Low Cost Attacks on Tamper Resistant Devices

Ross Anderson[1], Markus Kuhn[2]

[1] Computer Laboratory, Pembroke Street, Cambridge CB2 3QG, UK
rja14@cl.cam.ac.uk
[2] COAST Laboratory, Purdue University, West Lafayette, IN 47907, USA
kuhn@cs.purdue.edu

Abstract. There has been considerable recent interest in the level of tamper resistance that can be provided by low cost devices such as smartcards. It is known that such devices can be reverse engineered using chip testing equipment, but a state of the art semiconductor laboratory costs millions of dollars. In this paper, we describe a number of attacks that can be mounted by opponents with much shallower pockets.
Three of them involve special (but low cost) equipment: differential fault analysis, chip rewriting, and memory remanence. There are also attacks based on good old fashioned protocol failure which may not require any special equipment at all. We describe and give examples of each of these. Some of our attacks are significant improvements on the state of the art; others are useful cautionary tales. Together, they show that building tamper resistant devices, and using them effectively, is much harder than it looks.

1 Introduction

An increasing number of large and important systems, from pay-TV through GSM mobile phones and prepayment gas meters to smartcard electronic wallets, rely to a greater or lesser extent on the tamper resistance properties of smartcards and other specialist security processors.

This tamper resistance is not absolute: an opponent with access to semiconductor test equipment can retrieve key material from a smartcard chip by direct observation and manipulation of the chip's components. It is generally believed that, given sufficient investment, any chip-sized tamper resistant device can be penetrated in this way. A number of less expensive techniques for attacking specific tamper resistant devices are also known [2].

So the level of tamper resistance offered by any particular product can be measured by the time and cost penalty that the protective mechanisms impose on the attacker. Estimating these penalties is clearly an important problem, but is one to which security researchers, evaluators and engineers have paid less attention than perhaps it deserves. (The relatively short bibliography at the end of this article bears witness to that.)

We will adopt the taxonomy of attackers proposed by IBM to guide designers of security systems that rely to some extent on tamper resistance [1]:

Class I (clever outsiders): They are often very intelligent but may have insufficient knowledge of the system. They may have access to only moderately sophisticated equipment. They often try to take advantage of an existing weakness in the system, rather than try to create one.

Class II (knowledgeable insiders): They have substantial specialized technical education and experience. They have varying degrees of understanding of parts of the system but potential access to most of it. They often have highly sophisticated tools and instruments for analysis.

Class III (funded organisations): They are able to assemble teams of specialists with related and complementary skills backed by great funding resources. They are capable of in-depth analysis of the system, designing sophisticated attacks, and using the most advanced analysis tools. They may use Class II adversaries as part of the attack team.

In this paper, we present and develop a number of techniques that can make smartcards and other tamper resistant devices vulnerable to class II or even class I attackers.

2 Differential Fault Analysis

In [5], Biham and Shamir announced an attack on DES based on 200 ciphertexts in which one-bit errors have been induced by environmental stress. The fault model they used had been proposed by Boneh and others in [11] and its effects investigated further in [16, 10]. It assumes that by exposing a processor to a low level of ionising radiation, or some other comparable insult, that one-bit errors can be induced in the data used and specifically in the key material fed into the successive rounds.

In [6], it is shown how this method could be extended to reverse engineer algorithms whose structure is unknown. In each case, the critical observation is that errors that occur in the last few rounds of the cipher leak information about the key, or algorithm structure, respectively. In [7], a number of further results are given; if faults can be induced in the last one or two rounds of the algorithm, then fewer faulty ciphertexts are needed. In [11], it was shown that on a similar fault model, attacks could be carried out on public key systems. In particular, an RSA modulus could be factored given a number of faulty signatures.

