Encrypted? Randomised? Compromised?
(When Cryptographically Secured Data is Not Secure)

Mike Bond and Jolyon Clulow

Computer Laboratory, University of Cambridge,
JJ Thompson Av., CB3 0FD, UK
{Mike.Bond, Jolyon.Clulow}@cl.cam.ac.uk

Abstract. Protecting data is not simply a case of encrypt and forget: even data with full cryptographic confidentiality and integrity protection can still be subject to information leakage. We consider the issue of information leakage through side channels in protocols. Previous work by Bond and Clulow identified multiple vulnerabilities in APIs for financial PIN processing systems, and suggested remedies; however our work here shows that the fixes do not work, and that the problem of information leakage in these APIs has still not been adequately addressed. We argue that information flow and leakage analysis will play an important role in the security of encrypted databases in the future.

1 Introduction

Processing highly sensitive data is becoming a tricky business. Whether it is secret recipes, passwords and PINs or personal data, system designers now realise that they can't just encrypt and forget. They know that cryptographic data in storage needs to have both confidentiality and integrity protected, and adequate policy (and enforcement of policy) to determine who may access the data, and under what circumstances. However, as more and more sensitive data resides in encrypted form, and trusted computing initiatives drive up the quantity of system data considered sensitive, designers must now come to terms with understanding how to preserve the security of sensitive data *during processing*, and how to regulate its flow and leakage.

In this paper, we examine how security APIs – the interfaces to cryptographic processors – are failing to properly protect sensitive data. This data may be secure in storage, but becomes vulnerable during manipulation. We describe several newly identified flaws in the security APIs of hardware security modules supporting international ATM networks, which demonstrate that eradicating information leakage in encrypted databases is extremely difficult, particularly when the data stored is weak (i.e., easily guessable). We expand upon the previous work by Bond [2, 3] and by Clulow [5], which identified multiple ways to exploit information leakage in PIN processing APIs. They identified the *decimalisation table* input as being vulnerable. Bond suggested that only full authentication of the correct table (or settling on a single table and hard-coding

it) would be an adequate solution. We show that the underlying bias created by the decimalisation table is still exploitable, and that the shortcomings of the ISO-0 format cannot be masked.

2 Existing Attacks against Cryptographically Secured Data

Information leakage is usually thought of in the context of side-channel analysis of physical devices engaged in security protocols. However protocols that operate on protected data may leak a small amount of information, perhaps a few bits or a fraction of a bit, which if identified, can be recovered and accumulated through repeated protocol runs (or sequences of API calls), eventually revealing an entire secret, or bringing it within range of a brute-force search. There exist strong similarities between these channels and conventional side channels exploited during power analysis, timing or emissions attacks. However, a crucial difference is that key material processed in cryptographic algorithms is often processed in a known or obvious sequence, so observation of a characteristic may permit the identification of an explicit bit of the secret. Repetition is used in the attack process to target other distinct bits of the key, or to reduce noise. In the case of information leakage through protocols, the correspondence between the data leaked and key bits of a given secret may be much more complex. A non-trivial algorithm may be required to convert the information revealed about the secret into knowledge of the secret itself.

The game Mastermind™ provides a fine illustration of how protected data is compromised through the processing of the data or a query relating to a property of the data. One player choses a pattern of four coloured pegs that he fixes. This pattern is protected in the sense that it is not directly revealed to the opposing player. One can think of it as an encrypted secret. The second player tries in an iterative manner to guess the pattern. With each guess, the first player tells the second player the number of pegs of the correct colour in the correct place and the number of pegs of the correct colour in the incorrect place. In effect, the second player is given partial information about the secret. The player's objective is to develop an efficient strategy to turn this information into knowledge of the secret itself.

In practice, common sources of leaked information include error conditions and codes. These can be explicitly revealed through a message or indirectly revealed through timing patterns indicating the premature halting or abnormal execution of an operation. Bleichenbacher [1] and Manger [6] have both proposed attacks on the various RSA padding schemes: in these protocols an error response, or the timing of an error response leaks information about the plaintext, allowing a chosen ciphertext attack on RSA. Often such a leak is not visible or not explicitly stated during design process but becomes visible during coding or development. Sometimes one can readily determine that information leakage is occurring, but not be able to clearly identify the semantics of the information leaked, nor how it can be exploited.

