI X9.24, ANSI X9.17]. The use of offsets is the easiest method to explain. A 56 bit offset or transaction counter i is applied to the base key, k, as $k(i) = k \oplus i$ (or for a double length base key $k = \langle k_1, k_2 \rangle$, $k(i) = \langle k_1 \oplus i, k_2 \oplus i \rangle$).

Associating a transaction counter (or other unique information) with a key has the advantage that such a construction can prevent a key from being misused, defeating common attacks involving replay or out of sequence transactions. Another approach is binding the identities of entities involved in the transaction with the value of the key. This limits misuse of the key by parties other than the creator and intended recipient. The process of *notarization* [ANSI X9.24, ANSI X9.17] associates the identities of a pair of communicating parties with the key by cryptographically combining the key with a 16-byte origin identifier and a 16-byte destination identifier.

There exist several other custom algorithms related to key derivation including important payment schemes such as Europay MasterCard Visa (EMV) [EMV00a, EMV00b, EMV00c, EMV00d]. For the purpose of defining our standard API, we shall allow for those methods listed in the ANSI standards – offsetting, notarization and DUKPT.

2.2.3.3. Key Establishment

The traditional method for solving the initial trust problem for symmetric key cryptosystems has been through the use of key shares. This involves several trusted parties (at least two) having key shares that are then exclusive-ored together to form the key. For example, the three key shares k_a, k_b and k_c are combined using the exclusive-or operation to form the key $k = k_a \oplus k_b \oplus k_c$. Each trusted party is responsible for keeping the individual share in his or her possession secret. Without the collaboration of all guardians, it is not possible to create the key.

A typical call may have the following form.

```
Key Part Import
{
  //inputs

  Options,                  //First, Middle or Last
  Key Part,                 //16 byte clear key part
  Key Token,                //token to be updated with key part
```

```
  //outputs

  Error Code,               //error code
  Updated Key Token         //updated token
}
```

The first time the function is called, an empty token (or template) is provided. The template contains information relating to the type and length of the key. The options parameter (or rule) is set to '`First`' to indicate that this is the first share that is being entered. Since it is the first share, it is merely inserted into the key token, encrypted and outputted as a token (marked as a key part of the correct type). For subsequent shares the process is a little different. The outputted key part token from the previous step is supplied as the input key token with the option set '`Middle`'. Now the key token is decrypted and exclusive-ored with the new clear key part. Again the result is encrypted and outputted as a key part token. The final share is marked as '`Last`', which results in the final token being returned as a complete key token following the final exclusive-or operation.

Some systems do not pass the clear key part into the call, but require the guardian of the key share to enter it directly into the security device using an attached keypad. The operation is thus treated as a single call with the security device prompting the guardians for the shares.

2.2.3.4. Key Exchange (Export/Import)

The key export and import calls are fairly intuitive. As the name suggests, they facilitate the transfer of keys to and from a device. This requires that the exchanged key be encrypted under a transport key (the key encrypting key) that is shared between devices. This process is also known as key wrapping.

```
Key Export
{
  //inputs

  Target Key Token,         //key to be exported
  Key Encrypting Key Token, //key to be used to encrypt target key

  //outputs

  Error Code,               //error code
  Exported Key Token        //external token containing encrypted key
}

Key Import
{
  //inputs

  Exported Key Token,       //external token containing encrypted key
  Key Encrypting Key Token, //key to be used to decrypt key
```

```
  //outputs

  Error Code,               //error code
  Imported Key Token        //imported key in operational form
}
```

2.2.3.5. Key Test (Check Value)

The key test is simple function that generates a value using a one-way function which witnesses to the value of the key without revealing the value. This is achieved by encrypting a binary zero string with the key (i.e. $check\_val = e_k(0)$). The first 3,4 or 8 bytes of the ciphertext are returned as the check value. It is commonly used to confirm that a key has been exchanged or established correctly.

```
Key Test
{
  //inputs

  Key Token,       //


  //outputs

  Error Code,      //error code
  Check Value      //the check value of the key
}
```

2.2.4. Encryption/Decryption

The encryption/decryption function is responsible for data secrecy. The user-supplied data is either encrypted or decrypted under the given key. The resulting ciphertext or cleartext respectively is returned to the user.

```
Encrypt
{
  //inputs

  Key Token,       //
  Data,            //data to be encrypted


  //outputs

  Error Code,      //error code
  Encrypted Data   //the encrypted ciphertext
}
```

2.2.5. Message Authentication Codes (MAC)

Message authentication codes (as the name suggests) addressed the issue of authenticity. The MAC Generate call calculates a MAC over the user-supplied data using the given key. The MAC Verify call is used to verify that the data has not been modified. It recalculates the MAC and compares it to the supplied MAC thereby confirming the authenticity of the data.

```
MAC Generate
{
  //inputs

  Key Token,        //
  Data,             //data over which the MAC is to be calculated


  //outputs

  Error Code,       //error code
  MAC               //the 8 byte MAC
}
```

2.3. The Standard Cryptographic API

Thus we have arrived at our first pass standard cryptographic API. The functions we have included are:

- Key Generate
- Key Part Import
- Key Export/Import
- Key Test (Check Value)
- Encrypt/Decrypt
- MAC Generate/Verify

Some initial comments: We have not included any public key functionality. While our reference APIs do support public key operations particularly for key exchange (albeit limited otherwise), this topic falls beyond the scope of this dissertation. It also limits the analysis to a more manageable set of functions. For the same treason we have also simplified some of the functionality of our reference APIs.

In our function prototypes we did not explicitly indicate the ability to use key offsetting or notarization. However, we shall assume that it remains a feature of the API in some form.

2.4. Some Useful Cryptanalysis Results

2.4.1. Parallel Search Attacks

The use of parallelism in an exhaustive search was described in [DHQ86, EFF98]. Given many keys, each encrypting (or decrypting) the same text, it is possible to search for all keys in parallel (or simultaneously) using an exhaustive key search machine.

The key search machine iteratively generates keys and then performs a trial encryption (decryption) of the common plaintext (ciphertext). It then compares the resulting ciphertext (plaintext) with the data set and searches for a match. In this process, it searches for many keys in parallel for the expense of a single DES operation. As can be seen in Figure 2-2, the key space that is required to be searched decreases on average with the number of target keys.

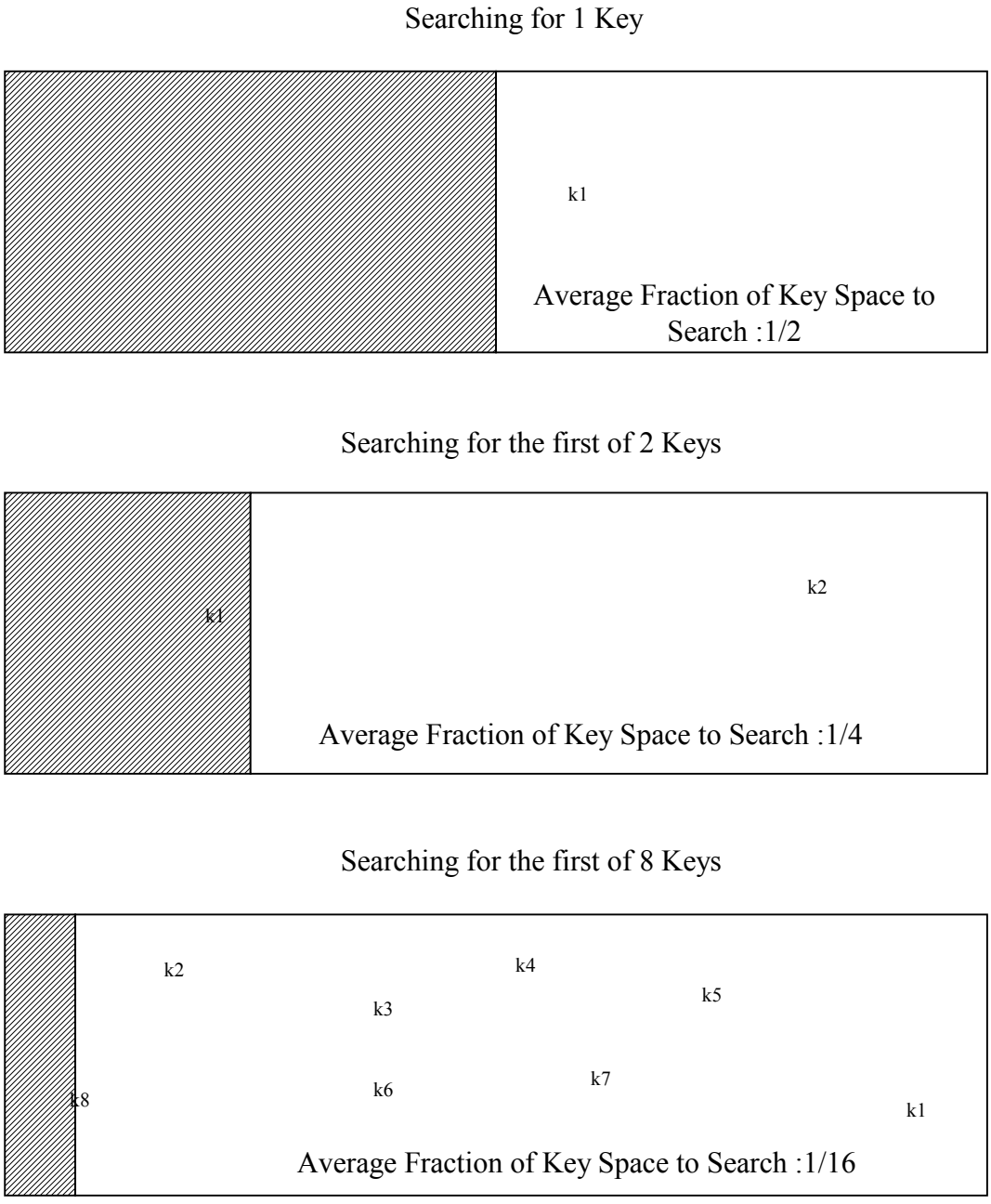

Figure 2-2: Parallel Search

If searching for 2^n keys, we can expect to encounter on average one in 2^{55-n} operations. This technique was used in [Bo01, AB01] in against the APIs of several crypto coprocessors. The attack was extended further using known related key [CB02]. If the differences between the related keys are known, then finding one key trivially recovers all other keys. The attack requires 2^n data and memory. (Bond described this attack as a 'meet in the middle attack').

2.4.2. Related Key Attacks

Eli Biham introduced related Key attacks in his paper "New Types of Cryptanalytic Attacks Using Related Keys" [Bi93]. In this paper, he attacked specific ciphers using related key methods. This work was extended by Kelsey, Schneier and Wagner [KSW96, KSW97]. Amongst other cipher specific attacks, they presented an attack against any cipher operating in a triple mode (e.g. Triple DES). We present a complete discussion of related key attacks against any symmetric cipher. Note that, although the attacks presented are applicable to all symmetric ciphers, we commonly describe them as attacks against DES due to its pre-eminence.

A related key attack involves the attacker's having access to data enciphered (or deciphered) by two related keys K1 = <k1>, K2 = <k1 ⊕ Δ>, where Δ is some (usually but not always known) difference. It has been questioned whether it was practically possible to obtain two such related keys. Bond settled any debate in attacking the CCA API of the 4758. Indeed, it will be shown that it is easy to achieve this within many security APIs.

2.4.2.1. 3 Key 3DES

The attack is elegantly simple and easily explained. Using the related key pair K1 = <k1, k2, k3>, K2 = <k1 ⊕ Δ, k2, k3>, encrypt a plaintext P with K1, and then decrypt the ciphertext with K2 yielding P'. Then

$$C = e_{(K1)}(P),$$
$$P' = d_{(K2)}(C),$$

and hence

$$P' = d_{(K2)}(e_{(K1)}(P)).$$

Using 3DES in EDE mode (the mode itself doesn't matter)

$$P' = d_{(k1\oplus\Delta)}(e_{(k2)}(d_{(k3)}(e_{(k3)}(d_{(k2)}(e_{(k1)}(P))$$
$$= d_{(k1\oplus\Delta)}(e_{(k1)}(P))$$

Thus k1 has been successfully isolated and can be recovered independently of k2 and k3 – typically by a related key search. The work required to effect the search is 2^{56} single DES operations. Hence the cipher in triple mode has been reduced to (almost) the strength of the cipher in single mode.

No attack was presented against 2 key 3DES and it was noted in [KSW97] that this may be the first attack for which 2 key 3DES was stronger than 3 key 3DES.

2.4.2.2. 2 Key 3DES

We now present a new related key attack applicable to 2 key 3DES. The related key pair for the attack is K1 = <k1, k2 >, K2 = <k1 ⊕ Δ, k2 >. We require two plaintext messages P and P', encrypted under K1 and K2 respectively yielding C and C'; that is, $C = e_{(k_1)}\left(d_{(k_2)}\left(e_{(k_1)}(P)\right)\right)$ and $C' = e_{(k_1 \oplus \Delta)}\left(d_{(k_2)}\left(e_{(k_1 \oplus \Delta)}(P')\right)\right)$. P and P' must satisfy the relationship:

$e_{(k1)}(P) = e_{(k1 \oplus \Delta)}(P')$

so

$d_{(k2)}(e_{(k1)}(P)) = d_{(k2)}(e_{(k1 \oplus \Delta)}(P'))$.

Since

$d_{(k2)}(e_{(k1)}(P)) = d_{(k1)}(C)$

and

$d_{(k2)}(e_{(k1 \oplus \Delta)}(P')) = d_{(k1 \oplus \Delta)}(C')$,

it follows that

$C' = e_{(k1 \oplus \Delta)}(d_{(k1)}(C))$.

C, C' and Δ are known, thus allowing us to search for k1 independently of k2.

The challenge remains to obtain P and P' satisfying the above relationship. One approach is the following. Given P and knowing Δ, calculate a suitable value of P' for each possible value of k1 using $P' = d_{(k1 \oplus \Delta)}(e_{(k1)}(P))$, and submit it under K2 to obtain C'. If it is true that $C' = e_{(k1 \oplus \Delta)}(d_{(k1)}(C))$, then we have found k1. This requires $2^{56} + 1$ queries of the target, which is practically infeasible given the current state of the art in crypto accelerators.

Another approach is to have 2^n values of P and 2^n values of P', encrypted under K1 and K2 respectively, thus requiring 2^{n+1} queries of the target. The corresponding values of C and C' are stored. The attacker iterates through values of k1 and searches for a pair (P, P') satisfying $P' = d_{(k1 \oplus \Delta)}(e_{(k1)}(P))$. The probability of this relationship existing for a given (P, P') is 2^{-64}. However, there are 2^{2n} (ordered) pairs (P, P'). Thus for n = 32 and by the birthday problem, we can expect to find such a pair, which will witness the true value of k1.

This attack does have some significant memory and data requirements, with the most onerous being the data collection (2^{33} plaintext-ciphertext pairs). Currently, top end crypto coprocessors typically exceed 1000 tps (transactions per second), at which rate the data could be collected in just under 100 days. 2^{37} bytes of memory would be required (or 128 GigaBytes). Thus, provided that an attacker can have access to a module for the required time (for instance, suppose he stole a tamper protected device), the data collection stage is feasible (albeit barely).

However, as one expects for any time data tradeoff, there is a penalty in the analysis stage. For each of the 2^{32} values of P and for each value of k1 we must search for an appropriate $P' = d_{(k1 \oplus \Delta)}(e_{(k1)}(P))$, requiring a total of 2^{89} DES operations. Although this can be performed offline by a customized, special purpose built machine, it represents another infeasible target.

2.4.3. Combined Attacks

It is evident that the parallelism offered by Related Keys can also be extended to attacks against ciphers in multiple modes of operation. In particular, 3 key 3DES is vulnerable to the combined attack. Choosing 2^n values for k1 (where the related key pair is K1 = <k1, k2, k3>, K2 = <k1 $\oplus \Delta$, k2, k3>), we can recover one value of k1 in 2^{56-n} operations on average. The attack requires 2^n data and memory.

2.5. Symmetric Key Attacks

The seminal work on attacking the APIs (or transaction sets as they are sometimes called) of security devices is that of Bond [Bo01] in which he identified a number of vulnerabilities in commercial APIs. These included the parallel search (which he called the meet in the middle attack), related key attacks, unauthorized type casting, poor key half binding and key conjuring. The work was extended in [AB01] and [CB02], the latter gaining a considerable amount of publicity.

2.5.1. Known Keys in the System

We briefly consider the dangers associated with having a known key in the system. The key type largely determines the exposure. Perhaps the most dangerous situation is a known key wrapping key, which can be used to export other keys from the system (Exporter key type). Any other key not marked as 'not exportable' can be recovered in this way. A known exporter key can also be used to get other known keys into the system. For example, the key generate function could be used to generate a key of the desired type, with the property of being exportable, that can then be recovered

using the known exporter key. A known importer key-wrapping key can also be used to obtain other known keys. In this case, an attacker can simply calculate the result of encrypting a clear key under the known importer key and then importing it. A known PIN encrypting-key can be used to recover the clear PIN from an encrypted PIN block while a known PIN generation and verification keys can similarly be used to recover PINs but require only the clear generation/verification data.

Any function that allows the establishment of a key is a potential hazard. Normally the key shares k_a, k_b and k_c are combined using the exclusive-or operation to form the key $k = k_a \oplus k_b \oplus k_c$. If all the key shares could be provided by the same person (that is, one person has the authority/ability to perform the key establishment by himself), then a known key can be trivially inserted. Hence this operation normally requires the authorization of two security officers. In [AB01] an interesting example is provided where the two key shares (k_a and k_b) are encrypted under a transport key and supplied as encrypted tokens to the establishment call ($e_{TK}(k_a), e_{TK}(k_b)$). The security device decrypts both tokens before performing the exclusive-or operation. This prevents the person who performs the establishment from being able to create his or her own known key. However Anderson and Bond observed that a malicious operator could still subvert the process by supplying duplicated encrypted shares, say $e_{TK}(k_a)$ and $e_{TK}(k_a)$ since, when the clear values of the components are exclusive-ored together, they result in a null key (i.e. $k = k_a \oplus k_a = 0$).

2.5.2. Related Key Attacks

Following from the earlier discussions on related keys, we now need only show how one might create a set of related keys. The simplest approach is through the use of key shares (i.e., three key shares k_a, k_b and k_c are combined, using the exclusive-or operation, to form the key $k = k_a \oplus k_b \oplus k_c$). Trivially the process can be repeated, but using a modified share, $k_c' = k_c \oplus \Delta$, where Δ is the chosen none-zero difference. Thus a second related key $k' = k_a \oplus k_b \oplus k_c' = k_a \oplus k_b \oplus k_c \oplus \Delta = k \oplus \Delta$ is obtained. Other techniques for obtaining related keys include the use of offsets and other key derivation and manipulation methods. The use of offsets where the offset i is applied to the key as $k(i) = k \oplus i$ can be similarly exploited as a mechanism to create related key pairs. A second key is used with the offset $i \oplus \Delta$ as follows $k' = k(i \oplus \Delta) = k \oplus i \oplus \Delta = k(i) \oplus \Delta$. Again the second key is related to the first key by a non-zero difference Δ. Notarization provides a similar opportunity.

2.5.3. Key Conjuring

Key conjuring is any technique that leads to the unauthorized generation of keys in the device. It is so named due to the fact that the keys are 'conjured' (magically created or appear) seemingly out of nowhere. Bond in [Bo01] first identified key conjuring as a security risk. This is for two reasons. Firstly, it defeats any access control that was placed on the official key generation function by providing an alternative and unauthorized mechanism to effectively perform the same operation. Secondly, a key conjuring mechanism can be exploited to build a large set of keys, which can then be attacked by a parallel search (as described in Sections 2.4.1 and 2.5.4).

Bond observed that crypto coprocessor designs that stored keys outside the tamper-proof device were vulnerable to unauthorized key generation. For example, a random 8 bytes submitted as an external encrypted DES key will be decrypted and used as key. For example, using random data (R), a user creates a token $T_{random} = R$ which is then supplied in a function call to the device. The device decrypts T_{random} as $d_{MK}(T_{random})$ yielding a new key $k_{random} = d_{MK}(T_{random})$. If parity checking is enforced, then there is a 1 in 2^8 chance that this new 'key' will have the correct parity. By repeating this process on average 2^8 times, an attacker can expect to successfully conjure a new key into the system in this manner. In fact this method is available in some older devices as a key generation function. Instead of merely testing for parity, the function will correctly set the parity in the process.