The problem with these proposed attacks is that no-one has demonstrated the feasibility of the fault model. Indeed, with many security processors, the key material is held in EEPROM together with several kilobytes of executable code; so it is likely that a random one-bit error which did have an effect on the device's behaviour would be more likely to crash the processor or yield an uninformative error than to produce a faulty ciphertext of the kind required for the above attacks.

In this section, we show that a different, and more realistic, fault model gives significantly better attacks. In the following sections, we will discuss some other faults that can be induced by low budget attackers and show that they, too, lead to feasible attacks. Many of these attacks can also be extended to cases in which we do not initially know the algorithm in use, or where our knowledge of the system is otherwise imperfect. They fall squarely within the definition of what a class I opponent might do.

2.1 A Realistic Differential Attack

In [2], we discussed an attack that has been used by amateur hackers in assaults on pay-TV smartcards. The idea is to apply a glitch — a rapid transient — in either the clock or the power supply to the chip. Typical attacks had involved replacing a 5 MHz clock pulse to a smartcard with one or more 20 MHz pulses. Because of the different number of gate delays in various signal paths and the varying RC parameters of the circuits on the chip, this affects only some signals, and by varying the precise timing and duration of the glitch, the CPU can be made to execute a number of completely different wrong instructions. These will vary from one instance of the chip to another, but can be found by a systematic search using hardware that can be built at home.

We do not claim to have invented this attack; it appears to have originated in the pay-TV hacking community, which has known about it for at least a year. In the form described in [2], it involved a loop that writes the contents of a limited memory range to the serial port:

```
1  b = answer_address
2  a = answer_length
3  if (a == 0) goto 8
4  transmit(*b)
5  b = b + 1
6  a = a - 1
7  goto 3
8  ...
```

The idea is to find a glitch that increases the program counter as usual but transforms either the conditional jump in line 3 or the loop variable decrement in line 6 into something else. Then, by repeating the glitch, the entire contents of memory can be dumped.

When applied at the algorithm level rather than at the level of control code, this attack is also highly effective, as we shall now see. The import of this work is that attacks based on inducing errors in instruction code are easier, and more informative, than attacks based on inducing errors in data.

2.2 Attacking RSA

The Lenstra variant of the attack on RSA goes as follows: if a smartcard computes an RSA signature S on a message M modulo $n = pq$ by computing it modulo p and q separately and then combining them using the Chinese Remainder Theorem, and if an error can be induced in either of the former computations, then we can factor n at once. If e is the public exponent, and the 'signature' $S = M^d (\mathrm{mod}\ pq)$ is correct modulo p but incorrect modulo q, then we will have

$$p = \gcd(n, S^e - M) \tag{1}$$

This is ideal for a glitch attack. As the card spends most of its time calculating the signature mod p and mod q, and almost any glitch that affects the output will do, we do not have to be selective about where in the instruction sequence the glitch is applied. Since only a single signature is needed, the attack can be performed online: a Mafia shop's point-of-sale terminal can apply the glitch, factor the modulus, calculate what the correct signature should be, and send this on to the bank, all in real time.

Thus the Mafia can harvest RSA secret keys without the customer or his bank noticing anything untoward about the transaction performed at their shop. Given that implementers of the new EMV electronic purse system propose to have only 10,000 different RSA secret keys per issuing bank [14], the Mafia may soon be able to forge cards for a substantial proportion of the user population.

2.3 Attacking DES

When we can cause an instruction of our choice to fail, then there are several fairly straightforward ways to attack DES. We can remove one of the 8-bit xor operations that are used to combine the round keys with the inputs to the S-boxes from the last two rounds of the cipher, and repeat this for each of these key bytes in turn. The erroneous ciphertext outputs that we receive as a result of this attack will each differ from the genuine ciphertext in the output of usually two, and sometimes three, S-boxes. Using the techniques of differential cryptanalysis, we obtain about five bits of information about the eight keybits that were not xor'ed as a result of the induced fault. So, for example, six ciphertexts with faulty last rounds should give us about 30 bits of the key, leaving an easy keysearch.