Another example is the attack against ISO-0 standard for encrypting PINs under DES described by Clulow in [5]. We briefly reproduce the description of the attack to demonstrate the salient points relevant to our discussion. In the ISO-0 standard, the PIN is formatted into an 8 byte buffer with control information and padding. This buffer is then exclusive-ORed with the customer's Personal Account Number (PAN) before being encrypted under the DES key. We represent this as $\{P \oplus A\}_{K_B}$ where K_B is the key, P the formatted PIN buffer and A the PAN. Whenever the encrypted PIN is verified or re-encrypted under a different key, the process is reversed. As an intermediate step, the PIN is extracted and an integrity check is applied. Since each digit of the PIN is meant to be a decimal digit in the range `0` to `9`, the check simply tests that each hexadecimal PIN digit extracted from the decrypted buffer is less than ten. Should this test fail, it means that either the PAN, key or encrypted PIN is incorrect or corrupted.

Essentially, we can simplify this and represent it in a simple protocol that, given a user specified value for the PAN (which we denote X), tests whether the recovered PIN is decimal. For a single digit PIN, the protocol can be described as follows.

$$\begin{array}{l} A \longrightarrow B : X, \quad \{P \oplus A\}_{K_B} \\ B \longrightarrow A : (P \oplus A) \oplus X < 10 \end{array}$$

Fig. 1. The PIN Integrity Check Protocol

It was not necessarily the explicit intention of the authors of the ISO-0 standard to create this protocol, but it results as a consequence. At first glance, the integrity check seems innocent and benign enough, and indeed perhaps a useful addition that detects unintended errors. However, with a little thought it becomes obvious that repeated execution of this protocol with different values of X quickly leads to the identification of the set $\{P, P \oplus 1\}$. This can clearly be seen from Figure 2. A given value of $P \oplus A$ results in a unique pattern of passes and fails.

The decimalisation attacks against PIN verification, described independently in [5] and [4], similarly extract information from protected data. Again we briefly detail a version of the basic attack that exploits the algorithm, shown in Figure 3, used to verify a PIN encrypted under a working key ($\{guess\}_{WK}$) to leak information. The personal account number (PAN), is encrypted under a PIN Verification Key (PVK). The result is then decimalised using the user supplied decimalisation table (*dectab*). This intermediate value is subtracted, modulo 10, from the clear PIN and the resulting value compared to the supplied value $offset$. If the values match, then the call passes and *true* is returned. Normally, the offset is of limited length, typically four digits, and so only the first four digits are compared.

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

Fig. 2. Table identifying the PIN using the PIN integrity Check Protocol

$$A \longrightarrow B : \{guess\}_{WK}, \{WK\}_{K_{MK}}, PAN, \{PVK\}_{K_{MK}}, dectab, offset$$
$$B \longrightarrow A : true\ if\ guess = dectab(\{PAN\}_{PVK}) + offset$$

Fig. 3. The PIN Verification Protocol

The *dectab* can be thought of as a table of 16 decimal values, an example of which is shown in Figure 4. The example table is interpreted as follows. The hexadecimal character `0` is mapped onto `0`. The hexadecimal character `A` is also mapped onto `0`. This *dectab* is written as `0123456890123456`.

Original Hexadecimal Digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Mapped Decimal Digit	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5

Fig. 4. An Example Decimalisation Table

Consider the effect of changing a single value in the *dectab*. Suppose we modify the first element and use the revised *dectab* `1123456789012345`. If the first four hexadecimal characters of the intermediate value $\{PAN\}_{PVK}$ do not contain the character `0`, then the modified *dectab* has no effect and the original value of the offset will continue to pass. However, should it contain the character `0`, each instance of this character will now get mapped to `1` (instead of the original `0`). This means that the original offset value will no longer pass. The new value of the offset can be determined by iterating through the set of possible offset values. This then allows the attacker to identify which digits changed and hence to calculate the corresponding PIN digits. This attack again demonstrates a scenario where secret data can be compromised, despite being encrypted.

3 Terminology

For our purposes, it is useful to categorize data into two classes, namely:

- *weak* data such as passwords, guessable secrets, other low entropy or non-uniformly distributed data, and
- *strong* data that is random (or possessing no statistically significant distributions), and has a sufficient range to be impractical to guess.

Low entropy-data is a common target of attack in fielded computer systems, typically by means of guessing attacks that seek to brute force the secret. Our approach here, is not a brute force search, but rather an incremental assimilation of information that is slowly leaked.