Referring back to possible key token formats for DES keys, we see that the so-called 'common' key token is vulnerable to this attack. This means that all three of our reference APIs are vulnerable to this attack. Key conjuring can be defeated through the associated use of a MAC, hash or potentially with an encrypted cyclic redundancy check (CRC) or error detecting code. This has the property of authenticating the clear value of the key as valid. This attack is not only limited to keys stored externally under the master key but is also possible through key exchange mechanisms (particular key import). This is due to the fact that the wrapped key is usually in an unauthenticated token format [ANSI X9.17, ANSI X9.24].

2.5.4. Parallel Key Searches

The key generation function can be used to facilitate a parallel key search. By generating 2^n keys (i.e., $\{k_i \mid i = 1..2^n\}$) we reduce the average search effort to 2^{55-n} operations to recover a DES key. The obvious method to protect against its misuse is the use of access control. However, the previous key conjuring technique (Section 2.5.3) also immediately leads to a parallel search attack as it too

presents a method to generate a large number of keys. As mentioned earlier, this defeats any access control protection on the key generation function.

The challenge with parallel key searches is to extend the results to double length DES keys. The problem lies in the time taken to generate the set of keys that is searched. High-speed crypto coprocessors can perform about 1000 operations or transactions per second (TSP), so theoretically a set of 2^{16} keys can be obtained in about a minute (in practice this may be a little slower). However, a year of harvesting keys would yield about 2^{35} keys, which would still require $2^{111-35} = 2^{76}$ operations on average to recover the first key – a currently infeasible task.

Sometimes an API provides a convenient mechanism to isolate the key halves. Consider the Check Value (or Key Test) function used to confirm value of the key without compromising its confidentiality. The check value (check_val) is the result of encrypting a test pattern (typically the null test pattern) with the key; hence $check\_val = e_k(0)$. Several APIs have chosen to provide a check value on each half of the key individually. For example, the check value for the key $k = \langle k_{left}, k_{right} \rangle$ is the ordered pair $\langle check\_val_{left}, check\_val_{right} \rangle = \langle e_{k_{left}}(0), e_{k_{right}}(0) \rangle$. In this case, the key halves can be attacked independently. This API construction is not uncommon and was identified in [Bo01].

Key search is also facilitated when the API allows the generation of double length keys with repeated halves (i.e, $k = \langle k_{left}, k_{left} \rangle = \langle k_{right}, k_{right} \rangle$). Functionally speaking such a key is indistinguishable from a single length key with the same value as one of the halves.

There exist additional means to obtain large key sets using methods apart from key conjuring and key generation. For example, key establishment using key shares ($k = k_a \oplus k_b \oplus k_c$) provides an opportunity to create a set $\{k_i \mid k_i = k_a \oplus k_b \oplus k_{c,i}\}$ where $k_{c,i}$ is a modified final share. Similarly key derivation provides a mechanism through the use of offsets ($k(i) = k \oplus i$) by creating the set $\{k_i \mid k_i = k(i) = k \oplus i\}$.

2.5.5. Key Integrity

Key Integrity is perhaps the most significant and widespread oversight in historical API design. The problem was exacerbated with the migration from DES to 3 DES. This required a change in the key tokens from single length to double (or triple length). The usual flaw is the lack of cryptographic

binding of the two key halves. So, although the key is encrypted, it is possible to replace one of the original key components of a double length key with a Trojan component or key. If the Trojan key is known, then this technique isolates the unknown key components, reducing the security from an unknown 112-bit key to 2 unknown 56-bit keys. The process is as follows. The original key $k = \langle k_{left}, k_{right} \rangle$ is modified to produce to derived keys $k' = \langle k_{left}, k_{trojan} \rangle$ and $k'' = \langle k_{trojan}, k_{right} \rangle$ allowing k_{left} and k_{right} to be searched for independently of each other.

This attack can be effective combined with the previously mentioned parallel search methods, to isolate the unknown key components first and then further reduce the search effort required through the use of parallelism.

Another variation exists when the left and right key halves are interchangeable, leading to the possible creation of two unknown double length keys with both halves repeated (i.e., $k' = \langle k_{left}, k_{left} \rangle$ and $k'' = \langle k_{right}, k_{right} \rangle$). Note that this eliminates the previously required knowledge of an encrypted Trojan key component. Again k_{left} and k_{right} have been made independent of each other and can both be attacked as single length keys (since a double length key with repeated halves is equivalent to a single length key).

2.5.6. Key Separation

The mechanics of a key separation attack depend on the method by which key separation is enforced. (Obviously if key separation is not enforced then the attack follows trivially as described in Section 2.2.2). Generally the attacks occur at points where different key types have the potential to 'interact' or where the type of a key can be manipulated. Typical areas of concern include

- shared (or common) functionality between key types,
- transitivity between key types (a poor key type system), and
- methods that manipulate the type of a key (type casting).

2.5.6.1. Shared (or Common) Functionality between Key Types

In our typical API, we are very restrictive in what functionality is available to a number of different key types. The only call that will allow (i.e. accept) keys of different types is the key test or check value function (which simply encrypts the zero binary string using the key). Even this innocuous looking function has the ability to break separation between key types. For example, we can use it to

test whether two keys of different types have the same clear key value. This is possible, since the check values of equal keys are necessarily the same. Consider the two encrypted tokens $T_1 = e_{MK \oplus CV_1}(K)$ and $T_2 = e_{MK \oplus CV_2}(K)$. They represent encrypted keys of different types (CV_1 and CV_2 respectively). Despite the fact that they have the same clear key value, the encrypted tokens are necessarily different due to the different control vectors. However, the check value function gives us a method to test for equality since $check\_val(T_1) = e_K(0) = check\_val(T_2)$. Thus we have broken the separation between T_1 and T_2. (Note that, since the check value is typically a truncated set of bits, this is not a sufficient check). This attack can be extended using other methods. Together with key conjuring or, alternatively, the ability to create a large set of derived or related keys, one could search for such a collision.

2.5.6.2. Transitivity between Key Types (Poor Key Type System)

Some APIs allow the translation of encrypted data from keys of one type to keys of another type. This transitivity between key types potentially creates an opportunity that could be exploited by an attacker. Some attacks are detailed in [Bo01], [AB01]. However, since our standard API does not include this functionality, we do not consider it further.

2.5.6.3. Type Casting

Our focus for this discussion is the use of control vectors to enforce key type and usage. Since key separation is effectively enforced by the encrypted key tokens, it is difficult to manipulate the key type of an existing key. Thus one naturally considers the points at which keys enter the system for an opportunity to manipulate the type. In particular, key establishment through the use of shares and key import deserves considerable attention. For example, combining the three key shares k_a, k_b and k_c yields the key $k = k_a \oplus k_b \oplus k_c$ that is returned as the encrypted key token $T = e_{MK \oplus CV}(k)$. If we choose this to be a key encrypting key (a KEK) then $CV = CV_{KEK}$ and $T_{KEK1} = T = e_{MK \oplus CV_{KEK}}(k_{KEK})$. Consider the effect of replacing one of the shares (without loss of generality we replace k_c) by a new share $k_c' = k_c \oplus CV_{old} \oplus CV_{new}$. A new encrypted key $T_{KEK2} = e_{MK \oplus CV_{KEK}}(k \oplus CV_{old} \oplus CV_{new})$ is obtained.

Let $T_{OLD} = e_{MK \oplus CV_{old}}(k)$ be an encrypted key token. Suppose we export T_{OLD} under T_{KEK1} and then re-import under T_{KEK2}, but specify a new key type CV_{new} during the import process. Thus the exported key in internal form is $T_{exported} = e_{KEK1 \oplus CV_{old}}(k)$. However since

$KEK1 = k = KEK2 \oplus CV_{old} \oplus CV_{new}$, the exported token can be rewritten as $T_{\exp orted} = e_{KEK1 \oplus CV_{old}}(k) = e_{(KEK2 \oplus CV_{old} \oplus CV_{new}) \oplus CV_{old}}(k) = e_{KEK2 \oplus CV_{new}}(k)$, which, when imported under T_{KEK2}, yields $T_{NEW} = e_{MK \oplus CV_{new}}(k)$. Thus we have obtained two keys in the system of different types but with the same clear key value.

This method of changing the key type is a slight variant of the IBM 'pre-exclusive-or' type casting method. It is documented as a method to import keys from systems that do not support control vectors. Bond first identified it as vulnerable to misuse and attack [Bo01]. There exist other methods to create the desired differences between keys. These include the use of key offsets and other key derivation and manipulation functions.

The key import function offers a more obvious threat. Many external key tokens do not include type information cryptographically bound to the encrypted keys. Typically the user specifies the type of key when it is imported. This approach is exceptionally vulnerable to separation attacks. Unfortunately it is common – perhaps largely due to the ISO/ANSI standards that cover key exchange [ANSI X9.17]. No type information is specified in the standard. This is discussed again in the next section covering 'mixed systems'.

Various approaches have been adopted by control vector based APIs in an attempt to achieve interoperability with systems that do not support control vectors. The simplest approach is to allow the API to ignore the use of control vectors for the import function (as in the NO_CV import option for the ICSF API [ICSF]). This is a blatant security threat and can be managed through the use of electronic access control.

2.5.7. Mixed Systems

One of the challenges facing API designers is secure interoperability with other crypto coprocessors. A real world system, involving multiple parties transacting, inevitably requires interaction between different crypto coprocessors. One must be conscious of non-homogenous systems where devices from different manufacturers interoperate. In this situation, one device may not enforce the restrictions or security precautions required by the device of another vendor. This is particularly true of the key exchange problem. For example, the only device that makes use of key separation using control vectors is the IBM 4758 secure coprocessor (IBM hold a patent on the use of control vectors). This means, that if another device (e.g. A Thales-Racal RG7000) is used to exchange keys with an

IBM 4758 secure coprocessor, then key separation may not be effectively enforced. Additional (non default) restrictions are required for the procedure.

The challenges of interoperability are typically addressed through the use of standards. For example the ANSI X9.17 Financial Institution Key Management has scope covering "both the manual and automated management of keying material for the wholesale financial services industry" including amongst others the "distribution of keying material to permit interoperability between cryptographic equipment" [ANSI X9.17]. Unfortunately, it does not take into consideration the key separation issue as well as many of the attacks listed above.

2.5.8. Untrustworthy Partners

Operating a cryptographic device in a system in which one or more of the role players have malicious intentions poses a security challenge, as they may have additional knowledge or powers as collaborators. In particular, one must be concerned about the situation where the adversary is also the administrator (or security officer) of the device. We shall refer to such a device as a subverted device. Consider the scenario where a key exchange key (KEK) is shared between the target device and a subverted partner. Should the malicious partner have knowledge of the clear value of the key, it is trivial to export any target key under the known key and recover the clear value thereof. This would compromise all exportable keys. Even without knowledge of the KEK, it is possible to inject known keys into the target device. This is done by first establishing a known key in the subverted device (using key shares for example) and then exporting it under the shared KEK. As before, by obtaining a known key-exporting key, the adversary can compromise all exportable keys in the system. There are a variety of like attacks that are similar in nature to the section on known keys.

Additionally a subverted device can be used to break other security features such as key separation. For example, the creation two key encrypting keys of the form k, $k \oplus CV_{old} \oplus CV_{new}$ on the subverted device, can be exchanged with a target device and then used to defeat the control vector typing of the keys on the target device as with the separation attacks (described in Section 2.5.6).

It is particularly challenging to defend against these types of attacks. One can help prevent the malicious party from gaining knowledge of shared key exchange keys by enforcing split knowledge between the parties. In addition, one must ensure that one only exchanges keys with devices that have been authenticated to be secure products of reputable vendors. However, if the adversary is assumed

to have security officer privileges, then it is perhaps unwise and naïve to assume that he will not gain knowledge of the shared key.

A useful precaution is the imposition of limitations on the types of keys that a KEK may import. For example, by specifying that a given KEK can only import data encryption/decryption keys – it is not possible to perform a key separation attack using only that KEK. Also, gaining knowledge of that KEK does not permit the attacker to compromise other aspects of the system beyond data encryption. This makes sense in terms of the threat model since the adversary can only compromise keys used to protect data shared with his system. And as we assume that the attacker as the administrator can compromise data protected on his system in any event, there is no increase in exposure.

However, none of the solutions are completely satisfactory. Consequently one is led to the suggestion of a trusted key type that cannot be compromised by the administrator of the device.

2.6. Solutions

A variety of attacks have been discussed, made possible by a multitude of features, often interacting with each other in subtle ways. While some features in isolation do not permit an attack, they are still weaknesses, which ultimately may form part of an as yet undiscovered attack.

The significance of access control cannot be overstated. A real world secure API will support functions that can be abused (such as the ability to import a clear key, whether in parts or not). These 'dangerous' functions are typically involved in the establishment of keys, the generating of PINs or the generation of PIN verification values. It is critical to protect these functions against exposure to a malicious user. API level electronic access control is just part of the solution, along with other 'industry standard' precautions (such as physical access control). There is a greater need for understanding the security implications of 'dangerous' functions. This is jointly the responsibility of both the vendor and customer. It may be true that vendors do not generally supply enough information to their customers (typically with respect of how to configure their crypto coprocessors 'securely'), and customers may be guilty of not rigorously examining the security of their particular configurations. We would welcome greater disclosure of information in this regard. Access control can be taken a step further by removing functions for different products (a 'hard' configuration of access control). For example, Prism produces a specialized Key Management System (KMS) crypto coprocessor, the intention of which is to manage all keys in the system centrally. In this type of system, only the KMS crypto coprocessor need have the specialized (and more 'dangerous') key management functions. Security can be concentrated on the KMS site, the only point in the system

where this functionality is available. It would also make sense to have a PIN management crypto coprocessor for the generation of PINs, verification values and PIN mailing.

It is desirable for a secure API to possess the ability to mark keys as not exportable (or, conversely, as exportable). Many keys will never need to be exportable. Ideally, the KMS should be the only point where keys can be exported (and stored in an exportable form). All other tokens, once imported (or shared) with another node, should not be exportable. Perhaps only the person who generated a key should have permission to export it.

Very restrictive type casting, either using IBM's Control Vectors or different master keys for different key types (e.g. RACAL LMKs), is desirable. Control vectors are far more flexible and are easily made very restrictive, whereas a new LMK is required to allow a new restriction. Also, typically the API specifies the 'categories' of LMKs and is somewhat limited. Alternative ways for obtaining restrictive type casting should be investigated.

The Key Part Import function is an inherently 'dangerous' but necessary command. It must be made impossible to tamper with the key part token, nor must it be possible to conjure one. Prism APIs currently require a check value to be supplied to calls that combine key parts (or shares) and thus ensures that the final key has the correct associated check value. In addition, key part entry is by means of a pin pad, directly connected to the crypto coprocessor. In such a system, a key part token is never available, or used by the system. Bond suggested in [Bo01] that, similarly, a check value be supplied to the Final Key Part Import command. Unfortunately, for a small check value, it may still be possible to modify the key. The attacker could perhaps simply iterate through all possible check values or could, alternatively, keep trying different final key parts until the complete key has the desired check value. Even requiring that the check value be submitted to the first Key Part Import call and verified after the final key part has been combined, is insufficient. An attacker may continually try final key parts, until he or she finds a match. The required length of check values is discussed later.

Another complication arises if it is possible to calculate a check value on a key part. Then, regardless of the previous precautions, one can produce a related key set with the property that finding one key allows one to calculate all keys (in this case key part).

An additional suggestion is that key part tokens should have a limited life span. By requiring a lifespan just long enough to combine components, one can prevent the key part from being used indefinitely or long enough for an attack. A simple implementation is to time stamp the key part.

It has been shown that the Key Test or Generate Check Value command can be used to effect parallelism for exhaustive key search, particularly by using conjured keys. There are also some additional unsatisfactory implications of encrypting a known pattern. Consider for example, the check value of a key encrypting key ($check\_val = e_{KEK}(0)$). Observe the effect of creating an external key token of the form $T = check\_val = e_{KEK}(0)$ and supplying it to a Key Import call using that same KEK. The external token is decrypted obtaining a NULL (zero) key, which is then returned in an internal token $T = e_{MK}(0)$. Thus we have obtained a known key in the system.

One solution (used by Prism) is to generate a check value only when a key is generated. Every time a given key is used, its check value is supplied along with the encrypted key token. The crypto coprocessor checks that the check value of the clear key equals the check value supplied. Some debate exists as to the best length of the check value. If it has length less than 4 bytes, it is trivial to search for the check value by repeated trial. A 4-byte value can also be found by exhaustive search, given current crypto coprocessors (50 days at 1000 tps). In this scenario, it is not feasible though to generate a 'large' number of conjured keys. On the other hand, more bytes of a check value that are available enhance an exhaustive key search (since we get fewer false witnesses). For keys that an attacker has no (or limited) other techniques for obtaining known plaintext-ciphertext pairs, this aids the attack.

As described by Bond in [Bo01], it is important to bind together key halves (or thirds) and to prevent key conjuring. In addition, it is important to bind the allowed functionality (of type) of the key to the key(e.g. with control vectors or LMKs). ECB (electronic code book) encryption of the halves (as in ANSI X9.17, CCA and the RACAL API) does not provide this. Nor does CBC (cipher block chaining) encryption in isolation since it remains possible to replace the halves and know the effect of the resultant key.

One solution to avoid the problem is to migrate to AES (Advanced Encryption Standard). Since it supports equal key and block sizes, there need be no binding issues to AES encrypting an AES key. However, this does not solve the key conjuring weakness.

A key-authenticating key (KAK) could be used to calculate a CBC MAC of the clear key and control vector (if present), hence not only binding the halves of the key but also providing protection against conjuring. However, since the DES MAC is 64 bits in length, it requires on average only 2^{63} queries to a crypto coprocessor to conjure a key or to replace a key half. So although this is an improvement

and practically secure (since 2^{63} queries to a crypto coprocessor (or even queries multiplexed to a number of devices) would take too long and hence is impractical), it remains a theoretical weakness for double length keys. For example, we could still take a known key and 'cut and paste' its halves into an unknown double length key (one half at a time). The work required to defeat the MAC is 2^{55} queries, following which we would have isolated an unknown half and thus required only 2^{55} DES operations to recover it. The entire key is recovered in 2^{57} operations. We could use two distinct KAKs or a 128-bit block cipher(like AES) to prevent this.

A similar alternative is to use a hash (possible encrypted, perhaps under a KAK). The hash can be long enough to prevent the same weakness. There exists an obvious relationship between the security of the hash and security offered by key length. The presence of such a hash would obviate the need for a 'standalone' calculate check value function.

These methods carry an overhead for the extra computation and may increase key or key token length.

2.7. Case Studies: The CCA API

Recently much publicity was generated in the media by the announcement that the CCA API can be hacked. This attack in question was due to Clayton and Bond [CB02] and was an extension of Bond's earlier work [Bo01].

2.7.1. The Cambridge Attack on the 4758 CCA API

2.7.1.1. Description

Developed by Clayton and Bond [CB02], this attack exploits a clever sequence of vulnerabilities. The goal of this attack is to generate a known exporter key (a key encrypting with the ability to export other keys). Having a known exporter key, one can export other keys (provided that they haven't been marked as not exportable). Since the exporter key is known, it is possible to recover all the exported keys. This work [CB02] is also noted for the development of a DES key search machine implemented in a field programmable gate array (FPGA) and costing less than $1000.

The ability to conjure related keys using the Key Part Import function is used to effect a parallel search. A fairly large number (2^{16}) of related keys are generated, and used to encrypt a known value. Using their 'DES cracking machine', they recover one of the related keys in an expected 2^{40} operations (or 25.4 hours on average for their cracker). The CCA API allows for double length keys

with repeated halves. Thus the parallel search can be used against double length KEKs. The lack of key binding allows the grafting of one of the known halves of a repeated onto double length keys. This modified key thereby allowing the attack to be extended to compromising existing double length keys. The attack proceeds as follows.