An even faster attack is to reduce the number of rounds in DES to one or two by corrupting the appropriate loop variable or conditional jump, as in the protocol attack described above. Then the key can be found by inspection. The practicality of this attack will depend on the implementation detail.

Thus DES can be attacked with somewhere between one and ten faulty ciphertexts. But how realistic is it to assume that we will be able to target particular instructions?

In most smartcards, the manufacturer supplies a number of routines in ROM. Though sometimes presented as an 'operating system', the ROM code is more of a library or toolkit that enables application developers to manage communications and other facilities. Its routines usually include the DES algorithm (or a proprietary algorithm such as Telepass), and by buying the manufacturer's smartcard development toolkit (for typically a few thousand dollars) an attacker can get full documentation plus real specimens for testing. In this case, individual DES instructions can be targeted.

When confronted with an unfamiliar implementation, we may have to experiment somewhat (we have to do this anyway with each card in order to find the correct glitch parameters). However the search space is relatively small, and on looking at a few DES implementations it becomes clear that we can usually recognise the effects of removing a single instruction from either of the last two rounds. (In fact, many of these instructions yield almost as much information when removed from the implementation as the key xor instructions do.)

2.4 Reverse Engineering an Unknown Block Cipher

We can always apply clock and power glitches until simple statistical tests suddenly show a high dependency between the input and output bits of the encryption function, indicating that we have succeeded in reducing the number of rounds. This may be practical even where the implementation details are unknown, which leads us to ask whether we can use our attack techniques to reverse engineer an unknow algorithm, such as Skipjack, without needing to use expensive chip testing equipment.

In [6, 7], Biham and Shamir discuss this problem in their fault model of one-bit random data errors. As before, they identify faults that affected only the last round or rounds; this can be done by looking for ciphertexts at a low Hamming distance from each other. They then identify which output bits correspond to the left and right halves, and next look at which bits in the left half are affected by one bit changes in the last-but-one right half. In the case of a cipher such as DES with S-boxes, the structure will quickly become clear and with enough ciphertexts the values of the S-boxes can be reconstructed. They report that with 500 ciphertexts the gross structure can be recovered, and with about 10,000 the S-box entries themselves can be found.

Our technique of causing faults in instructions rather than in data bits is more effective here, too. We can attack the last instruction, then the second last instruction, and so on. The number of ciphertexts required for this attack is about the same as for Biham and Shamir's.

Let us now consider an actual classified algorithm. 'Red Pike' was designed by GCHQ for encrypting UK government traffic classified up to 'Restricted', and the Department of Health wishes to use it to encrypt medical records. The British Medical Association preferred that an algorithm be chosen that had been in the open literature for at least two years and had withstood serious attempts to find shortcut attacks (3DES, Blowfish, SAFER K-128, WAKE, ...).

In order to try and persuade the BMA that Red Pike was sound, the government commissioned a study of it by four academics [18]. This study states that Red Pike 'uses the same basic operations as RC5' (p 4); its principal operations are add, exclusive or, and left shift. It 'has no look-up tables, virtually no key schedule and requires only five lines of code' (p 4). Other hints include that 'the influence of each key bit quickly cascades' (p 10) and 'each encryption involves of the order of 100 operations' (p 19).

We can thus estimate the effort of reverse engineering Red Pike from a tamper resistant hardware implementation by considering the effort needed to mount a similar attack on RC5 [19].

Removing the last operation — the addition of key material — yields an output in which the right hand side is different (it is `(B xor A) shl A` where `A` and `B` are the left and right halves respectively). This suggests, correctly, that the algorithm is a balanced Feistel cipher without a final permutation. Removing the next operation — the shift — makes clear that it was a 32 bit circular shift but without revealing how it was parametrised. Removing the next operation — the xor — is transparent, and the next — the addition of key material in the previous round — yields an output with the values A and B in the above expression. It thus makes the full structure of the data-dependent rotation clear. The attacker can now guess that the algorithm is defined by

```
A = ((A xor B) shl B) op key
B = ((B xor A) shl A) op key
```

Reverse engineering RC5's rather complex key schedule (and deducing that '`op`' is actually +) would require single-stepping back through it separately; but if we guess that '`op`' is +, we can find the round key bits directly by working back through the rounds of encryption.