We exploit techniques that allow us to partition the data into distinct, identifiable sets. By virtue of being able to identify the set membership, a non-negligible amount of information can be extracted. For example, encrypted PINs can be partitioned into distinct sets. Whilst we may not know the clear values of the PINs given two encrypted PIN blocks, we can still categorise them using a comparison for equality based upon the encrypted values. This particular technique makes use of what we call the *visible storage* of the secret. Visible storage is at work in electronic code book and frequency analysis type attacks.

The second (more general) technique is *manipulable testing* in which a configurable or parameterised test can be applied to the data and used to partition the data set. The attacks listed in the previous section exploit this paradigm. For example, the PIN integrity check offered a method for testing whether a given hexadecimal digit was a member of the set of decimal digits. Similarly, the decimalisation table attack tested the membership of the specified PIN based on the configurable decimalisation table. Using this technique, the information required to perform the partitioning is not available based on external storage, but rather revealed through processing or testing and represents a form of information leakage. We further classify leakages as either *inadvertent*, such as the decimal digit integrity check for PINs or *intentional*, such as through the verification function.

We now consider some specific *new* examples of information leakage in financial APIs.

4 Persistent Problems with Decimalisation Tables

A number of PIN processing APIs support transactions which generate PINs under a variety of algorithms, normally sending them to a secure printer.

However, many HSMs will also output generated PINs in encrypted form, instead of sending them in the clear to a secure printer. Such a transaction is useful for performing high-volume re-issue of PINs, where a bank's printing facilities are inadequate. In this scenario, the bank shares a top-level key by courier with a special printing facility, then sets up a *working key* (WK) under which the generated PINs are encrypted, in ISO-0 pinblock form. Such commands are

$$\begin{array}{ll} U \longrightarrow C: & PAN,\ \{PVK\}_{Km1},\ offset \\ C \longrightarrow Printer: & dectab(\{PAN\}_{PVK}) + offset \end{array}$$

Fig. 5. The PIN Printing Protocol

present in the APIs of Thales RG series, Atalla NSP and nCipher payShield APIs.

For the purposes of this paper, a generic financial *"Encrypted PIN Generate"* protocol is described in Figure 6. It derives a PIN by encrypting a PAN with the PVK (as described before), adds an initial offset, then stores it as an ISO-0 PIN block. To discount the previously identified attacks, imagine that the default decimalisation table (`0123456789012345`) is hardwired into the command, and cannot be altered.

$$\begin{array}{ll} U \longrightarrow C: & PAN,\ \{PVK\}_{Km1},\ \{WK\}_{Km2},\ offset \\ C \longrightarrow U: & \{\ (dectab(\{PAN\}_{PVK}) + offset) \oplus PAN\ \}_{WK} \end{array}$$

Fig. 6. The Encrypted PIN Generation Protocol

4.1 Known Attacks

The facility to add an offset is known to introduce risks when combined with other weaknesses. For instance, API designers are aware that should the encrypted PIN block corresponding to any trial PIN become known, it may lead to discovery of the true PIN. By repeatedly executing the generate command, and looping through the offset value an attacker could discover the offset which causes the known block to be generated, and thus determine the difference between the true PIN and the trial PIN. Practical countermeasures taken in operational environments include trying to segregate trial PINs (chosen PINs entered at an ATM machine) from freshly generated PINs by encrypting them with a different working key to that available for output of encrypted PINs, or by separating the entire generation facility from the verification facility.

4.2 A Statistical Attack

However, there is another way in which encrypted PIN blocks can be identified. If an attacker cycles through the offset field ($offset_i = i, i = 0000 \ldots 9999$), he can assemble a list of all possible encrypted PIN blocks for a particular fixed account, together with the relationship between each. The attacker stores tuples of the form:

$$\langle\ offset_i\ ,\ \{(\{PAN\}_{PVK} + offset_i) \oplus PAN\}_{WK}\ \rangle$$

However, he still does not know which are which. Before, the single chosen trial PIN gave him a starting point from which to unravel, and he could use the stored relationship to look up the offset between the trial PIN and the target PIN. However, the heavy cost of a visit to an ATM is required per account attacked, and furthermore, there must be interception software in place in the live system. Instead, he can use the *statistical distribution* of PIN blocks as a starting point, and unravel the values of the rest of the PIN blocks from there.