1. Generate a set $\{k_{data,i} \mid k_{data,i} = k_{data} \oplus \Delta_i, i = 1..2^{16}\}$ of related single length data keys using the Key Part Import function.
2. Using the data Encrypt call, calculate the ciphertext set $C_{data} = \{e_{k_{data,i}}(P), i = 1..2^{16}\}$ where P is a known value.
3. Perform a parallel search using the DES cracking machine against the ciphertext set C_{data}. This results in the discovery of some $k_{data,i}$. Since the value of Δ_i is known, we can calculate the value of k_{data}.
4. Generate a set $\{k_{\exp orter,i} \mid k_{\exp orter,i} =< k \oplus \Delta_i, k \oplus \Delta_i >, i = 1..2^{16}\}$ of related double length exporter keys (KEKs) with repeated halves using the Key Part Import function.
5. Using the Key Export call, calculate the ciphertext set $C_{\exp orter} = \{e_{k_{\exp orter,i}}(k_{data}), i = 1..2^{16}\}$ where k_{data} is the data key we recovered in step 3.
6. Perform a parallel search using the DES cracking machine against the ciphertext set $C_{\exp orter}$. This results in the discovery of some $k_{\exp orter,i}$. Since the value of Δ_i is known, we can calculate the value of $k_{\exp orter}$.
7. Use the known KEK $k_{\exp orter}$ to export all keys from the device.

2.7.1.2. Improving the Attack

An exporter key is a double length key and that only operates on other keys. Hence, to obtain the plaintext-ciphertext pair under each of their related exporter keys, as required, one first generates a known data key which is exported under each of the related exporter keys. This process represents half the work required to mount the attack.

This (additional) work can be avoided by observing that it is possible to use the Key Test function (CSNBKYT) to create a check value which is the encryption by the key of a 64 bit all zero data block. This is possible since the Key Test function accepts all key types. It also means that exactly the same

process can be used for all key types. This approach carries the additional requirement that the Compute Verification Pattern command is available. The benefit is that the attacker only has to collect half the amount of data and perform one less exhaustive key search. The revised attack is as follows.

1. Generate a set $\{k_{\exp orter,i} \mid k_{\exp orter,i} =< k \oplus \Delta_i, k \oplus \Delta_i >, i = 1..2^{16}\}$ of related double length exporter keys (KEKs) with repeated halves using the Key Part Import function.
2. Using the Key Test call, calculate the ciphertext set $C_{\exp orter} = \{check\_val_i = e_{k_{\exp orter,i}}(0), i = 1..2^{16}\}$.
3. Perform a parallel search using the DES cracking machine against the ciphertext set $C_{\exp orter}$. This results in the discovery of some $k_{\exp orter,i}$. Since the value of Δ_i is known, we can calculate the value of $k_{\exp orter}$.
4. Use the known KEK $k_{\exp orter}$ to export all keys from the device.

The Key Test function is perhaps the simplest function available to harvest data, and should be faster than any alternatives. Benchmarked on the 4758 Model 001/013 (i.e., the old version of the card), one can perform approximately 190 tps for a single length key, and a slightly lower 170-180 tps for a double length key. Values differ slightly depending on the use of the number of threads (the original exploit code did not employ the modules multi-tasking ability). Nonetheless 2^{16} calls could be made in between 5.5 and 6.5 minutes. The newer model (003/023) should be even faster. Nonetheless, this is a significant improvement from the original 20 minutes.

2.7.1.3. Variants of the Attack

The CCA API has additional functions, not found in our standard API that can be used in this attack. The Diversify Key command using session x-or (Generate Diversified Key (SESS-XOR)) is an alternative to using the Combine Key Parts for creating a related key set. It has a limitation in that only DATA, DATAC, MAC, DATAM, MACVER and DATAMV keys can be modified. Note that none of these key types can be used to export other keys.

2.7.1.4. Extending the Attack

The attack may be extended to triple length DATA keys by noting the previous discussion and the section on Combined Attacks (Section 2.4.3).

To attack a double length key with unequal halves, one can employ the Related Key attack against 2 Key 3DES. However one must satisfy the additional data requirements (for which the Key Test function would not be suitable). It has the advantage of not requiring the ability to generate/obtain a replicate key.

2.7.1.5. This is not the best attack

IBM indicated a far more powerful attack (requiring the same conditions) in its correspondence with Bond and referenced in his work [Bo01].

2.7.2. IBM's attack on the 4758 CCA API

This attack is in essence a key separation attack. It uses the Key Part Import function to create a related key that defeats the intended key separation mechanism.

2.7.2.1. Description

1. Using the Key Part Import function, generate an unknown related KEK key pair (UKEK, UKEK ⊕ EXPORTER ⊕ DATA) where (EXPORTER and DATA are the control vectors for key exporting keys and data keys in external form.
2. Conjure two forms of an encrypted unknown random key (EURK) in external format, one with an EXPORTER control vector, the other with the DATA control vector. (We note that the DATA control vector in an external key consists of only binary zero's, that is, it is a NULL or zero vector.)
3. Import the external EXPORTER key under UKEK obtaining

   $URK1_{EXPORTER} = d_{(UKEK \oplus EXPORTER)}(EURK)$.
4. Import the external DATA key under UKEK ⊕ EXPORTER ⊕ DATA obtaining

   $URK2_{DATA} = d_{((UKEK \oplus EXPORTER \oplus DATA) \oplus DATA)}(EURK)$

   $= d_{(UKEK \oplus EXPORTER)}(EURK)$

The clear value of $URK1_{EXPORTER}$ is identical to $URK2_{DATA}$ – but the tokens are of different type. We rename them $URK_{EXPORTER}$ and URK_{DATA}.

Thus far, we have followed the traditional 'Pre-exclusive OR technique' for changing control vectors as described by the CCA documentation and used in the Cambridge attacks. The difference is the following step:

5. Export URK_{DATA} under $URK_{EXPORTER}$, obtaining URK in external form.

$$EURK^* = e_{(URK \oplus DATA)}(URK)$$

Again we note that the control vector for a DATA key in external form is NULL. Hence

$$EURK^* = e_{(URK)}(URK).$$

6. We take the encrypted external key (ERK) and supply it as *data* to a Decipher call using URK_{DATA}, obtaining as plaintext (P)

$$P = d_{(URK)}(EURK^*)$$
$$= d_{(URK)}(e_{(URK)}(URK))$$
$$= URK$$

Hence we know the clear value of key (as we should now rename URK as KRK or known random key). We also have the key as an exporter key.

Thus we can export any key (not marked as un-exportable) under our known key, and easily decrypt it.

2.7.2.2. What if the DATA Control Vector of an External Key was not NULL

If DATA control not null then it is not possible to use the data key to decrypt 'correctly' to the external key since

$$EURK^* = e_{(URK \oplus DATA)}(URK)$$
$$P = d_{(URK)}(EURK^*)$$
$$= d_{(URK)}(e_{(URK \oplus DATA)}(URK))$$
$$\neq URK.$$

There are a variety of ways to avoid this problem. Firstly, one could attempt to obtain URK encrypted without the use of a control vector (i.e, $EURK^* = e_{(URK)}(URK)$). Secondly, one could use the data key URK ⊕ DATA. Listed briefly, these methods include:

1. Start with URK a key part (URKP). Then, using Key Part Import, one can obtain URKP ⊕ CV, where CV is any control vector one desires. Hence one can export any key of any type, and use the associated data key URKP ⊕ CV_TYPE to decrypt it.
2. Use the Diversify Key command on the data key URK to obtain URK ⊕ CV_TYPE .

3. Use the function Control Vector Translate to translate the control vector from DATA to NULL. This has the effect of translating the key from ($e_{(KEK \oplus DATA)}(URK) =$) $e_{(URK \oplus DATA)}(URK)$ to $e_{(URK)}(URK)$.
4. Use the Key Translate function to translate ($e_{(KEK \oplus DATA)}(URK) =$) $e_{(URK \oplus DATA)}(URK)$ to $e_{(DATA)}(URK)$.

2.7.3. Comparison between the IBM and Cambridge Attacks

Both attacks 'effectively' recover an exporter key. They both require access to the following functions (termed the shared functions between the attacks)

- Key Part Import
- Key Import
- Key Export
- Decipher

Control vector translate can be used as a substitute for the Key Part Import in the IBM attack (and to some extent in the Cambridge attack). The IBM attack does not require Generate Replicate function. Thus the IBM attack requires fewer API entry points and is potentially less restricted. However, any arguments about the ability to 'use' the shared functions in a real world scenario, are equally applicable to both.

The IBM attack does not perform an exhaustive key search and so does not require specialized hardware. In addition, it does not require that a large amount of data be harvested from the 4758. The Cambridge attack requires 20 minutes of access to the device (before improvements).

2.7.4. Conclusion

The IBM attack is the attack of choice. The 'DES cracking machine' is actually irrelevant to this attack scenario. The effort required is trivial and the time required is minimal. This represents a serious threat *if* one can overcome the requirements of the attack and is able to mount it.

2.7.5. A New Attack with Fewer Restrictions

In this section we present a new variant of the original Cambridge attack in which we have removed many of the restrictions with regard to required functions that may be regarded as sensitive. This

version makes use of a bug (intentionally searched for) and discovered in the CCA firmware (found in Version 1.3.2 of CCA app for 4758 Model 01/13 and subsequently confirmed for later versions including 2.40 for 4758 Model 02/23). This attack requires the harvesting of 2^{n+1} data and an exhaustive search requiring $2^{56 - n + 1}$ DES operations ($2^{55 - n + 1}$ on average).

To generalize the attack, two requirements must be accomplished. A mechanisms to effect parallelism is required to speed up the exhaustive key search, followed by a technique to isolate the separate keys in a multi-length key.

We avoid using known key difference methods to obtain parallelism, since the requirements to achieve this are onerous, and not easily extendable to, say, other crypto coprocessors or APIs. Hence we are satisfied with the option of attacking many unknown (and unrelated) keys simultaneously. In the absence of a readily available Key Generate function, it still remains easy to generate many keys by using key conjuring techniques (e.g. use a random string as the encrypted key). We need to be able to encrypt (or decrypt) a known value with these keys. For DATA keys this is trivial. One need only make an Encipher or Decipher call. Some other key types are equally easy (like ENCIPHER, DECIPHER and MAC keys), but most are more difficult (for example, KEKs only operate on other keys – hence we would require a known key encrypted under the KEK).

For now, we focus solely on DATA keys. We generate many keys (say 2^n) and encrypt a known pattern with each of the keys. We then perform an exhaustive search (possibly using a DES cracking machine) as in the previous attacks. This is practical for single length keys, since we will find a key in 2^{55-n} search operations on average, which, as demonstrated by the Cambridge attack, is very feasible.

Unfortunately, the naïve extension to attacking double length keys is impractical, requiring a 2^{111-n} search. Hence we need to attack halves separately. (Note that the Cambridge attack targeted double length keys with repeated halves, hence effectively a single length key.

It would be ideal to be able to create a double length with repeated halves. Suppose we could simply duplicate the left half of an encrypted double length key as the right half. In order to achieve this, we require that the left and right halves of a double length key be encrypted under the same variant of the KEK (i.e. $KEK \oplus CV_{left} = KEK \oplus CV_{right}$ or $CV_{left} = CV_{right}$). Unfortunately, IBM has wisely typed the control vector halves to ensure that this is not possible. It should be noted, that there are two exceptions to this rule. Firstly, the control vector translate function in the API provides a means to modify the control vectors.

Secondly, DATA keys in external form have the property that they have no control vector(or a null control vector). Thus $CV_{left} = CV_{right} = 0$. This gives us the ability to create a double length key with repeated halves, by repeating the halves of a DATA key in external for and then importing it. Hence we can attack all double length DATA keys, requiring only the Key Import and Encipher/Decipher commands.

Unfortunately, this attack is only applicable to keys that can be used in the Encipher/Decipher command (or similar functions like Mac Gen) and the trick to 'create' a double length replicate key is only applicable to keys without a CV or for which $CV_{left} = CV_{right}$.

The question remains whether we can obtain the encryption of a known pattern by any key type. The answer lies in the Key Test command. The purpose of the command is to provide a method to verify the value of a KEY without revealing any information about the key. It is a function common to many APIs (though sometimes referred to as the calculation of a check value). For example, calling the Key Test command with the ENC-ZERO key word encrypts the 64 bit binary zero string under a given key and returns the first four bytes of the answer as the verification pattern. This allows a user to confirm that a key is unaltered or the 'correct' key by comparing a previously calculated value with the new check value returned.

It is immediately obvious that this could be used in the attack. A complication arises since the check/verification value is limited to four bytes in length and hence will allow a false witness with probability 2^{-32}). Hence in testing all 2^{56} keys, we will be left with 2^{24} possible values for the key. Thus we would need to apply an additional test of some form such as a second call to Key Test but this time using another option (e.g. CCA DES Key Verification Technique) to eliminate the false witnesses.

It would be preferable not to have to perform a second call for every key in the set of 2^{16} keys to eliminate false witnesses, but it is only doubles the number of calls required to be made against the target crypto coprocessor. In the case of the CCA API, we would probably be better served by using another of the options (such as the CCA DES Key Verification Technique), which return an 8-byte verification value, hence making it unlikely that we would encounter a false witness.

The check value function is a common function across many APIs and is generally not viewed as a dangerous function (as opposed to key part imports for example). Typically, the check value is calculated by encrypting a 64 bit binary zero. Some APIs can return an 8-byte check value, but 3 or 4

byte values are common. It is possible to use other verification methods such as hashes. In that case, we would need to customize the DES cracker to use the associated algorithm (as opposed to a simple DES encipherment operation). This may reduce the speed of the cracker.

Again we have used parallelism to reduce the search effort, but it remains currently viable only for single length keys. How do we isolate the halves of a double length key (without using key generate replicate)? As pointed out by Bond, the halves are not strongly bound together. If it were not for the fact that the control vector halves used to encrypt the halves of keys are different, we could simply duplicate the halves as we did for the DATA key attack above.

We need to find functions that would accept a key for which $CV_{left} = CV_{right}$. As mentioned earlier, this should not be possible due to dedicated bits in the control vector, which distinguish between left and right halves.

This leads us to the CCA bug. The first candidate for a function that might accept a key with both halves of the control vector the same, is Key Test command, because it is the one call that operates on all keys and hence must accept many different types of keys and hence many different control vectors. It is also immaterial whether the key is single or double length. Thus we speculated that there might be scope for the programmers to have been less than rigorous in their implementation. In addition, it ties in well with our earlier work on key search and we would like the ability to operate on all key types.

On version 1.32 of the CCA firmware (the most recent version available on the 4758-001/013) if the control halves are repeated (so CV' = $<CV_{left}, CV_{left}>$) the Key Test function did not return a Control vector violation error. Hence given a double length encrypted key EK = $<EK_{left}, EK_{right}>$ with control vector CV = $<CV_{left}, CV_{right}>$, it is possible to create the encrypted key EK' = $<EK_{left}, EK_{left}> = EK_{left}$ with control vector CV'. Hence it is possible to isolate the first half of the key from the second half. We now have mechanism for isolating the keys.

Conjuring 2^n values for EK', and submitting them to Key Test, we can recover K_{left} in 2^{55-n} DES operations. Conjuring 2^n values for EK_{right}, and submitting EK to Key Test, we can recover K_{right} in 2^{56-n} DES operations. The total work to completely recover both halves of the key is $2^{55-n} + 2^{56-n} = 3 \cdot 2^{55-n}$ DES operations.

We note that even without the ability to effect parallelism, it is a valuable technique to recover a double length key with 2 single length searches.

2.7.6. Summary

If Key Test command is available, then :

- A given single length key can be searched in 2^{55} operations on average (with a data set containing 1 plaintext-ciphertext pair)
- A given double length key can be searched in $3 \bullet 2^{55}$ operations on average (with a data set containing 2 plaintext-ciphertext pairs)
- A known single length key can be generated in 2^{55-n} requiring with 2^n plaintext-ciphertext pairs.
- A known double length key can be generated in $3 \bullet 2^{55-n}$ operations requiring 2^{n+1} plaintext-ciphertext pairs.

If Key Export command is available, then one can use a known exporter key generated as above to export all keys that have not marked as ‘not exportable’.

3. The Financial Cryptographic API

3.1. Why Are We Interested?

The security of Automatic Teller Machine (ATM) and Electronic Funds Transfer at the Point of Sale (EFTPOS) networks was perhaps the most significant driver for modern cryptography, responsible for drawing it out of the secret government and military departments into the commercial world. It has been described as the so-called 'killer application' of cryptography [An01]. As a mature technology in a young field it presents a unique opportunity to study the phenomenon of the degradation of security over a prolonged time period.

Secondly, the concept of a 'PIN' is internationally understood and accepted. It competes with the use of passwords for the title of the most important identification technique. After all, everyone with a bank account and card has a PIN number and this was true before the advent of the Internet age.

The scale of use is impressive. It includes bank, credit and debit cards, card issuing banks and cards associations such as VISA, MasterCard, Europay and American Express. An enormous amount of money is protected daily by these security mechanisms.

3.2. Financial Security 101

3.2.1. Terminology

The account holder is the individual or entity that holds the account at the bank. We sometimes refer to the account holder as the user or customer. The bank, which issued the card and PIN to the account holder, is known as the issuing bank while the bank responsible for initially acquiring the transaction from the user (i.e., the account holder or customer) is known as the acquiring bank. PIN is an acronym for Personal Identification Number. This is a number that is used by the account holder to verify his identity to the issuing bank and is used as a means of authenticating a transaction. Associated with each user's account is an identifying account number known as the Personal Account Number (PAN). The bank uses the PAN to identify which account is being used in the transaction. PINs typically consist of between 4 and 12 decimal digits. Since knowledge of the PIN gives the right to transact with the account, it is imperative that the PIN be kept secret. This is done using encryption.

Prior to being encrypted, the PIN is formatted into an 8-byte buffer known as the clear PIN block (PB). The encrypted results is termed an encrypted PIN block (EPB).

3.2.2. Architecture

The diagram below (Figure 3-1) is a simple representation of an ATM or EFTPOS network. On the extreme left is the customer facing device, typically an ATM or point of sale (POS) device (although it could just as easily be a web browser, mobile phone or another of the new generation transacting interfaces). This device is connected to the bank that 'operates' the device, i.e., the acquiring bank. Note the user does not necessarily have to hold an account with the acquirer and for this illustrative example we assume that this is not the case. The acquiring bank must thus send the transaction to the bank with which the user does have the account (i.e., the issuing bank). The transaction is passed to an intermediary switch, which then 'switches' the transaction to the acquiring bank. Sometimes the switch does not have a direct link to the acquiring bank and so sends the transaction to second switch.

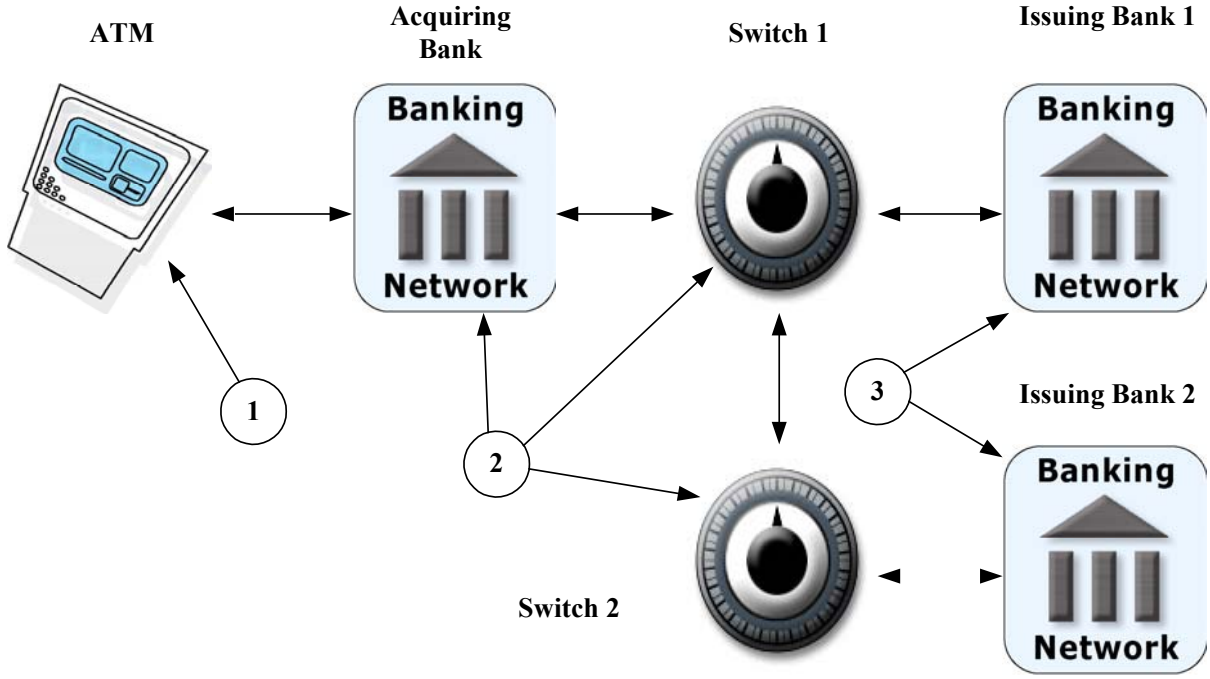

Figure 3-1: Simple representation of the EFT network

The reason for this architecture lies in the encryption methods used to protect the PIN. ATM and EFTPOS networks were first established in the late 70's and used symmetric key encryption only (typically DES). Thus any two entities that wanted to communicate with each other, first had to establish a shared key - normally done manually using key shares. The two entities are said to share a zone key. This process had three difficulties. Firstly a given bank could not transact with a (new) bank with which it had not yet established a shared key. Secondly, the process of establishing shared

keys with all the institutions with which it wanted to transact, was time consuming and costly. Finally, it required the secure storage of a large number of keys. Thus the use of switches arose to combat these problems. An acquiring bank needed only to share keys with one or more switches, which then provided the link from the issuing bank to the acquiring bank.