So, apart from its key schedule, RC5 may be about the worst possible algorithm choice for secret-algorithm hardware applications, where some implementations may be vulnerable to glitch attacks. If Red Pike is similar but with a simpler key schedule, then it could be more vulnerable still. However, since the government plans to make Red Pike available eventually in software, this is not a direct criticism of the design or choice of that algorithm.

It does all suggest, though, that secret-hardware algorithms should be more complex; large S-boxes kept in EEPROM (that is separate from the program memory) may be one way of pushing up the cost of an attack. Other protective measures that prudent designers would consider include error detection, multiple encryption with voting, and designing the key schedule so that the keys from a small number of rounds are not enough for a break.

3 Chip Rewriting Attacks

Where the implementation is familiar, there are a number of ways to extract keys from the card by targeting specific gates or fuses or by overwriting specific memory locations. Bovenlander has described breaking smartcards by using two microprobe needles to bridge the fuse blown at the end of the card test cycle, and using the re-enabled test routine to read out the memory contents [12]. Even where this is not possible, memory cells can be attacked; this can also be done on a relatively modest budget.

3.1 ROM overwrite attacks

Single bits in a ROM can be overwritten using a laser cutter microscope, and where the DES implementation is well known, we can find one bit (or a small number of bits) with the property that changing it will enable the key to be extracted easily. The details will depend on the implementation but we might well be able, for example, to make a jump instruction unconditional and thus reduce the number of rounds to one or two. We can also progressively remove instructions such as exclusive-or's of key material.

Where we have incomplete information on the implementation, ROM overwriting attacks can be used in other ways. For example, DES S-boxes in ROM can be identified and a number of their bits overwritten such that the encryption function becomes a linear transformation over GF(2); we can then extract the key from a single plaintext/ciphertext pair.

3.2 EEPROM modification attacks

Where the algorithm is kept in EEPROM, we can use two microprobing needles to set or reset the target bit [17]. We can use this technique to carry out the above attacks; but the fact that we can both set and reset bits opens up still more opportunities.

Recall that the DES algorithm uses keys with odd parity, and a proper implementation will require that a key with the wrong parity will cause an error message to be returned (the VISA security module described below is an example of such equipment). Suppose further that we know the location of the DES key in memory but cannot read it directly; this could well be the case where the key is kept in EEPROM at a known location (smartcard software writers often locate keys at the bottom end of EEPROM memory), but we lack the equipment to carry out the attacks described in [2]. We can proceed as follows.

Set the first bit of the EEPROM containing the target DES key to 1 (or 0, the choice doesn't matter) and operate the device. If it still works, the keybit was a 1. If you get a 'key parity error' message, then the bit was zero. Move on to the next bit; set it to 1 and see if this changes the device's response (from encryption to error or vice versa). Even where the protocol uses some form of key redundancy that we do not understand, we can react to error messages by simple changing the keybit back to its original value.

Both microprobes and laser cutter microscopes are often found in universities — the former in electrical engineering departments, and the latter in cellular biology laboratories. Undergraduates can obtain unsupervised access to them; other class I attackers can purchase them for at most a few thousand dollars.

3.3 Gate destruction attacks

At the rump session of the 1997 workshop on Fast Software Encryption, Eli Biham and Adi Shamir presented a novel and interesting attack on DES. The idea is to use a laser cutter to destroy an individual gate in the hardware implementation of a known block cipher.