The ISO-0 PIN block format includes the PAN, so the same PIN encrypted for different accounts will not be comparable, however, the format does not bind the generated PIN to a particular PIN derivation key. Thus the attacker can collect a large enough set of PIN derivation keys, perhaps by generating them himself or through the use of conjuring techniques (see [2] for a description). He uses each key to derive a 'correct' PIN from the fixed PAN of the target account, keeping the offset constant, say at `0000`. This generates a large set of encrypted PINs with many collisions (that is, many have the same clear PIN value, and hence, the same encrypted value). The encrypted PIN blocks are visible and can be partitioned into sets based on the encrypted value (which obviously corresponds to the clear PIN value).

Here is the trick: the derived PINs generated under different PIN derivation keys will be biased in accordance with the decimalisation table. This creates a unique frequency distribution of occurrence of encrypted PIN block outputs that is reflected in the size of the partitions. Viewing these values in a histogram, the more common PINs will become readily apparent. This accounts for the requirement of having a suitably large set of keys. Combined with an ordering on the encrypted PIN block values (the partitions), created by cycling through the offsets, the attacker has everything he needs to discover PINs for that account.

To see more clearly, consider a simplified example using PINs of only one digit. The action of the decimalisation table in the generation process causes PINs 0,1,2,3,4 & 5 to be twice as likely as PINs 6–9. The chart on the left in Figure 7 shows the ten possible PINs assembled in a loop (as offsets are calculated modulo 10), each one higher than the previous. The depth of shading represents frequency: 0–5 dark, 6–9 light. The chart on the right of Figure 7 shows (artificially shortened) encrypted PIN blocks corresponding to the ten PINs. By cycling through offsets, they have been assembled into order, and by generating PINs under a range of PVKs, the relative frequencies of generation of each block have been marked with shading. It is easy to visually align the two distributions and observe that the encrypted block `2F2C` must correspond to a PIN of `0`.

The estimated transaction cost of such an attack is 10,000 for the loop, and roughly 2,000–10,000 data samples to determine the distribution. With modern transaction rates of around 300 transactions per second, this equates to about 30 seconds per PIN. The attack should work on any financial HSM where a sufficient number of PVKs can be obtained.

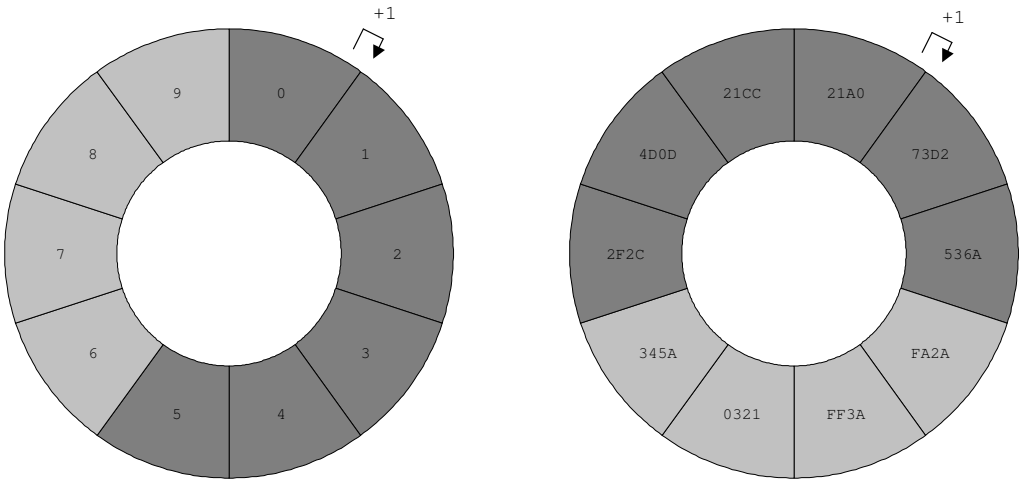

Fig. 7. Cyclic frequency distributions of PIN digits and encrypted PIN blocks (higher frequency shown in darker grey)

4.3 Implementation

We designed and implemented an algorithm employing this general principle using C code, with a software simulation of the HSM transaction. We found that with 10,000 transactions the PIN could be accurately determined for an account in a repeatable way. Our initial implementation in fact did not make full (information theoretic) use of the data returned, but applied some approximations and heuristics to analyse the histograms.

Closer examination of the possibilities for algorithm design, revealed a trade-off between effort conceptually spent creating a frequency distribution, and effort spent interrogating it to determine the value of a particular digit. In fact, when the single digit PIN example is generalised, because the offset is calculated modulo 10 independently for each digit, the loop alignment problem does not extend to a larger loop, but along four separate axes (one could possibly extend the loop analogy to visualise alignment in four dimensions – of hyper-tori?). We have yet to develop an optimal algorithm, but given the additional constraints and details of real HSM APIs, creating an optimal algorithm for a generic API command is of little practical value.

