Virtual Analysis and Reduction of Side-Channel Vulnerabilities of Smartcards

Jerry den Harog (hartogji@cs.utwente.nl)
Department of Computer Science, Universiteit Twente

Erik de Vink (evink@win.tue.nl)
Department of Mathematics and Computer Science, TU Eindhoven
Leiden Institute of Advanced Computer Science, Leiden University

Abstract. This paper focuses on the usability of the PINPAS tool. The PINPAS tool is an instruction-level interpreter for smartcard assembler languages, augmented with facilities to study side-channel vulnerabilities. The tool can simulate side-channel leakage and has a suite of utilities to analyze this. The usage of the tool, for the analysis of a cryptographic algorithm is illustrated using the standard AES and RSA. Vulnerabilities of the implementations are identified and protective measures added. It is argued, that the tool can be instrumental for the design and realization of secure smartcard implementations in a systematic way.

Keywords. smartcard, side-channel attack, power analysis, fault analysis, DPA, simulation, countermeasures, systematic hardening

1. Introduction

Since the ground-breaking papers of Paul Kocher [16, 17] a vast amount of research on power analysis and other side-channel attacks have been reported in the literature (e.g. [4, 18, 6, 20]). Taking all types of attacks into account for a concrete implementation of a cryptographic algorithm on a specific smartcard and, furthermore, investigating the feasibility of all appropriate countermeasure is a daunting task. Apart from experience and engineering principles, feedback on intermediate stages during the development —instead of after the implementation phase— is valuable to the smartcard algorithm programmer. The PINPAS tool, discussed here, can be used to provide this information. More generally, the tool aims to contribute to a better understanding of side-channel vulnerability and systematic hardening of a smartcard application.

PINPAS is an instruction-level interpreter for smartcard assembler together with facilities to collect and process side-channel information. In this paper we focus on the usability aspects of the tool when assessing the side-channel vulnerability of an implementation of a cryptographic algorithm. As running examples, we investigate the vulnerability for differential power analysis (DPA) of the AES and RSA algorithm.

We show, for basic smartcard implementations, how attacks and the effectiveness of subsequent countermeasures can be analyzed.

In outline, the approach works as follows: First, a candidate algorithm is implemented in the assembler language of the targeted smartcard. Then, power traces are collected by the PINPAS tool while interpreting the implementation. Next, iteratively, the feasibility of a DPA attack is estimated and necessary countermeasures are put in place.

The PINPAS tool is still only a prototype. However, it clearly shows the advantage of software-based feedback for side-channel attacks on smartcards. The benefits of the tool include the following:

- Side-channel analysis can already start early in the development process as no physical card or laboratory setup is needed.
- Time consuming measurements and signal-analysis can largely be avoided.
- It is easy to switch platform and card type and change the attack parameters such as the side-channel, attack point, etc.
- The tool provides quick feedback on the effectiveness of introduced countermeasures.
- The simulation environment enables one to balance resource utilization, performance and security, respectively.
- The flexible setup allows for easy experimentation with new potential attack or defense techniques. Additionally, the PINPAS tool provides a setting to study hypothetical side-channel leakage.

The idea of performing timing attacks and differential power analysis on a simulator is a clear one and may not be completely new. However, the PINPAS tool is, to our best knowledge, the first tool reported in the academic literature. Moreover, the increase of computing power and maturity of object-oriented programming languages have witnessed further evolution since the advent of SPA and DPA. Today, an industrial-sized tool based on PINPAS has come in reach. In fact, the prototype has been adopted for further development.

Because of its broad applicability in early development stages, its flexibility and ease of use, while avoiding involved physical measurements, the PINPAS tool can be valuable to smartcard manufacturers, algorithm designers, evaluators as well as researchers. The PINPAS tool provides an instrument to study information leakage via side-channels in a systematic sway. With the tool, countermeasures can be evaluated for their effectiveness. This way, the prototype may serve as a stepping stone toward a general theory of generating safe code from vulnerable algorithms. An overview of the PINPAS tool was reported in [8].

Acknowledgments This paper kindly acknowledges the student work conducted by Wouter Burgers and Gijs Hollestelle reported in [15] on which this work is based.

2. The PINPAS tool

The PINPAS tool is a Java program that provides a number of virtual machines for various smartcard assembler languages. On top of this, the tool facilitates various type of side-channel attacks, both first and higher-order. (The former type focuses on side-channel information from one single source; the latter seeks to combine such information retrieved from several places.) Given a particular implementation code of a cryptographic algorithm the tool provides an environment to assess the vulnerability of the implementation, in particular for timing attacks and for power analysis. Thus, the PINPAS tool consists of two main building blocks: a simulator and an analyzer.

The simulator part for a particular brand of smartcard is based on the overall architecture and the instruction set of the smartcard. If one wishes to do so, one can restrict the so-called power profile to the modeling of the CPU and storage, neglecting possible co-processors or dedicated components. Typically, a number of registers and memory space is allocated by the simulator, comprising –together with the program counter– its state. Instructions then are interpreted as state transformations. They affect the program counter, the registers and memory. For each instruction a Java method is defined in the class implementing the simulation of the smartcard. Interpretation of a program for the smartcard consists of a loop that repeatedly calls the Java method that corresponds to the card instruction to be executed. As an aside, note that system calls and native APIs need not to be represented as low level operations; Java methods that reflect the input/output behavior of such card components suffice. This way, the simulator abstracts away from aspects that are not directly related to the cryptographic algorithm itself.