The example they gave was DES, which is typically implemented with hardware for a single round, plus a register that holds the output of round k and sends it back as the input to round $k+1$. Biham and Shamir pointed out that if the least significant bit of this register is stuck, then the effect is that the least significant bit of the output of the round function is set to zero. By comparing the least significant six bits of the left half and the right half, several bits of key can be recovered; given about ten ciphertexts from a chip that has been damaged in this way, information about the round keys of previous rounds can be deduced using the techniques of differential cryptanalysis, and enough of the key can be recovered to make keysearch easy.

This is an extremely impressive attack, and in fact the first one that works against ciphers such as DES when the plaintext is completely unknown. (This is the case in many smartcard applications where the card uses successive payment transactions to report its internal state to the issuer.)

We observe that there is a simple countermeasure to this new attack: a chip modified in this way will have the property that encryption and decryption are no longer inverses. So a simple self-test procedure can be added that takes an arbitrary input, encrypts and decrypts under an arbitrary key, and compares the result with the original block. (This test is already being implemented by one of our clients in a chip currently under development.)

4 Memory Remanence Attacks

In a brilliant USENIX paper [15], Gutman described the mechanisms that cause both static and dynamic RAM to 'remember' values that they have stored for a long period of time. A prudent security engineer will ask what the effect of this is in the real world.

We looked at a security module used in a bank. Many banks use a system devised by IBM and refined by VISA to manage the personal identification numbers (PINs) issued to customers for use with automatic teller machines [4]. The PIN is derived from the account number by encrypting it with a 'PIN key', decimalising the result and adding a decimal 'offset' (without carry) to get the PIN the customer must enter. (The offset's function is to enable the customer to choose his own PIN.) An example of the calculation is [4]:

```
Account number:        8807012345691715
PIN key:               FEFEFEFEFEFEFEFE
Result of DES:         A2CE126C69AEC82D
Result decimalised:    0224126269042823
Natural PIN:           0224
Offset:                6565
Customer PIN:          6789
```

The function of the security module is to perform all these cryptographic operations, plus associated key management routines, in trusted hardware, so as to support a dual control policy: no single member of any bank's staff should have access to a customer PIN [20]. Thus, for example, the module will only perform a 'verify PIN' command if the PIN is supplied encrypted under a key allocated to an automatic teller machine or to a corresponding bank. In this way, bank programmers are prevented from using the security module as an oracle to perform exhaustive PIN search.

In order to enforce this, the security module needs to be able to mark keys as belonging to a particular functionality class. It does this by encrypting them with 3DES under one of 12 pairs of DES master keys that are stored in low memory. Thus for example ATM keys are stored encrypted under master keys 14 and 15, while the working keys used to communicate with other banks are stored encrypted under master keys 6 and 7. The encrypted values of long term keys such as the PIN key are often included inline in application code and are thus well known to the bank's programming staff.

So security depends on the module's tamper resistance, and this is provided for by lid switches that cut power to the key memory when the unit is opened (it needs servicing every few years to change the battery). Master keys are loaded back afterwards in multiple components by trusted bank staff.

We looked at one such device, which dated from the late 1980's, and found that the master key values were almost intact on power-up. The number of bits in error was of the order of 5-10%. We cannot give more accurate figures as we were not permitted to copy either the correct master key values, nor the almost-correct values that had been 'burned in' to the static RAM chips. We are also not permitted to name the bank at which these modules are installed, and do not consider it prudent to name their manufacturer.

This level of memory remanence would be alarming enough. However, it has a particularly pernicious and noteworthy interaction with DES key parity in this common application.

If each DES key is wrong by five bits, then the effort involved in searching for the 10 wrong bits in a double DES key might be thought to be 112-choose-10 operations. Each operation would involve (a) doing a 2-key 3DES decryption of a 56 bit PIN key whose enciphered value is, as we noted, widely known (b) in the 2^{-8} of cases where this result has odd parity, enciphering an account number with this as a DES key to see if the (decimalised) result is the right PIN. The effort is about 3 times 112-choose-10 DES operations — say 2^{50}. But it would probably be cheaper to do a hardware keysearch on the PIN key directly than to try to implement this more complex 2^{50} search in either hardware or software.