5 ISO-0 Collision Attack

When we consider the overlap between the sets of all possible PIN blocks for different PANs, we see that there are other non-uniformities. Because PINs are generated over a range of digits 0–9, while the nibble datatype used to store them ranges from 0–F, uneven distributions of encrypted PIN blocks can also be observed in accordance with this characteristic. Consider a similar encrypted PIN generation command to the one which is the subject of the attack in Section 4.

To aid with the explanation, this example is also restricted to single digit PINs and PANs. Suppose that the attacker repeatedly generates PINs for some fixed PAN, until all possible blocks have been observed. For our single digit PIN example, this amounts to ten different encrypted blocks. This could be achieved by repeating the PIN generation protocol (described earlier in Figure 6) with different generation keys a sufficient number of times and then excluding the repetitions from the list. A more efficient solution would be to make use of the offset field. Increasing the offset by one with each subsequent call results in a corresponding increase of one in the PIN value. Alternatively, a random generation method could be used, an example of which is the VISA-PVV method that is described later.

Again, the attacker can see the encrypted PIN blocks (visible storage), but this time has neither their ordering, nor a known starting point. However, he can perform a second generation run of the PIN space using a different PAN. We use the notation PAN_2 to denote the PAN associated with the second run. The attacker can now compare the sets of encrypted PIN blocks resulting from the two runs. Both runs contain all the PINs. However, the two runs used different PANs which affects how the set of PINs are encrypted and stored. Recall that the ISO-0 format exclusive-ORs the PIN with the PAN prior to encryption under the working key. Now the exclusive-OR of many PINs with the PAN_1 from the first run will match the exclusive-OR of other PINs and the different PAN_2 from the second run. Whenever this occurs, the encrypted blocks will match as well (that is, have the same value). Conversely, each encrypted PIN block that exists in only one of the lists (wlog we assume the list from the first run), corresponds to a value of $PIN + PAN_1$ that is not achievable in the second run (that is, is not an element of the set $\{0 + PAN_2, 1 + PAN_2, \ldots, 9 + PAN_2\}$). This allows the attacker to determine a set of possible values for $PIN + PAN_1$, and hence for PIN.

We illustrate this technique with a numerical example. The attacker has constructed the two lists of all the possible encrypted blocks for the accounts with PANs `7` and `0`, as shown in Figure 8.

He can easily observe that the encrypted block `AC42` from the left hand list does not occur in the right hand list, and likewise for the encrypted block `9A91`. Therefore, he knows that this encrypted block corresponds to a combination of PIN and PAN which cannot be produced by exclusive-ORing with a PAN of `0`. Given the PAN of `7` on the target account, he can deduce that the corresponding PIN to `AC42` is either `8` or `9`. This deduction is the same as that performed

PAN	PIN	(PAN⊕PIN)	Encrypted Block	PAN	PIN	(PAN⊕PIN)	Encrypted Block
7	0	7	2F2C	0	0	0	21A0
7	1	6	345A	0	1	1	73D2
7	2	5	0321	0	2	2	536A
7	3	4	FF3A	0	3	3	FA2A
7	4	3	FA2A	0	4	4	FF3A
7	5	2	536A	0	5	5	0321
7	6	1	73D2	0	6	6	345A
7	7	0	21A0	0	7	7	2F2C
7	8	F	AC42	0	8	8	4D0D
7	9	E	9A91	0	9	9	21CC

Fig. 8. Sets of encrypted all PIN blocks for accounts with PANs 7 and 0

in the ISO-0 PAN modification attack [5], and hence has the same associated restrictions.

6 PVV Clash Attack

The VISA Pin Verification Value (PVV) method for processing customer PINs has advantages over the IBM 3624 method in that it can use an unbiased random number source to generate the PINs themselves. This is immediately advantageous when considering storage scenarios for encrypted PINs. However, the verification method has a peculiar property. The first stage of the verification process is to construct a *transaction security parameter* (TSP), consisting of the PAN, an issue number referring to which PIN derivation key is to be used, and the trial PIN itself. The TSP is then encrypted with the PIN master key, then truncated and decimalised, yielding a four digit "PIN Verification Value" which can be stored in the clear in the banks main database, or written to the magnetic stripe on an ATM card itself.