Let us map the path of the account holder's PIN through the system and identify the operations performed upon it. At point 1 on the diagram (the customer facing device) the user enters his or her PIN number. The PIN is immediately encrypted in the PIN entry device using the key shared with the acquiring bank and then sent to the acquiring bank. The bank must now forward the encrypted PIN to the switch. But the key shared between the bank and the switch is different, belonging to a different key zone from that shared by the acquiring bank and ATM. Thus the acquiring bank must first decrypt the PIN under the first key and then re-encrypt it under the key shared with the switch. The same process occurs at the switches and is marked by a 2 on the diagram. Finally, when the encrypted PIN arrives at the issuing bank, the bank must confirm that it is in fact the correct PIN associated with the account. Thus we have identified the three basic functions that must exist in our financial API, namely

1. PIN encryption,
2. PIN translation between zone keys and
3. PIN verification.

3.2.3. PIN Algorithms

3.2.3.1. Encryption

The PIN encryption is a trivial process. The PIN is formatted into an 8 byte clear PIN block and then encrypted using DES or 3DES (the block size of the cipher accounts for the 8 byte length of the clear PIN block). However, there exist a multitude of recognised formats for the clear PIN block.

3.2.4. PIN Block Formats

A fairly standardized notation is used to describe the possible formats whereby characters are used as placeholders for possible decimal and hexadecimal values. We reproduce the notation here for completeness.

X'y' denotes the hexadecimal character y and is equivalent to y_{16}. For example, X'0' = 0_{16} = 0000_2, X'1' = 1_{16} = 0001_2 and X'F' = F_{16} = 1111_2.

P = A 4-bit decimal digit that is one digit of the PIN value.
C = A 4-bit control value. The valid values are X'0' and X'1'.

```
L = A 4-bit hexadecimal digit that specifies the number of PIN digits.
F = A 4-bit field delimiter of value X'F'.
f = A 4-bit delimiter filler that is either P or F, depending on the
    length of the PIN.
D = A 4-bit decimal padding value. All pad digits in the PIN block
    have the same value.
X = A 4-bit hexadecimal padding value. All pad digits in the PIN block
    have the same value.
x = A 4-bit hexadecimal filler that is either P or X, depending on the
    length of the PIN.
R = A 4-bit hexadecimal random digit. The sequence of R digits can each
    take a different value.
r = A 4-bit random filler that is either P or R, depending on the
    length of the PIN.
Z = A 4-bit hexadecimal zero (X'0').
z = A 4-bit zero filler that is either P or Z, depending on the
    length of the PIN.
S = A 4-bit hexadecimal digit that constitutes one digit of a sequence
    number.
A = A 4-bit decimal digit that constitutes one digit of a user-specified
    constant.
```

One of the simpler formats is known as VISA Format 3. This format specifies that the PIN length can be a minimum of 4 digits and a maximum of 12 digits. The PIN start from the left most digit and ends by the delimiter ('F'). An example of a 6 digit PIN is:

```
VISA Format 3 PIN Block (PB) = PPPPPPFXXXXXXXXX
```

The most important (and complicated) format is ANSI X9.8 – the format specified by the ANSI standards on retail banking [ANSI X9.8]. This format is also known as ISO-0, VISA-1, VISA-4 and ECI-1. It is unique in that the 12 rightmost PAN digits are prepended with zeros and exclusive-ored (xored, $\oplus$) with the PIN to yield the clear PIN block

```
P1 = CLPPPPfffffffffFF
P2 = ZZZZAAAAAAAAAAAA
ANSI X9.8 PIN Block (PB) = P1 ⊕ P2
where C = X'0' and L = X'4' to X'C'
```

This was done for two reasons. It binds the account number to the PIN block and so provides a measure of protection against an incorrect account number being subsequently used. Secondly it diversifies the encrypted PIN block according to the account number. Even though two users may have the same PIN number, the associated encrypted PIN blocks will bear no resemblance since they have different account numbers. This prevents the so-called "code book" attack, in which an adversary makes use of a dictionary of encrypted PIN blocks.

The list of PIN formats includes:

- ISO-0 (ANSI X9.8, VISA-1, ECI1)

- ISO-1
- ISO-2
- VISA-2, VISA-3, VISA-4
- IBM 3624, IBM 3621, IBM 4700
- ECI-2, ECI-3
- Docutel

3.2.5. PIN Algorithms Continued

3.2.5.1. Translation and Reformatting

The translation function merely decrypts the PIN encrypted under the input key and then re-encrypts it under the output key before transmission to the target entity. What happens should this entity not support the PIN format used? The simple solution is an extension to the translate function called reformat which allows the user to specify a different format of the output PIN block. Should the output format be ANSI X9.8, the function needs to be able to specify a PAN as well.

3.2.5.2. Generation and Verification

PIN generation can be achieved in a variety of ways, either by means of an algorithm or chosen by a user (or both). An example of the algorithmic approach is the IBM 3624 PIN Generation Algorithm, which generates a PIN based on account- or person-related data, called the validation data. The validation data is enciphered under a PIN generating key, decimalised, and the desired number of digits selected as the PIN.

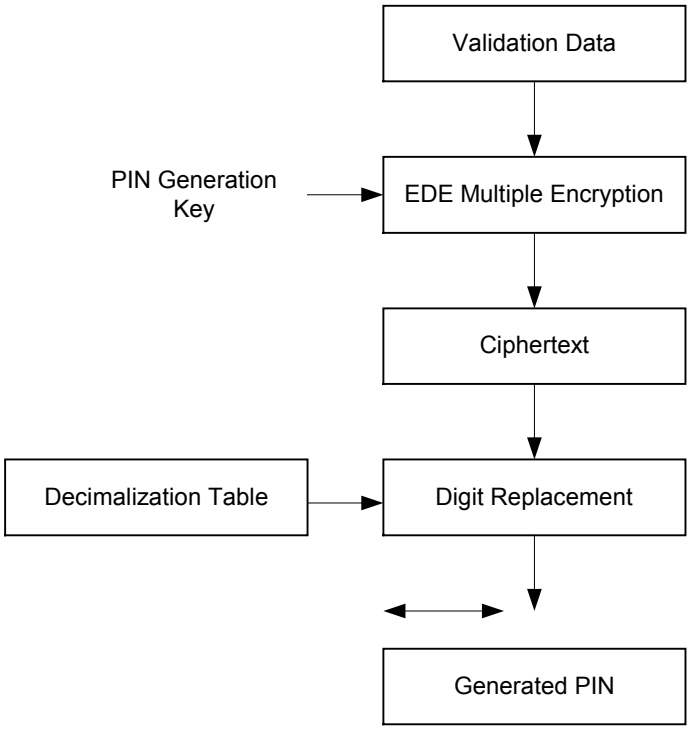

Figure 3-2: PIN generation algorithm

Verification is achieved by repeating the process (except that now the key may be called the PIN verification key) and comparing the calculated PIN with the PIN that is extracted from the encrypted PIN block.

3.2.5.2.1. Offsets

Generating algorithms can be extended to cater for chosen PINs. This is achieved through the use of offsets, which relate the algorithm generated PIN to the chosen PIN. The normal generation algorithm is run (with the same parameters) and the output called an intermediate PIN. The offset is calculated by subtracting modulo 10 some subset of the intermediate PIN digits (usually either the leftmost or rightmost digits) from the chosen PIN digits.

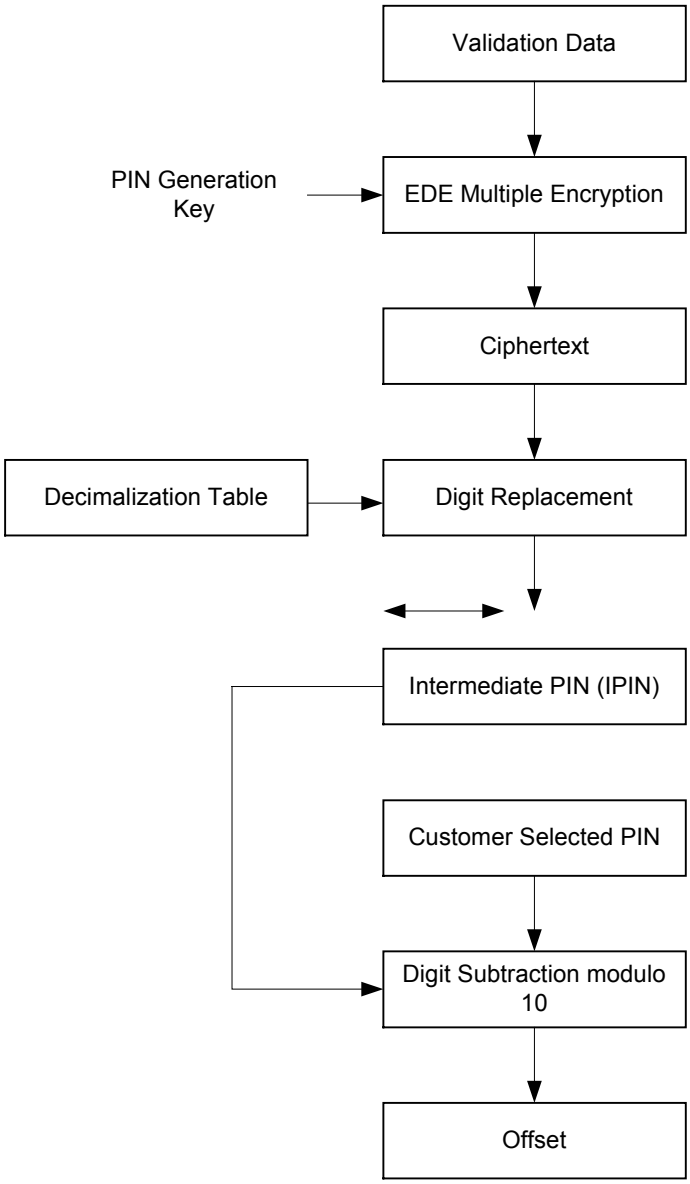

Figure 3-3: PIN offset generation algorithm

Verification requires that the offset be supplied to the call as well.

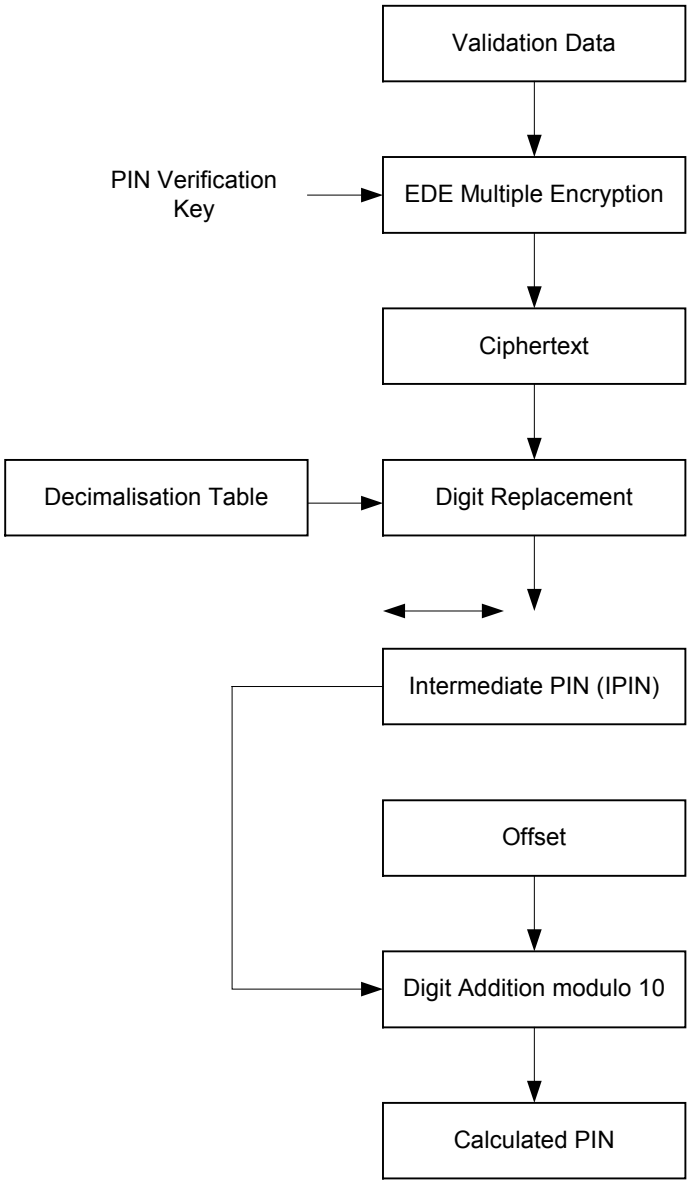

Figure 3-4: PIN verification algorithm

3.2.5.2.2. PIN Verification Values (PVV)

For a chosen PIN, the PVV generation algorithm accepts a PIN (either clear or encrypted) as input and must calculate a PIN verification value (PVV), which is stored. The algorithm is not unlike the PIN generation algorithm described in Figure 3-2 but with two significant differences. Firstly, the validation data has been replaced by the transformed security parameter (TSP). The VISA PVV algorithm encrypts the concatenation of 11 PAN digits, a key index (a digit in the range 1 to 6) and the first 4 digits of the PIN extracted from the encrypted PIN block (i.e. TSP = PAN || Key Index || PIN). Secondly, the result is decimalised in a unique way. Scanning from left to right, the first 4 decimal

digits encountered are returned as the PVV. Should there be less than 4 digits, a second scan is performed, in which the non decimal digits are converted to decimal digits (by subtraction modulo 10), and the first 4 resulting digits returned as the PVV.

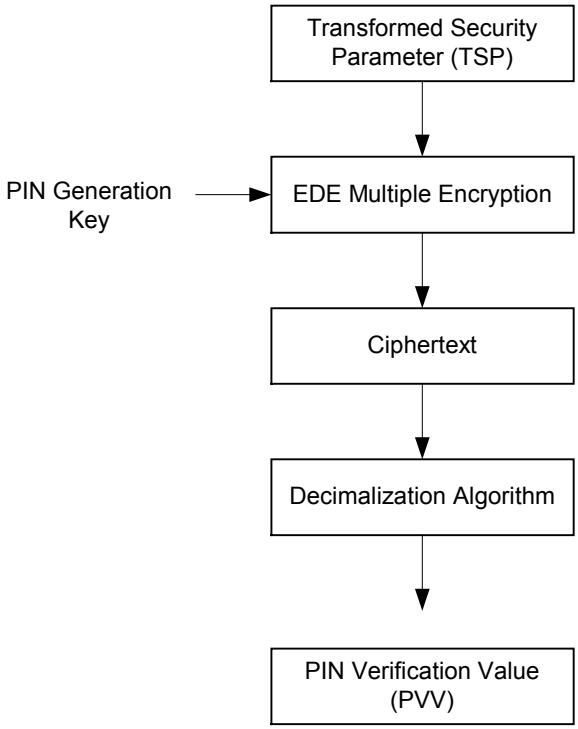

Figure 3-5: PVV generation algorithm

Verification involves extracting the PIN, calculating the verification value and comparing it to a supplied PVV. An example is the VISA PIN verification algorithm.

3.3. The Standard Financial API

3.3.1. Case Study of the CCA API

The PIN translate and reformat command in the CCA API (Release 2.40) is called CSNBPTR and has the following prototype. Note that the prototype has been slightly simplified.

```
CSNBPTR
{
  return_code,                    //Output – error code
  reason_code,                    //Output -  description of error
  input_PIN_encrypting_key_identifier,     //Input – old key
  output_PIN_encrypting_key_identifier,    //Input – new key
  input_PIN_profile,              //Input – the original PIN block format
```

```
  input_PAN_data,                //Input – the original PAN
  input_PIN_block,               //Input – the encrypted PIN block
  rule_array_count,              //Input
  rule_array,                    //Input
  output_PIN_profile,   //Input – the desired PIN block format
  output_PAN_data,      //Input – the desired PAN
  output_PIN_block      //Output
}
```

The `return_code` is a general error code with the `reason_code` indicating the reason for the error. The key identifiers (`input_PIN_encrypting_key_identifier` and `output_PIN_encrypting_key_identifier`) are the key tokens. The pin profiles specify the PIN block formats (e.g. ANSI X9.8, etc) and the `PAN_data` contains the PAN. The encrypted PIN blocks are called the `input_PIN_block` and `output_PIN_block` respectively.

The `rule array` is a list of parameters for the function (the `rule_array_count` indicates the number of parameters in the list). For this function, the caller can specify either `TRANSLAT` or `REFORMAT.` Translate indicates that only the PIN encrypting key is changed in the function, while the reformat option is used to change some combination of key, PIN block format or PAN.

The PIN verify command is known as CSNBPVR.

```
CSNBPVR
{
  return_code,                   //Output – error code
  reason_code,                   //Output – description of error
  PIN_encrypting_key_identifier, //Input – PIN encrypting key token
  PIN_verifying_key_identifier,  //Input – verification key token
  PIN_profile,                   //Input – PIN block format info
  PAN_data,                      //Input – the PAN
  encrypted_PIN_block            //Input – encrypted PIN block
  rule_array_count,              //Input – a count of the parameters
  rule_array,                    //Input – parameters for the call
  PIN_check_length,              //Input – length of PIN to check
  data_array                     //Input – the validation (or other) data
}
```

Here the `rule_array` indicates which algorithm to use to verify the PIN (e.g. IBM, IBM Offset, VISA PVV, etc). The `data_array` contains the validation data or transformed security parameter (depending on the algorithm).

3.3.2. Case Study of the Thales-Zaxus-Racal API

The Thales API supports a number of PIN encrypting keys types. These are

- *Terminal PIN keys (TPK)* which are used in the PIN encrypting terminal or device,

- *Zone PIN key (ZPK)* which are shared between entities sharing a key zone, and
- *Local Master keys (LMK)*.

The Pin verification key is known as the PVK.

There are a number of PIN translation functions that allow translation between each of the above key types. The basic prototype (ignoring key types) is as follows:

```
PIN Translate
{
  //inputs

  Source Key,               //source TPK/ZPK/LMK key
  Destination Key,          //destination TPK/ZPK/LMK key
  Maximum PIN Length,       //Value of 12
  Source PIN Block,         //the encrypted PIN block under the source key
  Source PIN Block Format,  //the format code for the source PIN block
  Destination PIN Block Format,  //the format code for the dest PIN block
  Account Number,           //the rightmost 12 digits of the PAN

  //outputs

  Error Code,               //e.g. 00 = no error; 20 = pin block data error
  PIN Length,               //the length of the returned PIN
  Destination PIN Block,    //the encrypted PIN block under the dest key
  Destination PIN Block Format
}
```

While the function does not support different source and destination account numbers (PAN), one can effect this operation by simply calling this function twice in sequence. On the first call we convert from the ANSI X9.8 format with associated source account number to a format without an account number. On the second call we translate back to ANSI X9.8 format, but specify the desired destination PAN.

The two PIN verification prototypes are also standard

```
PIN Verification Using Offsets
{
  //inputs

  PIN encrypting Key,       //TPK/ZPK/LMK
  PIN verification Key,     //PVK
  Maximum PIN Length,       //Value of 12
  Encrypted PIN Block,      //the encrypted PIN block
  PIN Block Format,         //the format code for the source PIN block
  Account Number,           //the rightmost 12 digits of the PAN
  Decimalization Table,     //the decimalization table
  Validation Data,          //the validation data
  Offset,                   //the offset

  //outputs
```

```
  Error Code                //e.g. 00 = no error; 01 = verification failed
}

PIN Verification Using PVV
{
  //inputs

  PIN encrypting Key,       //TPK/ZPK/LMK
  PIN verification Key,     //PVK
  Maximum PIN Length,       //Value of 12
  Encrypted PIN Block,      //the encrypted PIN block
  PIN Block Format,         //the format code for the source PIN block
  Account Number,           //the rightmost 12 digits of the PAN
  PVKI,                     //between 0 and 6
  PVV,                      //the pin verification value

  //outputs

  Error Code                //e.g. 00 = no error; 01 = verification failed
}
```

Hence we can conclude that the Thales PIN translate function matches our standard financial API functionality.