However, the bytewise nature of the DES key redundancy reduces the effort by several orders of magnitude. If no key byte has a double error, then the effort is seven tries for each even parity byte observed, or 3 times 7^{10} — about 2^{30}, which is easy. If there is one key byte with a double error, the effort is 2^{38}, giving a search of 2^{40} DES operations — which is feasible for a class I attacker.

This is not the first instance of DES parity being a hindrance rather than a help. In one case, the Kerberos partity-checking DES implementation was grafted into an encrypting telnet implementation that derived its key material from a Diffie-Hellman exchange. As the Diffie-Hellman key bits were random, only 1 in 256 exchanges resulted in a legal key; in all other cases, key loading failed and (as the implementation didn't check the return code) the session would continue using an uninitialised key [8]. In another, a misunderstanding led to the PIN key used by a number of banks being chosen as a password made up of ASCII characters with odd parity; and in other applications, it is common for an ASCII password to have parity set and be used as a key. The problem here is that ascii characters have a zero in the high order bit, while DES parity operates on the low order bit; so the key diversity is less than 2^{48}, and in fact is even less than the entropy of the chosen passwords [13].

5 Protocol Failure

Poorly designed protocols are a more common source of attacks than many people recognise [3]. Many of them also require only very simple and cheap equipment to exploit.

For example, satellite TV decoders typically have a hardware cryptoprocessor that deciphers the video signal, and a microcontroller which passes messages between the cryptoprocessor and the customer smartcard that contains the key material. If a customer stops paying his subscription, the system typically sends a message over the air which instructs the decoder to disable the card. In the 'Kentucky Fried Chip' hack, the microcontroller was replaced with one which blocked this particular message [3].

Another example is given in [2], which describes an attack on the Dallas Semiconductor DS5002FP secure microcontroller. This attack utilises a protocol failure to circumvent the encryption system used to protect off-chip memory.

Some protocol failures require no equipment at all to exploit, and one example that has come to our attention arose from a modification made to a bank security module.

One bank was upgrading its systems and wished to change the format of its customer account numbers. Changing the numbers meant that the 'natural PIN' calculated from the account number would change. But the bank did not wish to inconvenience its customers by forcing new PINs on them; so it decided to calculate suitable offsets so that the customer PIN would be unchanged from one card generation to the next.

The security module as supplied did not support such a transaction (for reasons that will shortly become clear). The manufacturer was duly contacted and asked to provide it; modified software was duly supplied, but with a warning that this should only be used for a batch run to calculate the necessary offsets, and then discarded, as it was dangerous. But the nature of the danger was not spelled out, and due to personnel changes and project delays the card number change could not be carried out at once. The effect was that the modified software was installed and left in place.

About a year later, one of the bank's programmers noticed a simple attack. The additional transaction had the syntax: 'given an initial account number of X and offset of Y, calculate an offset which will enable this PIN to be used on account number Z'. The programmer could input the account number and offset of a target as X and Y (the majority of offsets were zero in any case) and his own account number as Z. The returned value enabled him to trivially calculate the target's PIN.

Fortunately for the bank, the programmer brought this vulnerability to the attention of authority, rather than exploiting it.

6 Conclusions

We have improved on Differential Fault Analysis. Rather than needing about 200 faulty ciphertexts to recover a DES key, we need between one and ten. We can factor RSA moduli with a single faulty ciphertext. We can also reverse engineer completely unknown algorithms; this appears to be faster than Biham and Shamir's approach in the case of DES, and is particularly easy with algorithms that have a compact software implementation such as RC5. Unlike some previous work, our attacks use a realistic fault model, which has actually been implemented and can be used against fielded systems. The critical idea is to cause errors in code rather than in data.