The weakness is simple: due to the short length of the PVV, and considering the encryption function as a random oracle, the birthday paradox ensures that multiple transaction security values will produce the same PVV as a result. In practical terms, there will be several correct PINs for each account! Approximately 60% of accounts will have two or more correct PINs. Furthermore, about 0.5% of accounts have five or more correct PINs – an corrupt insider could use PVV generation transactions to observe clashes taking place, and pass on details of the weaker accounts to outsiders.

In practice, because accounts with ten or more correct PINs are very rare, this weakness of the algorithm is of limited practical value; however the legal implications of having multiple correct PINs for an account are interesting, and completely uncharted territory. Interestingly, Opel reports the successful use of two different PINs to withdraw money from an ATM using a VISA credit card [7],

in what could possibly be the first documented case of a PVV clash out 'in the wild'.

7 Towards Solutions

Laying down advice for designers developing *brand new* systems should not be too hard. When the nature of data to be processed is within the domain of the designer, for instance if they are choosing secrets to store for authentication purposes, it is clear that strong secrets should be used whenever possible. Weak secrets, whether too short, or with non-uniform statistical distributions are inherently at risk, especially when considering the necessary leakage of the authentication verification function. The designers may also include standard intrusion detection features such as rate limiting, or lock-out to limit information leakage through authentication verifications. When processing data, new systems should avoid data-dependent error codes and timing characteristics. In particular, any error which can be thrown after unauthenticated inputs have been combined with authenticated ones should be carefully scrutinised for leakage. Data storage formats should follow the usual best practices – in particular encryption should be *randomised* to prevent comparison of ciphertext blocks, and MACs should be used for integrity.

Shoring up *existing systems* against information leakage through APIs and protocols is a much harder proposition. The weak secrets to be protected may be core assets (as with encrypted databases of personal data), and not easily discarded and regenerated. Alternatively, they may once have been arbitrary, but now are irreversibly entrenched. Indeed, as far as financial PIN processing systems are concerned, the cost of distribution of new PINs may be prohibitively high, and upgrade of an individual bank's system may only be possible if interoperability can be maintained. The designers thus have a much harder task in regulating information flow, when they must hide the weakness of the data, and have constrained cryptographic resources to do so. It seems clear that randomised encryption would be beneficial for PIN storage, and there is evidence of a move toward this: the ISO-0 PIN block may soon be superseded by the ISO-3 format, which includes further randomisation. However, increasing the storage format up from a single 8-byte block to allow a cryptographic hash for integrity is simply out of the question. One must be very careful when proposing minimal fixes to a system: generic problems usually require generic fixes.

The hardest information leakage problems are those associated with limiting the rate of leakage when there is already an intentional leak. This extends beyond verification of authentication data into encrypted database query, where the data returned in unencrypted form may be many hundreds of bits, and could potentially leak a huge amount of information. Our hope is that existing expertise from other areas of computer science, such as information flow in multi-level secure systems, and anonymisation of statistics during information retrieval can be brought to bear to increase understanding about information leakage in security protocols and APIs.

8 Conclusions

We have shown that information leakage attacks pose a significant threat in the realm of security APIs and protocols, as well as in the wider context where they are already well-known. In particular, we documented several new attacks on APIs for financial PIN processing systems that supersede the best previous strategies, and show that some of the dangerous architectural features cannot be made secure without applying truly generic fixes.

9 Acknowledgments

We would like to thank George Danezis for his help and advice, and acknowledge the generous funding of the CMI Institute and the Cecil Renaud Educational and Charitable Trust.

References

1. D. Bleichenbacher, "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1", Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology, Springer LNCS 1462, p. 1–12, 1998
2. M. Bond "Attacks on Cryptoprocessor Transaction Sets", CHES 2001, Springer LNCS 2162, p. 220–234
3. M. Bond "Understanding Security APIs", Phd. Thesis, available at the URL http://www.cl.cam.ac.uk/users/mkb23/research.html, 2004
4. M. Bond, P. Zielinski "Decimalisation Table Attacks for PIN Cracking", University of Cambridge Computer Laboratory Technical Report TR-560, Jan 2003
5. J. Clulow "The Design and Analysis of Cryptographic Application Programming Interfaces", MSc. Dissertation, University of Natal, South Africa, http://www.cl.cam.ac.uk/users/jc407/, 2003
6. J. Manger, "A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0", Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, Springer LNCS 2139, p. 230–238, 2001
7. T. Opel, Private communication. Forthcoming description at the URL http://www.kreditkartendiebe.de/