3.3.3. On the Validity and Applicability of the Standard Financial API

After comparing the functionality of the proposed standard financial API with the equivalent functionality found within the devices that dominate this market (including the CCA, Thales and Atalla APIs), we can conclude that our prototypes are indeed accurately reflective of the industry norms. We can thus proceed to the analysis stage with confidence knowing that the results will be applicable to the majority of relevant APIs.

3.4. Known Attacks and Assumed Level of Security

3.4.1. Exhaustive Key Search (Brute force)

Exhaustive key search is (almost) always a possibility. It provides an understood upper bound on the level of security that is hoped to be achieved - and so any technique that does not improve on it is of little interest. The strength of a secure algorithm is measured by the length of the key (effectively the size of the key space required to be searched). The current standard PIN transaction systems are 3DES [ANSI97, ANSI X9.8, ANSI X9.17, ANSI X9.24] - although many (typically historic) single DES systems still exist. Note that this is not an attack against correctness. In our current reality, a 3DES system is practically immune to such an attack.

3.4.2. Exhaustive Pin Search

The PIN space is considerable smaller than the key space. It is thus critical to prevent an adversary from being able to mount such an attack (since he would surely and rapidly succeed). It is probably for this reason that the ANSI X9.8 standard (Personal Identification Number (PIN) Management and Security) states "The system shall not be capable of being used or misused to determine a PIN by exhaustive trial and error". One obvious potential weakness would be any implementation, which accepts clear PINs to either the PIN verification or generation functions. An API that allows such functionality typically restricts it to a secure or authorized mode, thereby ensuring that it is not misused. At some point, the user has to be able to enter a PIN in the clear. However, this is a manual process (usually at a trusted interface) and hence cannot be easily automated. In addition, it is common practice to detect and prevent a user's trying of many combinations.

3.4.3. The Code Book Attack

To mount such an attack, the adversary will build up a 'code book' containing every PIN and the result of that PIN encrypted under a given key. To recover an unknown encrypted PIN, the attacker simply consults the codebook to find the encrypted PIN block and hence identify the associated PIN. Should the PIN be encrypted under a different key to the one used for the codebook, the attacker simply translates it to encryption under the codebook's key. As with the exhaustive key search, it should not be possible for the attacker to build up such a codebook and the same practical limitations apply.

For an ANSI X9.8 format n digit PIN, 10^n encrypted PIN blocks are required to be stored (for a given account number). This represents a trivial memory requirement and is insignificant to search. Naively, attacking formats containing random padding would appear to require a larger codebook, but this is not the case. All formats are 'equally' vulnerable, since the format can be changed in the translate (reformat) call (to the weakest one). The same technique can be used to 'eliminate' the variation offered by the PAN. It is noteworthy that regardless of format, key and pan, all encrypted pins are potentially vulnerable to a single codebook.

3.4.4. Key Separation Attacks

Key separation is a mechanism, which enforces that a given key is used as intended (described in Section 2.5.6). The well known attack scenario is supplying an encrypted PIN block and the PIN encrypting key to a standard data decrypt call, which would result in the clear PIN block being returned. This is an attack on the correctness of the transaction set and demonstrates the necessity to 'separate' PIN encrypting keys from data decrypting keys. There are two common methods for

achieving this. The first is to use different master keys for encrypting different types of keys. Incorrectly using an encrypted key would result in the key being decrypted under the incorrect master key, yielding a 'random' result. This may be detected if parity checking of keys is enforced or enabled. The second method, involves the creation of a variant of the master keys based on the type of key. Perhaps the most significant system is IBM's control vector method. A unique control vector is associated with each type of key. To encrypt or decrypt the key, the master key variant is created by exclusive-oring the master key with the control vector. Again, using an encrypted key as incorrect type, results in a 'random' result.

3.5. PIN Recovery Attacks

In this section we present a family of six new API attacks leading to PIN recovery. We describe these attacks in the context of the standard financial API. Thus we can apply them immediately to our reference APIs.

3.5.1. The Attack Model

We have the following inputs:

- *Query access* to a (potentially tamper proof) device with the typical PIN transaction set (our standard financial API as described above)
- An encrypted pin encrypting key which is valid for the device above (i.e., the PIN encrypting key is itself encrypted under a (master) key resident in the device)
- A valid encrypted PIN block (EPB), which was encrypted under the encrypted PIN encrypting key

It is our goal to find superior techniques to those listed under `Known Attacks', which exploit potential lack of correctness of the typical PIN transaction sets.

3.5.2. Our Results

We present 6 distinct attacks against the standard financial API functions that lead to the recovery of the PIN from an encrypted PIN block. These are

- the *ANSI X9.8 (ISO-0) attack* against functions to which the PAN is supplied,
- the *extended ANSI X9.8 attack* against translate and reformat functions,
- the *decimalization attack* against PIN verification algorithms using offsets,
- the *key separation attack #1* against PIN verification functions based on failure to enforce key separation between verification and translation (encryption),

- the *key separation attack #2* against PIN verification functions based on failure to enforce key separation for different verification algorithms, and
- the *check value attack* against PIN verification algorithms using the check value of a key.

The attacks are computationally trivial and extremely fast to implement with the capability of recovering many PINs per second, depending on the speed of the target device. Since the vulnerabilities identified by these attacks can be exploited in a number of similar variations, it can be viewed as families of attacks.

3.5.3. ANSI X9.8 (ISO-0) Attack

The significant input parameters for this attack are the

- Encrypted PIN Block (EPB)
- PAN
- Encrypted 'In' Key
- Encrypted 'Out' Key

Our attack strategy is as follows. In an iterative manner, we make a single modification to the PAN and observe the effects. Under normal operation, the encrypted PIN lock (EPB) will be decrypted and combined with the PAN to recover the PIN as per the following process:

INPUTS (EPB, P2)

1. PB = d_k(EPB)
2. P1 = PB ⊕ P2
   = 04PPPPFFFFFFFFFF
3. Extract PIN as PPPP and test to confirm it is a valid PIN (i.e. each P is a valid decimal digit)
4. If test fails then exit with an error
5. Process the rest of the function as normal

Algorithm 3: Normal PIN extraction process

Since the PIN we are attacking is valid by definition, the function will complete normally. However, instead of supplying the correct PAN (effectively P2) to a call, we use a modified PAN (P2' = P2 ⊕ Δ) where say Δ = 0000x000000000. Now observe the same process:

INPUTS (EPB, P2')

1. PB = d_k(EPB)
2. P1' = PB ⊕ P2'
   = (P1 ⊕ P2) ⊕ (P2 ⊕ Δ)
   = P1 ⊕ Δ
   = 04PPPPFFFFFFFFFF ⊕ 0000x00000000000
3. Extract PIN as PPPP ⊕ 00x0 and test to confirm it is a valid PIN

```
4. If test fails then exit with an error
5. Process the rest of the function as normal
```

Algorithm 4: Basic ANSI X9.8 attack method

So if (P ⊕ x) is a decimal digit then the call will proceed normally and pass. If (P ⊕ x) is not a decimal digit then we expect the call to fail and return an error code. Thus we have a test for (P ⊕ x) < 10.

Our next step is to turn this test into an algorithm to identify P. The simple approach is to try all possible values of x, yielding a semi-unique pattern of 'passes' and 'fails', allowing us to identify P. We observe that for $y \in Z_{16}$, $y < 10$ if and only if $y \oplus 1 < 10$. Thus, for any given number $x \in Z_{16}$, both x and $x \oplus 1$ will result in an identical pattern of 'passes' and 'fails' and, furthermore, no other values of x will yield the same pattern. A more sophisticated approach is to build a decision tree.

This attack will work against any function that accepts the encrypted PIN block as well as the PAN and (attempts to) extracts the PIN. Thus it can be applied against both the verification, translation and reformatting functions.

3.5.3.1. Work Factor

The simplest (and least efficient) algorithm requires 16 calls per PIN digit for a total of $\sum_{i=1}^{n-2} 16 = 16 \cdot (n-2)$ to recover the last $n-2$ PIN digits.

3.5.4. Extended ANSI X9.8 Attack

The previous attack had two limitations, namely:

1. it identified each PIN digit as P as belonging to the set $\{P, P \oplus 1\}$, and
2. did not recover the first 2 PIN digits.

We now consider methods to extend this attack.

The attack mechanism involved manipulating (i.e. lying) about the value of the PAN associated with the encrypted PIN block. The logical extension is to ask the question whether there are other parameters to these functions that can be manipulated? The answer is yes - the PIN block format. Consider the effect of submitting an ANSI X9.8 formatted PIN to the reformat function *but specifying*

it as a PIN block of another format (e.g. a VISA-3 format). To simplify the analysis, we assume a NULL PAN (i.e. PAN = 000000000000).

INPUTS (EPB, P2, Format')

1. The clear PIN block is obtained by decrypting the encrypted PIN block (i.e. PB = d_k(EPB)). Since the clear PIN block is actually an ANSI X9.8 format, hence
   PB = P1 ⊕ P2
   = 0LPPPPFFFFFFFFFF ⊕ 0000000000000000
   = 0LPPPPFFFFFFFFFF
2. The PIN is extracted according to the VISA-3 formatting rules (i.e. PIN is extracted as 0LPPPP.
3. The PIN is then formatted into a new PIN block as
   P1 = 0L'0LPPPPFFFFFFFF (where L' = L+2)
   P2 = 00000000000000000
   PB = P1 ⊕ P2
   = 0L'0LPPPPFFFFFFFF (where L' = L+2)
4. The encrypted output is EPB = e_k(PB))

Algorithm 5: Increasing PIN length

In this process we have extended the original PIN to a new value 0LPPPP. By doing this, we have now exposed the first two PIN digits to the previous ANSI X9.8 attack.

Consider now the effect of using a non-null PAN (Δ) at the same time. The clear PIN block is obtained by decrypting the encrypted PIN block (i.e. PB = d_k(EPB)). Since the clear PIN block is actually an ANSI X9.8 format,

PB = P1 ⊕ P2
= P1 ⊕ Δ (say Δ = 00000x0000000000)
= 0LPPPPFFFFFFFFFF ⊕ 00000x000000000000

The PIN is extracted according to the VISA-3 formatting rules, which means we expect the hexadecimal digit 'F' to mark the end of the PIN. There are 3 possible outcomes:

If P ⊕ x < 10 (i.e. a decimal digit) then the PIN will be extracted as the six digit PIN 0LPPPP ⊕ 00000x. The call will proceed normally.

If (P ⊕ x) = F then the PIN will be extracted as the 5 digit PIN 0LPPP. The call will proceed normally.

However 10 ≤ (P ⊕ x) < 'F' is interpreted as a format error resulting in the call failing.

This method allows us to identify uniquely the value of the PIN digits.

3.5.4.1. Work Factor

Again, the simplest (and least efficient) algorithm requires 16 calls per PIN digit for a total of $\sum_{i=1}^{n} 16 = 16 \cdot n$ queries plus the initial call to expose the first two PIN digits. A typical 4 digit PIN is thus recovered in $16 \cdot 4 = 64$ queries.

3.5.5. Decimalization Attack

The significant input parameters to the decimalization attack are the

- Encrypted PIN Block (EPB)
- Validation Data
- Decimalization Table
- Offset
- Encrypted PIN Encrypting Key
- Encrypted PIN Verification Key

Our attack strategy is as follows. In an iterative manner, we make a single change to an entry in the decimalization table and observe the effects on the offset. We first demonstrate this by means of an example.

Example.
We start off with a PIN (6598), a PIN verification key, some validation data, a decimalization table and the resulting offset (2117).

```
PIN           = 6598
PIN Ver Key   = 05050505 05050505
Val. Data     = 11223344 55667788
Ciphertext    = E481FC56 58391418
Dec. Table    = 01234567 89012345
IPIN          = 4481
Offset        = 2117
```

We now make a single change to the decimalization table replacing the first entry (which previously mapped a 0 in the ciphertext to a 0) with a new value of 1 (i.e. now maps a 0 in the ciphertext to a 1).

```
Dec. Table (0)= 11234567 89012345
```

Since the ciphertext does not contain a 0 in the first 4 nibbles, there is no effect on the value of IPIN and so the original offset will continue to pass.

```
IPIN          = 4481
```

```
Offset          = 2117 (will pass)
```

Next we replace the second entry in the decimalization table with a 2 (i.e. it now maps a 1 in the ciphertext to a 2).

```
Dec. Table (1)= 02234567 89012345
```

Since the ciphertext does contain a 1 in the 4th nibble, this results in a change to the IPIN value.

```
IPIN            = 4482
```

This time the original offset of 2117 value will fail. However the value 2116 will pass.

```
Offset          = 2117 (will fail)
                = 2116 (will pass)
```

Thus we have identified that the 4th digit in the original IPIN is a 1 and hence that the 4th PIN digit is 1+7 = 8 (IPIN + Offset).

This process can be explained more formally as follows. Since the user supplies the decimalization table to the call, one can repeatedly query the target with modifications to the table. Suppose we know the offset for a given PIN block (using a given decimalization table). Consider the effect of changing a single element in the table (the k^{th} entry in the table mapping $\{i \rightarrow j_i, i \in Z_{16}, j_i \in Z_{10}\}$ to a new value j_k')

If the hexadecimal digit k was not found in the first n digits of ciphertext, then there is no change in the value of IPIN, and the same value of the offset will pass the verify call. However, for each instance of k in the ciphertext, the corresponding digit of IPIN will be remapped to j_k'. The original offset will now fail the verify call. Using this approach we can identify the possible values of the hexadecimal digits in the first n digits of the ciphertext.

This technique can be further strengthen by setting $j_k' = j_k + m$ where m is a known (non zero) value (addition is modulo 10). Due to the simple relationship between the offset and the IPIN, we know that by adding m to a digit in the IPIN, the corresponding digit in the offset is reduced by m. Thus we can search through all possible ciphertext digit locations that contain k, by modifying the offset value and supplying the modified offset and decimalization table to the verify function. After at most 2^n queries, we will have identified all digit locations in the ciphertext with the value i. By repeating through all possible values of k, we can uniquely determine the value of the first n digits of ciphertext and thus IPIN. Again since we know the offset and IPIN, we can trivially calculate the value of the PIN.

INPUTS: DEC. TABLE

1. Search for offset (using the decimalization table i → i mod 10)
2. For k = 0..15
   2.1. Replace the entry k → k mod 10 in the original decimalization table with k → k + m mod 10
   2.2. For each possible location of k in the ciphertext (including none).
      2.2.1. Test the corresponding modified offset.
      2.2.2. If 'pass', then store locations of k in ciphertext.
3. Decimalise the 'recovered' ciphertext and store as IPIN
4. Return PIN = IPIN + OFFSET

Algorithm 6: Decimalization Attack

3.5.5.1. Work Factor

The initial search for (an unknown) n digit offset requires at most 10^n queries (although it may be possible to reduce this to $10^4 + (n-4)\bullet 10$ queries using the method described in Section 3.5.5.2). Each change to the decimalization table requires at most 2^n queries (again it may be possible to reduce this to $2^4 + (n-4)$ queries). At most we need to try 15 of the 16 entries in the table for a maximum total of $15\bullet 2^n$ queries. The actual attack time is dependant on the speed of harvesting the data. An attack against a 4 digit PIN using a known initial offset will require at most 240 queries or less than 0.25 seconds for a device capable of 1000 transactions per second (TPS). A low-end device (10 TPS) will take at most 24 seconds.

3.5.5.2. Improving the Search for Offsets

Some APIs allow the caller to specify the length of the offset (i.e., the length of the offset is treated as independent of the length of the PIN). The only restriction is that the offset is at least as long as the minimum possible length of the PIN, traditionally 4 digits, and does not exceed the length of the PIN. In this case, an attacker can initially search for a 4-digit offset (requiring 10^4 queries). Thereafter, each subsequent digit can be searched independently by specifying incrementally increasing offset lengths. This search method requires a total of $10^4 + (n-4)\bullet 10$ queries to recover a n -digit offset.

3.5.6. Key Separation Attack #1 (Failure to separate PIN encryption and verification keys)

We describe an exhaustive PIN search attack that exploits a lack of separation between PINGEN/PINVER and IPINENC/OPINENC keys to perform the search. Since we have no separation, we may interchange the use of the pin encrypting key and the pin verification key. We

wish to recover the PIN (P), encrypted under some key k. We have the associated encrypted PIN block EPB and the encrypted key k.

Our first step, is to translate the EPB to ANSIX9.8 with PAN = 000000088888. The clear PIN block is of the form PB = $04P_1P_2P_3P_4FFFFF77777$. Let $P^i = P^i_1P^i_2P^i_3P^i_4$ be a guess for P (for simplicity of notation, we have used a 4 digit pin). We format a 16 hexadecimal validation data string as VAL = $04 P^i_1P^i_2P^i_3P^i_4FFFFF77777$. We now use this validation data as input to say the IBM Pin Generation algorithm using the encrypted key k. The validation string is encrypted under the clear value of k, decimalised and the first 4 digits returned as the generated PIN (P_{gen}). We simply compare the first 4 decimalised digits of our translated encrypted PIN block to P_{gen}. If $P^i = P$, then the values will be equal. We can expect multiple collisions due to the decimalization process and the fact that we only have 4 digits to compare. However, we can eliminate false witnesses by repeating the process and modifying the validation data and the PAN (e.g. use PAN = 000000088887 and VAL = $04P_1P_2P_3P_4FFFFF77778$). Simply by iterating through all possible values of P^i we can identify P. Alternatively, we can build up a codebook.

One may question why we didn't use a null PAN, and format the validation data as VAL = $04P_1P_2P_3P_4FFFFFFFFF$. The reason lies in the fact that some API's take as input, effectively an 11 hexadecimal digit validation string, and extract the last 5 digits from the rightmost 5 digits of the PAN. Other API's require the 12^{th} digit to specify a key index in the range 1 to 6. The description above will satisfy the addition requirement that the PANs supplied to generate and translate functions consists of decimal digits (for the above description, the generate call uses a PAN = 000000077777).

A limitation of this attack, is that it requires access to the generate function. However, as mentioned before, this is a security risk in itself, usually protected against by restricted its usage to a secure or authorized mode, or alternatively, encrypting the output (or requiring encrypted inputs). If the output is encrypted, then we must obtain the first 4 decimalised digits of the encrypted PIN block as an encrypted PIN for comparison. This in itself is perhaps not a strenuous requirement. However, there is a more powerful version of the attack, using the verify function (which is unlikely to have access control restriction). We proceed as before, generating the validation as before, but supplying it to the verification function (using offsets) and supplying the encrypted pin block. We require one extra piece of information, namely the offset. If $P^i = P$, the validation data is identical to the clear pin block as before, the intermediate PIN will equal the first 4 decimalised digits of the encrypted PIN block (which is known). We denote this value by IPIN.

The offset is the difference of the intermediate PIN and the clear PIN modulo 10. Hence the offset will equal will equal the difference of P^i and IPIN modulo 10. Thus for each value P^i we calculate OFFSET = P^i - IPIN and supply it to the verification call. If P^i = P, the call will pass. False witnesses can be eliminated as before.

3.5.6.1. Work Factor

Essentially a method to search the PIN space, this attack therefore requires 10^n queries to identify an n digit PIN given knowledge of a suitable offset. Searching for the offset requires 10^n queries (although it may be possible to reduce this to 10^4 + (n-4)•10 queries using the method described in Section 3.5.5.2).

3.5.7. Key Separation Attack #2 (Competing Verification Algorithms)

In this attack, we play two different verification algorithms against each other. This is a fascinating result since it shows that taking two (potentially) individually secure verification functions, and allowing both in a single API can destroy its security. It serves as a warning for designers (and implementers) of API's against blindly adding functionality, even though the function may be secure on its own. It also shows the need for extremely fine granularity of separation of keys associated with different functions, even though the functions are of a similar nature.