We have also shown how low cost and commonly available laboratory equipment, such as microprobes and laser cutter microscopes, can be used to implement chip-surface attacks to recover key material from supposedly secure processors. Key redundancy, such as the key parity of DES, can be used to facilitate such attacks. This work reinforces the lesson from [9] — that key redundancy requires more careful consideration than has usually been accorded it in the past.

An example of this is that the particular form of key redundancy used in DES can interact quite lethally with the memory remanence properties of SRAM chips commonly used in banking security modules. The effect is that master key material can fairly easily be recovered from a discarded security module. Such devices should as a matter of policy be destroyed carefully.

Finally, in addition to attacks involving non-obvious interactions of protocol features with hardware features, there are many cases in which physical protection can be circumvented by pure protocol attacks. Tamper resistant devices are not only much harder to build than many people realise; they are also much harder to program and to use.

Acknowledgements

Mike Roe pointed out that the glitch attack on RSA can be done in real time by a Mafia owned point-of-sale terminal; Stefan Lucks the way to linearise DES.

References

1. DG Abraham, GM Dolan, GP Double, JV Stevens, "Transaction Security System", in *IBM Systems Journal* v 30 no 2 (1991) pp 206–229
2. RJ Anderson, MG Kuhn, "Tamper Resistance — a Cautionary Note", in *The Second USENIX Workshop on Electronic Commerce Proceedings* (Nov 1996) pp 1–11
3. RJ Anderson, RM Needham, "Programming Satan's Computer", in *'Computer Science Today'*, Springer Lecture Notes in Computer Science v 1000 pp 426–441
4. RJ Anderson, "Why Cryptosystems Fail", in *Proceedings of the 1st ACM Conference on Computer and Communications Security* (November 1993) pp 215–227
5. E Biham, A Shamir, "A New Cryptanalytic Attack on DES", preprint, 18/10/96
6. E Biham, A Shamir, "Differential Fault Analysis: Identifying the Structure of Unknown Ciphers Sealed in Tamper-Proof Devices", preprint, 10/11/96
7. E Biham, A Shamir, "Differential Fault Analysis: A New Cryptanalytic Attack on Secret Key Cryptosystems", preprint, 21/11/96
8. M Blaze, *personal communication*
9. M Blaze, "Protocol Failure in the Escrowed Encryption Standard", in *Proceedings of the 2nd ACM Conference on Computer and Communications Security* (2–4 November 1994), ACM Press, pp 59–67
10. F Bao, RH Deng, Y Han, A Jeng, AD Nirasimhalu, T Ngair, "Breaking Public Key Cryptosystems in the Presence of Transient Faults", *this volume*
11. D Boneh, RA DeMillo, RJ Lipton, "On the Importance of Checking Computations", preprint, 11/96
12. E Bovenlander, invited talk on smartcard security, Eurocrypt 97
13. P Farrell, *personal communication*
14. L Guillou, *comment from the floor of Crypto 96*
15. P Gutman, "Secure Deletion of Data from Magnetic and Solid-State Memory", in *Sixth USENIX Security Symposium Proceedings* (July 1996) pp 77–89
16. M Joye, F Koeune, JJ Quisquater, "Further results on Chinese remaindering", Université Catholique de Louvain Technical Report CG-1997-1, available at `<http://www.dice.ucl.ac.be/crypto/tech_reports/CG1997_1.ps.gz>`
17. O Kocar, "Hardwaresicherheit von Mikrochips in Chipkarten", in *Datenschutz und Datensicherheit* v 20 no 7 (July 96) pp 421–424
18. C Mitchell, S Murphy, F Piper, P Wild, "Red Pike — An Assessment", Codes and Ciphers Ltd 2/10/96
19. RL Rivest, "The RC5 Encryption Algorithm", in *Proceedings of the Second International Workshop on Fast Software Encryption* (December 1994), Springer LNCS v 1008 pp 86–96
20. *'VISA Security Module Operations Manual'*, VISA, 1986