The VISA PVV algorithm encrypts the concatenation of the 11 digit transformed security parameter (TSP), a key index (a digit in the range 1 to 6) and the first 4 digits of the PIN extracted from the encrypted PIN block (i.e. TSP || Key Index || PIN). The result is decimalised in a unique way. Scanning from left to right, the first 4 decimal digits encountered are returned as the PVV. Should there be less than 4 digits, a second scan is performed, in which the non decimal digits are converted to decimal digits (by subtraction modulo 10), and the first 4 resulting digits returned as the PVV.

The probability that the first 4 digits are indeed all decimal digits (and hence form the PVV) is $(\left(\frac{10}{16}\right)^4 = 0.153)$. Thus for a given key, by trying 7 different values for the TSP, we can expect one to exhibit this property.

Consider now the result of supplying the value (TSP || Key Index || PIN) as validation data to the (IBM) offset algorithm under the same verification key and using a decimalization table that maps n $\rightarrow$ n mod 10. The first 4 digits recovered from the encryption are the same as the PVV, and

unchanged by the IBM decimalization process (i.e. IPIN = PVV). These 4 digits are subtracted from the clear value of the PIN to obtain the offset (or OFFSET = PIN - PVV).

It is possible to use this relationship to search for the PIN, by iterating through all possible value for the PIN (denoted P^i), and testing whether $OFFSET^i = P^i$ - PVV, satisfies the verification algorithm with validation data (TSP || Key Index || P^i). As indicated earlier, the entire process needs to be repeated 7 times on average to witness the true value for the PIN. We can expect some false witnesses for the PIN but these can be eliminated by increasing the number of repetitions (say to 14 when we can expect the real PIN to have been witnessed twice).

INPUTS: Encrypted PIN Block

1. Loop i: (i = 1..7)
   1.1. TSP^i <= i || 1 || 1234
   1.2. Search for PVV^i
   1.3. Loop j: (j = 0...9999)
      1.3.1. PIN^j = j
      1.3.2. $OFFSET^{i,j} = PIN_j - PVV^i$
      1.3.3. VAL $DATA^{i,j}$ = i || 1 || j
      1.3.4. Test if $OFFSET^{i,j}$ verifies the PIN, if so then j is a candidate.

Algorithm 7: Key Separation Attack against competing verification algorithms

Example:

```
PIN          1234
Verification Key (K) 0505050505050505

i = 1:
TSP[1]           10000000000011234
E(K)(TSP[1])     D84355EF089EF293
PVV[1]           8435
Will not correctly witness the PIN.

i = 2:
TSP[2]           2000000000011234
E(K)(TSP[2])     9D15A1C0BE1DD129
PVV[2]           9151
Will not correctly witness the PIN.

i = 3:
PVV[2]           3000000000011234
E(K)(TSP[3])     6787FA69080E3C93
PVV[3]           6787
Will correctly witness the PIN.
```

```
  j = 1
  VAL_DATA[3,1]   = 3000000000110001
  OFFSET[3,1]     = 4324
  PIN_verification call returned FALSE

  ...

  j = 1234
  VAL_DATA[3,1]   = 300000000011234
  OFFSET[3,1]     = 5557
  PIN verification call returned TRUE.  Store j = 1234 as candidate for the
PIN.

  ...
```

Changing the decimalization table and calling the offset verify function again can eliminate some false witnesses (the digits 0 to 9 must remain mapped to themselves). For example, let $n \rightarrow n$ ($n < 10$), else $n \rightarrow n+1 \bmod 10$ ($n \geq 10$).

3.5.7.1. Work Factor

The inner loop of the algorithm requires 10000 calls to the offset verify function plus the initial search for the PVV, which requires a further 5000 calls on average (10000 maximum). The outer loop needs to run 7 times on average to witness the correct value of the PIN. In practice, we run this loop 14 times to eliminate false witnesses for a maximum possible total of $14 \cdot (10000 + 10000) = 280000$ queries.

3.5.8. Check Value Attack

The check value function encrypts a 64 bit binary zero under the supplied key. The PIN verify function (for IBM and GBP algorithms) encrypts a 64 bit user supplied validation data. The resulting ciphertext is decimalised via a user supplied decimalization table and termed the intermediate PIN (IPIN). The offset is the result of subtracting the IPIN from the customer selected PIN modulo 10 (OFFSET = PIN - IPIN modulo 10). The key observation is the similarity in operation between the check value function and the verification function.

Consider the case when the validation data is a 64 bit binary zero. The result of the encryption stage is the same as the result from the check value call (or more accurately, the first n bytes are the same for a n byte check value). The decimalization table is known, so it is trivial to calculate the IPIN value. It is also possible to recover the value of the offset by exhaustive search (again using the verify function). With knowledge of both the IPIN and offset, it is a trivial calculation to determine the PIN.

Algorithm:

1. Calculate the check value of key
2. Decimalise the check value and store as IPIN
3. Search for the OFFSET (with validation data = 000000000000000)
4. Return PIN = IPIN + OFFSET

Example

```
PIN                  1234
PIN ver key          0505050505050505

Validation data      0000000000000000
Decimalization Table 0123456789012345
OFFSET(searched)     3034

Check Value          8CA0964
IPIN                 8200 (= the check value decimalised)

Calculated PIN       1234 (= OFFSET + IPIN)
```

Some observations: The attack (as always) accepts an encrypted PIN block and its associated (encrypted) PIN encrypting key. The actual value of the PIN verification key is not important and hence one can merely conjure the key (if possible) or use any other PIN verification key in the system.

3.5.8.1. Work Factor

The attack is fairly efficient, requiring only a single call to the key test function (to obtain the check value) and a single search for the offset. For an n digit PIN this requires at most 10^n queries (although it may be possible to reduce this to $10^4 + (n-4) \bullet 10$ queries using the method described in Section 3.5.5.2).

3.6. Analysis of the Commercial APIs

3.7. Real World Implications.

The question that begs asking is whether these attacks are a credible real world threat. The first comment that should be made (and typically the first response from a vendor susceptible to a newly published vulnerability [IBM01]) is that real world systems should be following standard industry best practices that, if implemented correctly and enforced, should limit a potential hacker's ability to perform such attacks. This is typically taken to mean the use of physical access control and operation

procedures to restrict access to the security devices. This defence is correct in that an attacker does require some form of query access to the device (whether it be physical or remote electronic access) in order to effect the attack.

However there are some observations and counter arguments that question the validity of this defence. The first is that an attacker can attack at weakest point. By the nature of the financial systems' architecture, this means that one institution's account holder can be compromised on another institution's network (i.e., as the encrypted PIN travels through the network belonging to a different institution, it may be attacked). Hence one must guarantee that all potential networks through which the PIN may travel are secure. Can one realistically expect, claim or prove that all possible institutions in all cases correctly implemented and observed the required best practices? And how would one justify buying an expensive tamper ***resistant*** security module in the first place if one's defence rests on physical access control?

In addition, these devices now find use in other environments such as multi-lane retail stores where they are used as concentrators for the EFTPOS devices in the store. While it may be reasonable to expect a financial institution to have the necessary rigorous controls in place, it may not be necessarily be the case in a high volume, public store.

Finally, there is an implied assumption as to the impenetrability of the network in discounting the possibility of a remote attack. A traditional hacker gaining access to host with a security device could easily implement the attack.

3.8. Solutions

So what went wrong? Firstly, it is clear that a number of functions were just badly thought out and are insecure. The ease with which decimalization tables can be manipulated is perhaps a good example. Secondly, individually secure functions were added to the API in a manner to make the entire system insecure. The use of check digits function to compromise the verification algorithm is a classic example. Insufficient attention was given to the possible interplay between functions such as the key separation with competing verification algorithms. Finally, the absence of a single standard, to which everyone completely adheres, contributed to the complexity of the system, creating opportunities to exploit (e.g. the many different formats and algorithms that exist due to historical reasons). Saying this, it is difficult for a vendor to restrict functionality when different customers want different functionality from the same product.

Our proposed solutions are the following.

1. **Remove 'weaker' algorithms and functions and adhere to a single standard**

   When there is a choice between algorithms that effectively provide the same functionality, the weaker one should be removed. By leaving only the strongest function the level of security has immediately been increased. This approach complements the goal of supporting only a single standard to the exclusion of everything else. It reduces the possibility to interaction between algorithms.

2. **Parameter (data) Integrity**

   Many of the attacks exploit the possibility of modifying input parameters to the functions (including, amongst others, the PAN, the PIN block format, decimalization tables, validation data, transformed security parameter (TSP), etc). Encrypting the parameters offers a possible solution. However perhaps the most logical solution is to include the use of MACs (or similar methods) to ensure data integrity. For example, a decimalization table is supplied to a function call along with a MAC and the key for verifying that MAC. It may be natural to expand the PIN block, adding fields to include the PAN and PIN block format and a MAC over the entire PIN block (or encrypted hash). A clever solution to enforcing the correct PIN block format is the use of an additional control vector that is unique per PIN block format. This is exclusive-ored to the key before use. This method known as PIN block variance control was found in an earlier IBM API known as ICSF. The advent of AES may provide an option for extending the PIN block owing to its increase block size (128 bits instead of 64). As a result, a PIN block of the form Length||PIN||PAD||PAN||Format could be used, eliminating many of the problems.

   These solutions are fairly obvious. The challenge comes in ensuring interoperability between devices from different manufacturers.

3. **Key Separation**

   Key separation is of vital importance. As mentioned before, it is a well-understood principle. Failure is typically either as a result of a design or implementation oversight or alternatively as a lack of understanding of the extent of separation required.

While implementation oversights are potentially inevitable (and not limited to key separation issues) they can be addressed through training and peer review. A structured and complete approach is required to identify possible areas of compromise in design where there exists the possibility of compromising interactions.

Similar functionality (such as slightly different PIN verification algorithms) is often naively grouped together, for example as parameterised functions or supporting the same key types. While this may seem intuitively acceptable, in our analysis we have shown this be a source of separation issues. Separation must be extremely fine grained down to the individual algorithm level and the design must resist the temptation to categorize on a more general level. In the case of PIN verification, this means that keys must be identified and restricted for use with only a single verification algorithm as opposed to the current common practice of being useable with a number of algorithms. Likewise a distinction between PIN encrypting and PIN verification keys must be enforced.

The use of PIN block variance (essentially a key separation mechanism that relates PIN block format to the key) is a useful method offering the fine-grained restrictions required of an API.

4. Electronic access control

The benefit of electronic access control cannot be overstated. It should be fine grained, allowing the individual enablement/disablement of:

- formats,
- algorithms, and
- functions.

Whenever working with access control one must observe the mantra of permitting only the barest minimum required functionality. Start by disabling everything. Then enable only what is required.

Access control has the exceptionally useful property of being able to disable (or remove) a function that has been shown to be insecure. As such it provides an immediate defensive response by removing vulnerability in the device. It can also be effective in enforcing dual control.

5. Implementation

As has already been alluded to, a number of issues can arise as a result on implementation errors and oversights. Building a system that is expected to withstand a high level of malicious or 'out of specification' operation by malicious users, is a complex and challenging task and famously referred to as 'Programming Satan's computer' [AN95]. An engineering or development team working on such an assignment requires an emphasis on security priorities and quality control which at the very minimum requires specialized training and a submission to peer review.

3.9. Hackers and Threats: Real World Scenarios

We diverge briefly from the theoretical analysis of API vulnerabilities to explore the exploitation thereof and the impact of such failures in the real world. Thus we present a series of scenarios in which we leverage our knowledge to attack financial institutions and their account holders.

3.9.1. Insider Attack

The insider attack is the first and obvious attack arising from the weaknesses in the standard financial API. It presupposes the existence of an 'insider' who is either a person internal to a given financial institution (e.g. employee, contractor, cleaning staff, etc) or else an individual who has gained access to the financial network, perhaps through some traditional network hacking technique. This attacker either has knowledge of the attack methods described or else has downloaded exploit code available on the Internet. He begins by monitoring the transaction stream for a period, recording the encrypted PIN blocks and associated account numbers that pass through the system. Using the exploit code, the clear PIN numbers are extracted through a sequence of queries against the hardware security module. Having obtained a list of account numbers (and other information) along with the associated PIN numbers, the attacker surreptitiously leaves the scene of the attack.

To exploit this list, the attacker purchases a set of 'white cards' (blank cards without any silkscreen logo) and a card reader/writer (available as a combined package on the internet for $595 [HH]). He then writes the account information for each harvested account onto a card, creating in effect a duplicate card of the original card still in the customer's possession. He distributes these cards to a network of accomplices (perhaps a gang of street kids) and encourages them to perform a random tour of ATMs in the area. The instruction is that each card be used once a day, at a random ATM and used to withdraw the estimated daily withdrawal limit.

Let n be the number of compromised accounts, p the average period before unnatural transaction behaviour is noticed and l the daily withdrawal limit. The total fraud value is $F = n \cdot p \cdot l$.

Example:
$n = 5000$
$p = 2$
$l = \$1000$
Total Fraud = $ 10,000,000

Harvesting a large number of accounts (like the 5000 in our example) is trivial due to the efficiency of the attacks. A Trojan or piece of software could harvest many millions of accounts over just a few days – limited only by the transaction throughput. Log files could potentially provide an immediate and large source of encrypted PIN blocks.

There exist other possible methods to extract monies from the compromised accounts such as the use of Internet banking or account transfers. However, with the exception of bill numbers on the currency, there is limited ability to trace the money after the attack. Regardless of the method employed, a wise adversary would immediately launder the money.

3.9.2. Account Holder Attack

While the insider attack was the work of a third party that defrauded an account holder, this attack provides a mechanism for the account holder to defraud the institution. The malicious account holder first produces a number of duplicate 'white cards' of his own card. These he distributes to multiple accomplices, preferably in different geographical locations and instructs them to perform a random tour of the ATMs in the region. Upon receiving the first bank statement with these transactions on them, he immediately reports the 'unauthorized' activity on his account and disputes the transactions.

A first point is that it may be advisable for the malicious account holder to perform a valid transaction "simultaneously" with the fraudulent ones since this 'proves' that the account holder is in possession of his card at the same time as the fraudulent transaction occurred and preferably was in a different physical location. This provides fairly significant circumstantial evidence that the account holder was not involved in the fraud.

Extending this further, it is best done by multiple cardholders from a given institution since

1. it shows that it was not an isolated incident,
2. hence raises questions the security of the institution and
3. gives the impression of a possible insider attack.

Again let n be the number of compromised accounts, p the average period before unnatural transaction behaviour is noticed and l the daily withdrawal limit. The total fraud value is $F = n \cdot p \cdot l$ and the average return per individual is $R = p \cdot l$

Example:

$n = 100$

$p = 10$

$l = \$1000$

Total Fraud = $ 1,000,000

Average return = $ 10,000

3.9.3. The Repudiation Attack

The repudiation attack is an extension of the account holder attack. Here the malicious account holder simple denies a transaction. This will result in the bank's normal dispute resolution procedure being followed leading to possible litigation. The attacker will disown the transaction and claim that it must be an error on the bank's part or deliberate fraud and argue the insecurity of the system. The approach is best if the security of the institution has already been questioned.

For example, following a successful account holder or insider attack being made public – other account holders (acting individually) may dispute valid transactions that occurred during (or after) the attack period. The financial risk is great due to the possible scale (e.g. 0.1 % of an institution's 1,000,000 customers each disputing a $1000 transaction would result in a combined value of $1,000,000).

Perhaps more significant would be the loss of confidence in the given institution which could well be more damaging than any dollar loss. This leads us to some other approaches to benefit from these attacks.

3.9.4. The Competitor Attack

As mentioned earlier, an account holder is able to use his or her card on networks other than the account issuing bank's network. As a result, an account holder can be attacked on any network through which his PIN travels. This includes competitors' networks.

Thus a competitor could choose to mount an attack against account holders of a particular institution. In addition, as administrator of their own network, is it possible for them to use the additional benefits and powers that are associated with administrator privileges to effect this. Combined with unlimited time and access, the success of the attack is guaranteed.

However, the reward is not the stolen money but rather the 'after effects'. The negative publicity and damage to customer relationships that would arise after such an attack could destroy a bank's credibility and customer base. This resulting fall out would benefit the competitor. This has the advantage that there is less of a connection between accomplices and the attacker (in this case the competitor institution). There is no incriminating cash trail leading back to the attacker and no need for laundering. In addition, the effective monetary value of the reward could be considerable.

3.9.5. The Stock Market Attack

In another variation that eliminates the cash trail and potentially increases the value of the reward, the attacker sells the stock 'short' prior to any attack (a stock market strategy that pays off should the stock price fall). The negative after effects could contribute to a dramatic decrease in stock price.

3.9.6. The Anarchist Attack

With the goal of creating confusion and disorder in the world's financial institutions, an anarchist would employ all and any combinations of all the previous attacks.

3.10. Looking Ahead

Question: How can a bank guard against such attacks?

Answer: The bank should:

6. Contact its vendor, request any best practices information and implement it.
7. Be vigilant. Increase its auditing.
8. Reassure its clients.

9. Wait.
10. Apply positive pressure on the role players.

Question :Is that all? Why can the bank not just solve the problem?
Answer:
The nature of the problem is such that it is not a single bank's alone (unless it disconnects from the network). The entire network must be secured and until that happens the bank and its account holders are potentially vulnerable.

3.10.1. The Road Ahead?

It is envisaged that the process will be driven by the card associations (i.e., VISA, MasterCard, etc). This is due to their role and influence over the infrastructure. There should be a revision of the associated standards involving new design and security requirements. These requirements could be very prescriptive, limiting what functionality is allowed. Following a finalization of the standards, the vendors will then update products. Hopefully this will provide an opportunity for more uniformity and collaboration between different vendor product offerings. (This would makes business sense for institutions, given them more flexibility, choice and bargaining power). Finally, the card associations will mandate the new requirements to member institutions.

3.11. The Unanswered Question

Who is liable in the event of such an attack leading to fraud?

4. Designing for Failure: Can you survive your security mechanisms?

Financial institutions have traditionally presented automatic Teller Machine (ATM) networks as a stable, mature and trusted technology. However, in practice, such networks cannot differentiate between a transaction where funds have been drawn from the account of an innocent victim by a fraudulent third party, and a transaction where a fraudulent third party withdraws funds from its own account and poses as an innocent victim. Where an institution is faced with purported unauthorized withdrawal of funds from an account, and where no independent evidence exists to explain the withdrawal, institutions tend to debit the account holder with the loss. This tends to happen particularly in the case of multiple transactions.

The account holder is thus faced with the expense of instituting legal proceedings for recovery and the prospect of being unsuccessful in such proceedings. The institution may also instigate criminal proceedings against the account holder or his family or associate, as a criminal conviction substantially reduces the account holder's prospects of being successful in a civil action. Institutions tend to act ruthlessly in these situations, as they perceive an unfavourable court decision may open up a floodgate of fraudulent claims against them.

In this chapter we investigate the failure of security mechanisms in this regard. We discuss the concept of 'design for failure' (DFF) as a design philosophy and an extension to common risk analysis methods. Finally we propose some DFF extensions for existing PIN and card based banking systems that increase the tolerance for failure.

4.1. Introduction

We begin with a tale of two account holders – Dubya the innocent and Dubya the evildoer.

Dubya the innocent is beyond reproach - a morally upstanding person endowed with the finest of characters and a healthy respect for the law and doing what is right. He is also an account holder at the local bank, has a bankcard and the associated PIN, both of which he meticulously protects. However, through no fault of his own and perhaps due to a security weakness at the bank, a third party, Malice, gains knowledge of Dubya's card details and PIN. This information allows Malice to

produce a phoney duplicate of Dubya's card (called a white card). Malice embarks on a whistle stop 24hour tour (over the weekend) of the ATMs in an area, and at each one, he withdraws as much money as the card allows. In total, Malice milks the account of almost $80,000 after near 200 withdrawals. On Monday morning, Dubya the innocent gets a phone call to please report to his local bank with his bank card in hand to explain how $80,0000 was withdrawn from his account over the weekend.

Dubya the evildoer has a flexible moral code and is an account holder at his local bank for which he has a card and the associated PIN. Recently, Dubya the evildoer has learnt that there exists a security weakness in his local bank system, making it possible for a third party to gain knowledge of both his card details and PIN. Dubya creatively hatches the following plot. He jumps on an airplane and flies to another country. Once off the plane, he embarks on a random whistle stop 24-hour tour of the local ATMs, and at each one, he withdraws as much money as the card allows. In total, Dubya the evildoer milks the account of almost $80,000 after near 200 withdrawals. He rushes back to the airport, jumps on a return flight home arriving in time to receive the Monday morning phone call requesting that he report to his local bank with card in hand to explain how $80,0000 was withdrawn from his account over the weekend.

It is unfortunately the case that it is **not** possible to distinguish between Dubya the evildoer and Dubya the innocent!

The financial forensic evidence (the transaction data) is identical. In both cases, the system has logged that a magnetic stripe card contain Dubya's details was presented along with the correct PIN number associated with the account. This was sufficient to authorize the transaction.

So what happens now?

Dubya the innocent is understandably distressed that such a significant amount of money has disappeared from his account without his authorization (and without his doing anything wrong). He demands his money back and initiates a dispute procedure.

If it had been a single disputed transaction, the bank may have chosen to adhere to typical banking counsels ruling to refund the disputed transaction. However, the multiple transactions clearly indicate

fraud in this particular case! They consult the security expert, Eric, who can say one of three things, namely, that

1. the bank's system is secure, or that
2. there exists a weakness that may well have been exploited, which could explain this activity, or finally that
3. there exists a weakness in the bank's system, but he believes that it is either impossible or unlikely to have been exploited.

If the first option is advanced, then the conclusion will be reached that the account holder must be a fraudster. The situation will in all likelihood develop into a legal dispute where (despite Dubya's protestations of innocence) Eric will present 'evidence' and his 'expert opinion' that since the system is in fact secure, and that there is proof that the correct PIN number was supplied, it must be the case that Dubya (the only one with knowledge of the PIN) was responsible for the withdrawals.

If the second option were advanced, then one would expect the account holder not to be debited with the withdrawal.

If the third option is advanced, the outcome is less certain. There is a possibility that the institution would give the account holder the benefit of the doubt.

However, one has to consider the effect of the institution's conceding a flaw in the system. By refunding an innocent account holder, they lay themselves open to being exploited by a fraudulent third party. A concerted collective effort by a syndicate of evildoers could easily damage even the strongest of institutions. The initial weakness could thus lead to a catastrophic situation.

Accordingly one anticipates that an institution would probably continue to debit the account holder with the withdrawal. This forces the parties into legal proceedings. The institution may increase the pressure by laying criminal charges against the account holder, his associates or members of his family.

The situation quickly deteriorates into an ugly, unpleasant battle.

Two questions arise from this situation.

1. How does one design a system to prevent it from failing catastrophically upon the discovery of a weakness or flaw?

2. Can one define a realistic, achievable definition of security, which would avoid involving an innocent account holder in litigation and delivers a viable 'real world' solution?

4.2. Related Research

Perhaps the most well known research into the failure of ATM networks can be found in [An94a] and a treatment of the implications and liability in [An94c]. Anderson's book [An01] serves as a comprehensive introduction and text on the topic of security engineering and designing secure systems. Robustness is considered in [An92, An94b] within the context of a smart card payment mechanism. Unfortunately, the reluctance of financial institutions, card associations and product vendors to publish weakness, skews their contributions largely to the specification of various transaction and payment schemes and products. Recently, there have been significant contributions in the guise of API attacks [An00, AB01, Bo01, Cl01, CB02, BZ02, Cl02] against the infrastructure upon which ATM and payment networks are based.

Since the original research, a number of factors have impacted or continued to impact on this particular aspect of security engineering for the financial industries.

4.2.1. Pervasiveness of Existing, Known Vulnerabilities

Early work regarding insecurities of financial transaction systems [An94a] was focused on incident specific failures that were not necessarily generally applicable to other institutions. This case-by-case treatment of the subject matter still allowed proponents of the existing networks to argue the security of the system.

It is thus of great significance that the key token weakness described in [AB01, Bo01, CB02] is so universally applicable and has forced a change to the standards [ANSI02, AHTV02a, AHTV02b]. Furthermore, the PIN recovery attacks of [Cl01, BZ02, Cl02] are equally pervasive. We have yet to encounter a security module, designed for protecting transaction and ATM networks, that is not vulnerable to at least a subset of these attacks. With a high degree of confidence, we can thus claim the existence of weaknesses in any and all such transaction systems.

4.2.2. Change in nature of weaknesses

Historically, failures occurred typically as a result of implementation or operation errors and led to the observation in [An01] that "the threat model commonly used by cryptosystem designers was wrong: most frauds were not caused by cryptanalysis or other technical attacks, but by implementation errors and management failures". Currently, this probably remains true for incidents of fraud. However the vulnerabilities described in [Cl01, BZ02, Cl02] are predominantly design flaws. And while the existence of a weakness does not in itself guarantee fraud, it is conceivable that it could well be exploited in the future.

4.2.3. Connectivity and Interoperability

ATM networks display a high degree of connectivity and interoperability. We take this for granted when we use our bankcard at an ATM belonging to another bank, in a country other than our home country, perhaps on another continent. We expect to be able to transact successfully.

But as a result, we do not only rely on the security of our own bank (the issuing bank) but also on the security of the acquiring network and any intermediary networks and switches over and through which our 'secret' transaction authorizing details travel. Thus the historical claim by an institution of the security of its system can been seen as wholly inadequate and shortsighted.

4.2.4. Inability to Detect Fraud

While the existence of fraud arising from one of these new attacks may be detectable, it is not possible to verify the nature of the fraud. It remains indistinguishable from an account holder's fraudulently repudiating transactions.

4.2.5. Continued Legal Action

The number of legal cases involving 'phantom withdrawals' continues to increase. One is never certain whether this should be dismissed as expected or investigated as a possible indicator of innocent account holders being unfairly treated.

This motivates further investigation into the security of ATM and payment networks, including their ability to survive failure, and to re-visit some of the themes from the earlier work on the topic.

4.3. Design for Failure

Normally when we design, we focus on preventing failure. Design for failure (DFF) is used to describe the process whereby the implications of the possible failure of the various security mechanisms employed is considered with the goal of obtaining a failure tolerant, survivable system (i.e., we start by assuming the failure of our security mechanisms and design for survivability). One may think of it as a form of insurance for security designers[1]. The goal is that any potential future failure is detected, identified and contained. The damage is limited by the upfront design intent and remains beneath predefined levels of tolerance.

One can immediately realize the benefits of such a process in the dynamic and adversarial environment of security engineering. In no way does this limit or aim to detract from the primary goal of designing a 100% secure system. Rather it augments this, by providing a safety net.

We next consider the criteria for design for failure.

4.4. Principles of Design for Failure

4.4.1. Principle 1

Believe that it can be broken. Expect that it will.

This is a common risk avoidance strategy and indeed a justification for DFF. Consider the familiar example of backups of electronic data. The need to anticipate and prepare for failure is a particularly well-known and understood requirement. Organizations and individuals alike have formal and informal procedures concerning the process of backing up data and securely storing it – typically to

[1] This is not to be confused with another common design philosophy, that of 'design by failure'. Here, following the initial design, the system undergoes a repetitive re-design process as it gets repeatedly broken and then patched. This is particularly true of some ciphers and commercial software.

the extent of having multiple, secure, offsite copies. Such a system is designed to tolerate even catastrophic failure and limit the losses associated with such an event to an acceptable level.

Sometimes, in our focus of building an unbreakable solution, we lose sight of the possibility that our solution may one day succumb. This could be for a variety of reasons, from a flawed design to an implementation error. When such a system fails – it can fail spectacularly. Some brief case studies provide illustrative examples.

Case Study: Tamper Resistant Security Modules DES Key Tokens
The security modules of the three main vendors who, combined, probably account for close to 99% of the devices in the field are subject to key recovery attacks due to a design flaw involving key tokens. Almost every key used in existing financial networks could potentially be or has been recovered. This raises questions as to the authenticity of past, present and (at least until they fix the standards and products) future transaction. Similar results exist for techniques to recover PINs from the same networks. There is no capacity built into the system to tolerate failure.

Perhaps we fall victim to believing in our own infallibility (enforcing design procedures can combat human nature to an extent). Often we simply fail to recognize the inevitable advancement of the capabilities and knowledge of potential adversaries. As a result, the 'level of security' provided is a dynamic property that can degrade over time. One must remain aware that a secure system has the potential to transform to an insecure one over its life cycle.

A system can be described in terms of algorithms, protocols, architecture, implementation and procedures. The system's architecture is comprised of protocols, which in turn consists of various algorithms. An implementation is a practical instance of the architecture, while the procedures describe the process of interacting with the implementation.

It is common for the security analysis to focus predominately on the algorithms to the exclusion of the implementation or procedures. One sees it in the product documentation and expert witness testimony, revealed by statements such as "with 128 bit encryption, the product/system is unbreakable". Statements like this indicate contempt for the first tenet of design for failure as well as the second, which we now introduce.

4.4.2. Principle 2

Analyse everything. Identify the threats and design to prevent them. Consider the implications of the failure scenarios

The goal here is to identify every possible adversary and attack strategy and quantify the associated potential exposure. This then allows us to design our system to defend against these. Fortunately there exist several established risk analysis and management techniques, to aid us in this process, including threat trees and failure modes and effects analysis (FMEA). The application and the benefit thereof are well understood and are dealt with comprehensively in [Ba93].

As with risk analysis techniques, there exist many methods to quantify an exposure as an input to managing the risk it represents. For our purposes later, we will briefly describe the Annualised Loss Expectancy (ALE) approach, which multiplies the expected loss arising from an incident by the expected yearly number of occurrences of the incident. While often criticized as an analysis of guesstimates by 'experts' in the absence of real figures (particularly for the less common attacks), it does provide an intuitive and real world quantifiable measure for the system's exposure. A typical extract from the ALE for an ATM network is reflected in Table 1.

Table 1Extract of Annualised Loss Expectancy for ATM networks.

Loss Type	Amount	Incidence	ALE
HSM attack	$10,000,000	0.01	$100,000
ATM network attack	$5,000,000	0.05	$250,000
ATM fraud – third party	$50,000	0.2	$10,000
ATM fraud – repudiation	$20,000	0.5	$10,000
Teller takes cash	$3,240	200	$648,000

While FMEA and threat trees are risk analysis techniques which provide a valuable source of design requirements when analysing the security of a system, the application of these risk analysis techniques is unfortunately not necessarily sufficient for designing failure tolerant systems. Consider the fundamental methods behind these approaches. In sequence, the analysis identifies (hopefully) all imaginable failures, quantifies the effects of each failure (largely to justify the expense of building in a control to prevent the failure) and leads to a control to prevent or protect against the failure. Essentially, these methods are themselves *designed to protect against (an initial) failure*. They are not

extended to the next step, protecting *against the effects of failure,* should it occur, that is, designing *to make a failure tolerable*. It can be justifiably argued that designing for failure tolerance is a subset of designing against failure. In practice, this criterion is often overlooked.

Any procedural method that aims at increasing tolerance of failure (particularly in a security setting), should include at least the following points:

- Controls to identify the existence of failure (particular the two distinct cases of fraud and error),
- Methods to respond to failure,
- Controls to limit exposure,
- Controls to identify the nature of the failure and limit impact on other security mechanisms and the security and integrity of the system as a whole,
- Controls to protect other users.

The first 3 points are obvious, and the last two typically not given the consideration they warrant.

Consider, the impact of one attack on another. Some attacks, such as the theft of cash by a teller, have no impact on other attacks. However, an HSM (Key or PIN recovery) attack has dramatic implications for ATM repudiation attacks. The security mechanism preventing the repudiation attack is nullified by even the possible real world existence of the Key or PIN recovery attacks. The same is true should third party ATM attacks have been shown to be realistically achievable. This exposes the system to secondary attacks.

How does this affect the ALE? Once it becomes known that the security mechanisms do not offer a defence against the repudiation attack (or even there is debate as to the integrity of the security mechanism) one could expect two effects – an increase in incidence of repudiation attacks and an increase in the attack amount. Consequently, the ALE gets revised upwards.

Table 2Revised Annualised Loss Expectancies following knowledge of a public weakness in a security mechanism.

Loss Type	Amount	Incidence	ALE
HSM attack	$10,000,000	0.01	$100,000
ATM network attack	$5,000,000	0.05	$250,000
ATM fraud – third party	$50,000	0.2	$10,000

ATM fraud – repudiation	$25,000	**1000**	$25,000,000
Teller takes cash	$3,240	200	$648,000

This is precisely the case of the two Dubyas and explains why the financial institutions are so concerned about a legal precedent being set that exposes them to mass fraud. In a truly failure tolerant system, a weakness should not provide a mechanism for valid transactions to be disputed. Unfortunately, in reality, most systems do contain master secrets, which represent a single point of failure.

4.4.3. Principle 3.

Have the ability to identify fraud and error. Verify the transaction automatically, regularly, repeatedly and in real-time. Seek confirmation. Respond to limit fraud.

It seems sensible and obvious that this principle should be implemented. Unfortunately, many designs wait for users (or account holders) to detect anomalies during their own personal reconciliation processes (e.g. when they check their bank statements). The responsibility for detecting the fraud in this circumstance rests on the shoulders of the user. A stronger approach is to develop a proactive, aggressive mechanism for automatically and transparently verifying the transaction history. This should not be done at only a single point during the transaction flow or at one time only, since then there exists the possibility of the fraud occurring after the check or that the mechanism failed, or that it was itself 'attacked' or bypassed. Rather by repeatedly performing the verification process, one can remove this single point of failure and further reduce the amount of risk. Expert or AI systems could provide an excellent intelligent mechanism to provide this monitoring. This leads to an interesting point regarding the case of the two Dubyas. How/why did the bank allow 200 separate, maximum limit withdrawals within a 24 hour period? Their 'single point of failure' mechanism for identifying fraud did just that - it failed! Regardless of how seemingly 'authenticated' each individual transaction may have appeared – the institution is clearly in the wrong for processing those blatantly fraudulent mass transactions.

This naturally leads us to the requirement to seek confirmation. While it may not be possible to get confirmation on each individual transaction – a sequence of transactions (particularly if it is an unnatural, high rate sequence) should require independent confirmation (say mandatory telephonic verification required between every 5 to 10 transactions). In the absence of confirmation, notification

by means of a second channel provides the user (account holder) with the ability to respond rapidly to fraudulent activity. As a consequence, the monetary value of fraud would be limited, hence reducing the fraudster's return on investment. Furthermore acknowledged receipt and reading of messages (if available) could mark the point in time at which the responsibility for identifying the fraud shifts to the user (i.e., the user must notify the institution of receipt of notification of a fraudulent transaction. Failing to do so could make the user liable for subsequent fraudulent transactions.)

4.4.4. Principle 4.

Have the ability to differentiate between the innocent victims and evildoers. Design in the ability to prove innocence.

As has been shown in the case of the two Dubyas, the ability to identify fraud after the fact can be of little usefulness if it is not possible to identify the source thereof. Without this property, the owners of the system must paint both the innocent and repudiating fraudster with the same brush. It is unlikely that they will want to expose themselves to mass fraud due to users repudiating transactions and hence may act to the detriment of the innocent victims. One could argue that there is an obvious moral requirement to protect the victims and that legally financial institutions must take reasonable measures to do so.

Designing in a mechanism with which to verify and prove the 'innocence' of a victim is one solution. Of course the mechanism by which this is achieved could still be attacked and defeated by the fraudster and so should be at least as strong as the weakest component of the system.

4.4.5. Principle 5.

Limit exposure and the attacker's reward

Confirmation, notification and account behaviour analysis are all techniques that can limit the monetary value of the fraud, by providing an early detection mechanism and a preventative response. Designing the solution such that the potential reward does not justify the expense associated with performing the fraud extends this principle further. This is particularly true when the system relies on

a component that offers only some level of resistance but not complete 100% protection. In this circumstance, the attacker must be required to perform a certain minimum amount of work per fraud – hopefully making the fraudulent activities financially unjustifiable. This should be true per individual act of fraud – as opposed to one effort having multiple rewards.

Bank and credit cards provide a good example. To copy a magnetic strip card is trivial and requires an initial outlay of a few hundred dollars. It can be easily achieved during normal day-to-day life. For example, it can be readily done (and is done with surprising frequency) by waitrons at restaurants. A greater technical challenge is presented by smart cards, using dynamic data authentication techniques and in the absence of other potential weaknesses. Perhaps by using a scanning electron microscope or by a novel side channel attack the secrets inside can be recovered, allowing the smart card to be duplicated. However this currently requires a significant amount of time, skill and money for each card that is attacked. The reward per compromised card should be limited to a figure beneath the expected cost of such an attack and so discourage the would be attacker.

4.5. Revising our security target

We thus arrive at a definition of security in the presence of potential failure that yields a survivable solution applicable in real world situations, which we term failure tolerant secure (FTS).
A failure tolerant secure system, is a system that is

- practically secure, or insecure[2]
- possessing the following properties:
    1. A single[3] failure does not destroy the integrity of other security mechanisms.
    2. Loss due to failure is restricted to predefined acceptable limits.
    3. Action of malicious parties can be identified (i.e., the exact point of failure can be identified).
    4. Innocent users are protected in the event of failure.

Having established our new security target largely in response to failure of historical card transaction schemes, one may wish to compare it to the design response of the major players in the card industries. Naively one would hope to see evidence of a similar analysis taking cognisance of the

[2] Suffering from a non master secret failure
[3] Non master secret failure

previous schema's shortcomings – particularly with regards to surviving failure. The exercise is illuminating and shows an arrogance and disregard for the consumer by the designers.

We have already indicated that the current systems have known, existing security weaknesses. Furthermore, we have shown that these systems do not satisfy failure tolerant requirements such as having the ability to distinguish an innocent victim of third party fraud from a repudiating fraudster. Current legal action indicates that the innocent victims are not protected but in fact targeted as fraudsters.

4.5.1. EMV

The current development in the industry is the EMV (Europay, MasterCard and VISA) initiative. Primarily this involves the replacement of magnetic stripe bankcards and credit cards by PIN authenticated smart cards. This allows extra security functionality, namely

- Offline PIN Verification (i.e. using the smart card to verify the PIN), via either
  - Clear PIN submission, or
  - RSA encrypted PIN submission, and
- Card Authentication, using either
  - Static Data, or
  - Dynamic Data.

The card authentication feature is intended to reduce the risk of the card being duplicated. While this was trivial to do with magnetic stripe cards, it will be far more difficult with the smart card. The ideal situation is that it would not be practically possible for the attacker to extract the secrets from the card or, alternatively, require that the attacker have access to the card for a significant time period or cause damage to the card in the process. If one assumes this, then any transaction authenticated by both card and PIN is in fact a valid transaction or is the result of negligent cardholder behaviour.

We note that the concept of security of this solution is identical to that of the magnetic stripe cards and PINs. It implies that the system is designed to be practically secure. There remains no designed intolerance for failure and no attempt at a mechanism to distinguish between fraudsters and innocent victims in the event of the failure (e.g. the possible advent of a simple card cloning scheme). If the system fails, one expects it to fail in a similarly catastrophic manner, since any weakness that contradicts the premise that a seemingly 'authenticated transaction' is in fact a valid transaction, allows Dubya the evildoer to mount effectively the same attack.

Several attacks exist, the best known being the 'static data authentication' attack. In this replay attack, the adversary needs to record only one response from the smart card since the challenge and associated response from the card is static. So, in a system that supports static data authentication, it remains trivial to 'skim' the card.

4.6. Failure Tolerant Extensions

The question arises: Who should trust the smart card? By 'trust' we mean have confidence that the smart card cannot be cracked, forged or duplicated. The 'who' refers to either the consumer who uses the card or the financial institutions and card associations that initially provide the cards.

Perhaps, since the card is in the possession of the consumer, he should be responsible. The consumer, after all, is responsible for the card's safekeeping. However this is not a guarantee that an adversary will not find a method to compromise the device during its normal, responsible use. The institutions on the other hand selected the technology. Thus they have direct control over the level of security offered by a particular device. The institution certainly expects its customers to trust the devices, given that they supply them to the customer. But the fact that a compromised device allows a malicious third party to perform unauthorized transactions, which, being unauthorized, should not be at the expense of the consumer, would seem to imply a level of trust. Further, the traditional defence against the fraudulent account holder (i.e., your PIN was used so you performed the transaction) does not make allowance for the device being compromised and so again indicates a level of trust.

Having established a precedent for the institution to trust a device or smart card (particular without any evidence of physical tampering), we look to techniques to translate this 'obligation to trust' into a failure tolerant mechanism.

4.7. On card transaction history and dispute resolution mechanism

The on card transaction history and dispute resolution mechanism, described below, was conceived as a mechanism to protect the innocent user in the scenario of an attack by a malicious third party (or an error). It provides proof to the institution that the card was not involved in the fraudulent transactions. As such it removes a single point of failure that previously compromised the entire system (by preventing repudiation by fraudsters posing as innocent victims) while at the same time providing a measure of protection of innocent victims from persecution.

4.7.1. Mechanism

Before accepting a transaction, the host requests the card to store a receipt for transaction on the card used. The card must accept this and return an acknowledgment of the receipt and storage thereof before the host continues.

1. The host submits the following data to the ICC, encrypted and signed:
    1) Secure date time stamp
    2) Terminal identification
    3) Transaction amount
    4) Encrypted PIN block
    5) Hash (optional)
2. The card decrypts the packet and verifies signature, hash (if present), terminal ID and time stamp.
3. If it verifies OK, the card stores the data and returns signature of data, record ID, transaction count.
4. The host verifies the signature.

An innocent account holder, who was in possession of his card at all times, can prove this as follows.

4.7.2. Dispute Resolution Procedure

1. The host submits the following data to the ICC, encrypted and signed:
    1) Secure date time stamp
    2) Terminal identification
    3) Transaction amount
    4) Encrypted PIN block
    5) Hash (optional)
2. The card decrypts the packet and verifies the signature and hash (if present).
3. The card checks for a matching record in the transaction log. If one is found, the ICC returns an acknowledgement of that transaction.
4. If the card does not acknowledge the transaction, the institution is obliged to refund it.

This simple technique identifies transaction processing errors or fraudulent activity which has defeated the security design or technologies employed (e.g. a compromise of the card technology). It does

place the obligation to trust the technology on the institution, which as per the earlier discussion is justified.

5. References

[AB96] R. J. Anderson and S. J. Bezuidenhoudt, "On the Reliability of Electronic Payment Systems." *IEEE Transactions on Software Engineering,* v 22 no 5 (1996), pp 294-301. Available from http://www.cl.cam.ac.uk/ftp/users/rja14/meters.ps.gz

[AB01] R. J. Anderson and M. Bond, "API-Level Attacks on Embedded Systems." *IEEE Computer Magazine October 2001*, (2001), pp 67-75.

[AHTV02a] ACI Worldwide, HP Atalla, Diebold, Thales e-Security and VeriFone Inc., "Newly-Formed Payment Consortium Moves Ahead with Endorsement of Secure 3DES Implementation Specification: Industry Leaders Align on New Proposed Key Management Standard." 24 June 2002. Available from http://www.aciworldwide.com/3des/.

[AHTV02b] ACI Worldwide, HP Atalla, Diebold, Thales e-Security and VeriFone Inc., "Global Interoperable Secure Key Exchange Key Block Specification." 24 June 2002.

[AK96] R. Anderson and M. Kuhn, "Tamper Resistance - a Cautionary Note". *Proceedings of the Second USENIX Workshop on Electronic Commerce*, 1996, pp 1-11.

[AK97] R. J. Anderson and M. G. Kuhn, "Low Cost Attacks on Tamper Resistant Devices." *Proceedings of the 1997 Security Protocols Workshop*, Springer LNCS, v 1361 (1998), pp 125-136.

[An92] R. J. Anderson, "UEPS – A Second Generation Electronic Wallet." *Computer Security – ESORICS 92*, Springer LNCS, v 648 (1992), pp 411-418.

[An94a] R.J. Anderson, "Why Cryptosystems Fail." *Communications of the ACM,* v 37 no 11 (1994), pp 32-40. An earlier version is available at http://www.cl.cam.ac.uk/users/rja14/wcf.html

[An94b] R. J. Anderson, "Making Smartcard Systems Robust." *Proceedings of Cardis 94*, 1994, pp 1- 14.

[An94c] R. J. Anderson, "Liability and Computer Security: Nine Principles." *Computer Security – ESORICS 94*, Springer LNCS, v 875 (1994), pp 231-245.

[An00] R. J. Anderson, "The Correctness of Crypto Transaction Sets." *Security Protocols – 8th International Conference*, Springer LNCS, v 2133 (2000), pp 125-127.

[An01] R. J. Anderson, *Security Engineering – A Guide to Building Dependable Distributed Systems*, Wiley, New York (2001), ISBN 0-471-38922-6

[AN95] R. J. Anderson and R. M. Needham, "Programming Satan's Computer." *Computer Science Today*, Springer LNCS, v 1000 (1995), pp 462-441.

[ANSI97] American National Standards Institute (ANSI) Accredited Standards Committee (ASC) X9 - Financial Services (X9-F), "PIN Security Compliance Guideline", TG-3-1997, ASC X9 Secretariat – American Bankers Association, 1997.

[ANSI02] American National Standards Institute (ANSI) Accredited Standards Committee (ASC) X9 - Financial Services (X9-F), "Notice Regarding TDES Key Wrapping Techniques", ASC X9 Secretariat – American Bankers Association, 24 June 2002.

[ANSI X9.8] American National Standards Institute (ANSI) Accredited Standards Committee (ASC) X9 - Financial Services (X9-F), "American National Standard for Financial Services – Banking – Personal Identification Number Management and Security", ASC X9 Secretariat – American Bankers Association, 1995.

[ANSI X9.17] American National Standards Institute (ANSI) Accredited Standards Committee (ASC) X9 - Financial Services (X9-F), "American National Standard for Financial Services – Financial Institution Key Management (Wholesale)", ASC X9 Secretariat – American Bankers Association, 1995.

[ANSI X9.24] American National Standards Institute (ANSI) Accredited Standards Committee (ASC) X9 - Financial Services (X9-F), "American National Standard for Financial Services – Retail Financial Services Symmetric Key Management", ASC X9 Secretariat – American Bankers Association, 2002.

[Ba93] R. Baskerville, "Information systems security design methods: Implications for Information Systems Development." *ACM Computing Surveys (CSUR)*, v 25 no 4 (1993), pp 376-414.

[BAN89] M. Burrows, M. Abadi and R. M. Needham, "A Logic of Authentication." *Proceedings of the Royal Society of London*, v 426 (1989), pp 645-657.

[BDHJNN98] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimbalu, and T. Ngair, "Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults." *Proceedings of the 1997 Security Protocols Workshop*, Springer LNCS, v 1361 (1998), pp 115-124.

[BDL97] D. Boneh, R. DeMillo and R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults." *Advances in Cryptology – EUROCRYPT '97*, Springer LNCS, v 1233 (1997), pp 37-51.

[Bi93] E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys." *Advances in Cryptology EUROCRYPT '93*, Springer LNCS, v 765 (1994), pp. 398-409.

[BMV00] I. Biehl, B. Meyer and V. Muller, "Differential Fault Attacks on Elliptic Curve Cryptosystems." *Advances in Cryptology – CRYPTO 2000*, Springer LNCS, v 1880 (2000), pp 131-146.

[Bo99] D. Boneh, "Twenty Years of Attacks on the RSA Cryptosystem." *Notices of the American Mathematical Society*, v 46 no 2 (1999), pp 203-213.

[Bo01] M. Bond, "Attacks on Cryptoprocessor Transaction Sets." *Cryptographic Hardware and Embedded System - CHES 2001 Third International Workshop*, Springer LNCS, v 2162 (2001), pp. 220-234.

[BS97] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems." *Advances in Cryptology – CRYPTO '97*, Springer LNCS, v 1294 (1997), pp 513-525.

[BZ02] M. Bond and P. Zielinski, "Decimalisation Table Attacks for PIN Cracking." Unpublished manuscript, 2002.

[Cl01] J. S. Clulow, "PIN Recovery Attacks", Prism Technical Report, 2001.

[Cl02] J. S. Clulow, "I Know Your PIN", RSA Europe, 2002.

[CB02] R. Clayton and M. Bond, "Experience Using a Low-Cost FPGA Design to Crack DES Keys." *Cryptographic Hardware and Embedded Systems - CHES 2002*, to appear.

[CCD00] C. Clavier, J. S. Coron and N. Dabbous, "Differential Power Analysis in the Presence of Hardware Countermeasures." *Cryptographic Hardware and Embedded Systems – CHES 2000*, Springer LNCS, v 1965 (2000), pp 252-263.

[CG00] J. S. Coron and L. Goubin, "On Boolean and Arithmetic Masking against Differential Power Analysis." *Cryptographic Hardware and Embedded Systems – CHES 2000*, Springer LNCS, v 1965 (2000), pp 231-237.

[CJRR99] S. Chari, C. S. Jutla, J. R. Rao and P. Rohatgi. "Towards Sound Approaches to Counteract Power-Analysis Attacks." *Advances in Cryptology – CRYPTO '99*, Springer LNCS, v 1666 (1999), pp 398-412.

[Da02] W. Dai, "Crypto++ Library 5.0." Version 5. 2002. http://www.eskimo.com/~weidai/cryptlib.html.

[DHQ86] Y. Desmedt, F. Hoornaert and J. J. Quisquater, "Several Exhaustive Key Search Machines and DES." EUROCRYPT '86, 1986, pp 17-19.

[DKLMQW98] J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestre, J. J. Quisquater and J. L. Willems, "A Practical Implementation of the Timing Attack." Technical Report CG1998/1, Universite catholique de Louvain, 1998. Available from http://www.dice.ucl.ac.be/crypto.

[DPSL99] J. Dyer, R. Perez, S.W. Smith and M. Lindemann. "Application support architecture for a high-performance, programmable secure coprocessor." *22nd National Information Systems Security Conference*, October 1999.

[Ec85] W. van Eck, "Electronic Radiation from Video Display Units: An Eavesdropping Risk?" *Computers and Security*, v 4 (1985) pp 269-286.

[EFF98] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design,* O'Reilly, Sebastopol (1998), ISBN: 1-56592-520-3.

[EMV00a] EMV2000 Integrated Circuit Card Specification for Payment Systems Book 1 - Application Independent ICC to Terminal Interface Requirements, Version 4.0, EMVCo, 2000.

[EMV00b] EMV2000 Integrated Circuit Card Specification for Payment Systems Book 2 - Security and Key Management, Version 4.0, EMVCo, 2000.

[EMV00c] EMV2000 Integrated Circuit Card Specification for Payment Systems Book 3 – Application Development, Version 4.0, EMVCo, 2000.

[EMV00d] EMV2000 Integrated Circuit Card Specification for Payment Systems Book 4 - Cardholder, Attendant and Acquirer Interface Requirements, Version 4.0, EMVCo, 2000.

[GMO01] K. Gandolfi, C. Mourtel and F. Oliver, "Electromagnetic Analysis: Concrete Results." *Cryptographic Hardware and Embedded Systems - CHES 2001*, Springer LNCS, v2162 (2001), pp 251-261.

[Gu96] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory." *Proceedings of Sixth USENIX Security Symposium*, USENIX Association, Berkeley (1996), pp 77-89.

[Gu00] P. Gutmann, "The Design and Verification of a Cryptographic Security Architecture." PhD Thesis, University of Auckland, 2000.

[Gu01] P. Gutmann, "Data Remanence in Semiconductor Devices," *Proceedings of 10th USENIX Security Symposium*, USENIX Association, Berkeley (2001).

[HH] Hackers Homepage, Section #6 Magnetic Stripe and ID Card. http://www.hackershomepage.com/section6.htm

[HH98] H. Handschuh and H. Heys, "A Timing Attack on RC5." *In Workshop Record of Selected Areas of Cryptography - SAC '98*, Springer LNCS, v 1156 (1998), pp 306-318.

[HPS99] H. Handschuh, P. Paillier and J. Stern, "Probing Attacks on Tamper-Resistant Devices." *Cryptographic Hardware and Embedded Systems - CHES '99*, Springer LNCS, v 1717 (1999), pp 303-315.

[HK99] A. Hevia and M. Kiwi, "Strength of Two Data Encryption Standard Implementation Under Timing Attacks." *ACM Transactions on Information and System Security*, v 2 no 4 (1999), pp 416-437.

[IBM01] "Update on CCA DES Key Management." 2001. Available from http://www-3.ibm.com/security/cryptocards/html/ccaupdate.shtml

[KA98] M. G. Kuhn and R. J. Anderson, "Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations." *Information Hiding 1998*, Springer LNCS, v 1525 (1998), pp 124-142.

[KJJ98] P. Kocher, J. Jaffe and B. Jun, "Introduction to Differential Power Analysis and Related Attacks." Technical report, Cryptography Research Inc., 1998. Available from http://www.cryptography.com/dpa/technical/index.html.

[KJJ99] P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis." *Advances in Cryptology - CRYPTO '99*, Springer LNCS, v 1666 (1999), pp 388-397. Available from http://www.cryptography.com/dpa/Dpa.pdf.

[KK99] O. Kommerling and M. G. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors." *USENIX Workshop on Smartcard Technology – Smartcard '99*, USENIX Association, Berkeley (1999), pps 9-20.

[Ko96] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems." *Advances in Cryptology - CRYPTO '96*, Springer LNCS, v 1109 (1996), pp 104-113. An alternate version is available from http://www.cryptography.com/-timingattack/paper.html

[KQ99] F. Koeune and J.J. Quisquater, "A Timing Attack Against Rijndael." Technical Report CG-1999/1, Universite catholique de Louvain, 1999. Available from http://www.dice.ucl.ac.be/crypto.

[KSW96] J. Kelsey, B. Schneier and D. Wagner, "Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES," *Advances in Cryptology - CRYPTO '96*, Springer LNCS, v 1109 (1996), pp 237-251.

[KSW97] J. Kelsey, B. Schneier, and D. Wagner, "Related-Key Cryptanalysis of 3-WAY, Biham-DES,CAST, DES-X, NewDES, RC2, and TEA." *1997 International Conference on Information and Communications Security*, Beijing (1997).

[KSWH00] J. Kelsey, B. Schneier D. Wagner, and C. Hall, "Side Channel Cryptanalysis of Product Ciphers." *Journal of Computer Security*, v 8 no 2-3 (2000), pp 141-158. Available from http://www.counterpane.com/side channel.html.

[Ku02] M. G. Kuhn, "Optical Time-Domain Eavesdropping Risks of CRT Displays." *Proceedings of 2002 IEEE Symposium on Security and Privacy*, (2002), pp. 3-18.

[LU01] J. Loughry and D. A. Umphress, "Information Leakage from Optical Emanations." *ACM Transactions on Information and System Security*, v 5 no 3 (2002), pp 262-289.

[Ma97] D. P. Maher, "Fault induction attacks, tamper resistance, and hostile reverse engineering in perspective." *Financial Cryptography*, Springer LNCS, v 1318 (1997), pp. 109-121.

[Mc] J. McNamara, "The Complete, Unofficial TEMPEST Information Page." Available from http://www.eskimo.com/~joelm/tempest.html

[MDS99a] T. Messerges, E. Dabbish and R. Sloan, "Investigations of Power Analysis Attacks on Smartcards." *USENIX Workshop on Smartcard Technology*, USENIX Association, Berkeley (1999), pp 151-161. Available from http://www.eecs.uic.edu/~tmesserg/papers.html.

[MDS99b] T. Messerges, E. Dabbish, and R. Sloan, "Power Analysis Attacks of Modular Exponentiation in Smart Cards." *Cryptographic Hardware and Embedded Systems - CHES '99*, Springer LNCS, v 1717 (1999), pp 144-157.

[Me00a] T. Messerges, "Securing the AES Finalists Against Power Analysis Attacks." *Fast Software Encryption Workshop - FSE 2000*, Springer LNCS, v 1978 (2000), pp 150-164.

[Me00b] T. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software." *Cryptographic Hardware and Embedded Systems - CHES 2000*, Springer LNCS, v 1965 (2000), pp 238-251.

[MOV96] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton (1997), ISBN 0-8493-8523-7.

[Mu01] J. A. Muir, "Techniques of Side Channel Cryptanalysis." M.Sc Thesis, University of Waterloo, 2001.

[NIST94] National Institute of Standards and Technology, "Security Requirements for Cryptographic Modules." Federal Information Processing Standards Publication 140-1, 1994.

[P94] E. Palmer, "An Introduction to Citadel - a Secure Crypto Coprocessor for Workstations." *Proceedings of the IFIP SEC'94 Conference*, Curacao, 1994.

[RR01] J. R. Rao and P. Rohatgi, "EMpowering Side-Channel Attacks." Technical Report, 2001.

[SA98] S. W. Smith and V. Austel, "Trusting trusted hardware: Towards a formal model for programmable secure coprocessors." *Proceedings of the Third USENIX Workshop on Electronic Commerce*, USENIX Association, Berkeley (1998).

[Sc00] W. Schindler, "A Timing Attack against RSA with the Chinese Remainder Theorem." *Cryptographic Hardware and Embedded Systems - CHES 2000*, Springer LNCS, v 1965 (2000), pp 109-124.

[Sc02] M. Scott, "MIRACL - Multiprecision Integer and Rational Arithmetic C Library". Version 4.74, 2002. Available form http://indigo.ie/~mscott/

[Sh97] A. Shamir, "How to check modular exponentiation." Presented at the rump session of EUROCRYPT'97, Konstanz, 11-15th May 1997.

[Sh00] A. Shamir, "Protecting Smart Cards from Passive Power Analysis with Detached Power Supplies." *Cryptographic Hardware and Embedded Systems - CHES 2000*, Springer LNCS, v 1965 (2000), pp 71-77.

[SKQ01] W. Schindler, F. Koeune and J.J. Quisquater, "Unleashing the Full Power of the Timing Attack." Technical Report CG2001/3, Universite catholique de Louvain, 2001. Available from http://www.dice.ucl.ac.be/crypto.

[Sm96] S. W. Smith, "Secure Coprocessing Applications and Research Issues." Los Alamos Unclassified Release LAUR -96-2805, Los Alamos National Laboratory. August 1996.

[SM97] S. W. Smith and S. M. Matyas, "Authentication for Secure Devices with Limited Cryptography." IBM T. J. Watson Research Center. Design notes, August 1997.

[SPW98] S. W. Smith, E. R. Palmer and S. H. Weingart, "Using a High-Performance, Programmable Secure Coprocessor." *Proceedings of the Second International Conference on Financial Cryptography*. Springer LNCS, v 1465 (1998), pp 73-89.

[SPWA99] S. W. Smith, R. Perez, S. Weingart and V. Austel, "Validating a High-Performance, Programmable Secure Coprocessor." *Proceedings of the 22nd National Information Systems Security Conference*, October 1999.

[St95] D. Stinson, *Cryptography: Theory and Practice*, CRC Press, Boca Raton (1995), ISBN: 1584882069.

[SW99] S. W. Smith and S. H. Weingart, "Building a High-Performance, Programmable Secure Coprocessor." *Computer Networks (Special Issue on Computer Network Security)*. v 31 (1999), pp 831-860.

[TY93] J. D. Tygar and B. S. Yee, "Dyad: A System for Using Physically Secure Coprocessors." *Proceedings of the joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment.* April 1993.

[WC87] S. R. White and L. D. Comerford, "ABYSS: A Trusted Architecture for Software Protection." *IEEE Computer Society Conference on Security and Privacy,* 1987.

[We00] S. H. Weingart, "Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defences." *Cryptographic Hardware and Embedded Systems - CHES 2000*, Springer LNCS, v 1965 (2000), pp 302-317.

[Wr87] P. Wright, *Spy Catcher: The Candid Autobiography of a Senior Intelligence Officer,* Australia: William Heineman (1987), ISBN 0-85561-098-0.

[WWAP91] S. R. White, S. H. Weingart, W. C. Arnold and E. R. Palmer. "Introduction to the citadel architecture: Security in physically exposed environments." Technical Report RC 16672, IBM Research Division, Thomas J.Watson Research Center, Yorktown Heights, NY 10598, May 1991. Version 1.4.

[Ye94] B. S. Yee, "Using Secure Coprocessors." PhD thesis, Computer Science Technical Report CMU-CS-94-149, Carnegie Mellon University, 1994.

[YJ00] S. Yen and M. Joye, "Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis." *IEEE Transactions on Computers,* v 49 no 9 (2000), pp 967-970.

[YT95] B. S. Yee and J. D. Tygar, "Secure Coprocessors in Electronic Commerce Applications." *Proceedings of The First USENIX Workshop on Electronic Commerce*, July 1995.

5.1. Companies

[AEP] Advanced Encryption Technologies. Formerly Baltimore. http://www.aep.com

[ATALLA] Atalla Security Products. http://atalla.nonstop.compaq.com/

[ERACOM] Eracom. http://www.eracom.co.au

[IBM] IBM Security. http://www-.ibm.com/security/

[NCIPHER] nCipher. http://www.ncipher.com

[PRISM] Prism. http://www.prism.co.za

[THALES] Thales (formerly known as Zaxus, formerly known as Racal). http://www.thales-esecurity.com/

[UTIMACO] Utimaco. http://www.utimaco.com.