The analyzer part supports the collection of abstract power traces. A physical model of the smartcard architecture and the instruction set are parameters to the tool. A rather coarse approach, very much like common practice in actual power attacks to smartcards though, only takes the Hamming weight of data into account. On the other side of the spectrum, one can generate sampling information in a format compatible to that of the oscilloscope. Although the tool-generated power traces cannot, by their nature, be exactly the same of those generated physically, experiments have shown a clear similarity between the two. While obtaining selected or randomly generated input values from file one-by-one, the tool produces, by running the algorithm on the virtual machine, as many abstract power traces as desired.

In addition, the tool provides a few built-in mechanisms for further manipulation. In particular the DPA selection criteria or brute-force byte attacks can be launched automatically. Note that, power traces are produced directly from the assembler code, thus bypassing time and effort needed to collect power traces physically from the algorithm loaded onto a smartcard. Also, post processing of power traces to eliminate noise and jitter has become a non-issue in this setting; by construction, the traces are noise-free and lined up. However, in principle, physically collected power traces can be presented to the analyzer part of the tool as well.

3. A case study: AES and RSA

In the previous section we described the tool developed in the PINPAS project. In this section, we demonstrate the steps in the development of a secure smartcard implementation of a cryptographic algorithm. In particular, we will create, in several steps, implementations of the AES and RSA algorithms for a typical smartcard. For simplicity we have chosen a generic platform based on the Hitachi H8 RISC-processor.

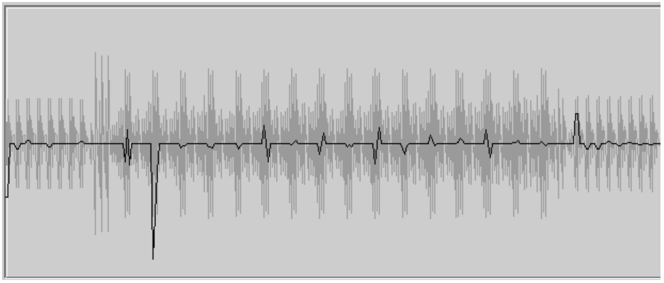
3.1. AES and the PINPAS tool

The advanced encryption standard AES algorithm [19, 7] is a block cipher which uses a number of rounds consisting of reversible linear 'confusion' and non-linear 'diffusion' transformations. The linear transformations exploit exclusive ors with key material and AES's typical shifting and moving of rows and columns. The non-linear transformation includes a manipulation of bytes by S-boxes.

Starting point for the smartcard implementation discussed here, was a C-coded version of the AES algorithm. The C-code was optimized with the efficiency of the resulting machine code in mind. Next, the `gcc` compiler was run to generate Hitachi H8 code from this. In the discussion below we will concentrate on the case of an 128-bit key length.

The analysis of the smartcard implementation starts off with some standard procedures that are provided by the tool. A preliminary power analysis shows that the different rounds can be distinguished from the power signature. Also other information, such as the use of specific input bytes, can be revealed.

Partial power trace for AES showing 6 complete rounds



Zoomed view of the usage of second input byte and the power trace (gray)

Starting from the assumption that Hamming weights of data may leak, an attacker can focus on the first linear transformation using key material, a so-called `addroundkey`. The crucial instruction in this code fragment is

$$\texttt{data[i] XOR key[i]} \tag{1}$$

which, obviously, xors two bytes. In the first round the data value is known: It is simply the input to the algorithm. Thus, guessing the `i`th byte of the key will allow one to predict the outcome of the xor. Having such a criterion available a power attack can be launched.

In the tool this criterion, or trace condition, – forming two groups of traces with Hamming weights larger and smaller than 4, respectively – can be reflected by the following piece of Java code.

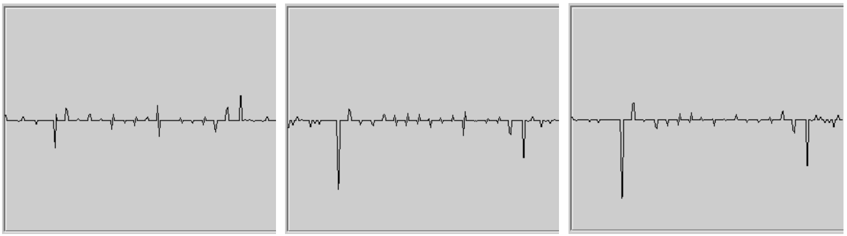
```
boolean select( Trace trace )
{  wgt = hammingWgt( trace.getInputByte( i ) ^ keyguess );
   if ( wgt > 4 ) return true;
   if ( wgt < 4 ) return false;
   throw new UnsortableException();
}
```

The exception `UnsortableException();` is used to discard traces, as is done in this example with traces for which the Hamming weight `wgt` equals 4. (Other, more advanced trace conditions can be used as well, but for the purpose here, the above will do.)

The tool will divide the power traces in two groups according to the trace condition. The average power consumption over these two groups is compared, resulting in a so-called difference trace. If the key byte is guessed correctly, a difference in average power consumption will occur at the position of the instruction in equation (1) and result in a peak in the difference trace. For an incorrect guess, a random partition of the traces without significant peaks in the difference trace is expected.

To facilitate a quick and automatic attack, the PINPAS tool was directed to record the power traces only for a small window including the instruction to be analyzed. Apart from speeding up the simulation, such facilities are also convenient when memory usage becomes a bottleneck.

An experiment using 250 traces discovered all but a few bits of the 128 bit round key. Because of the linearity of the attacked xor instruction, a relatively high spike in the difference trace will already occur when 6 or more bits of the key byte are guessed correctly. Therefore, there is a small probability that a false positive might be given while iteratively stepping through all possible key byte values. This probability can be reduced by using more traces. The same experiment based on 500 traces predicted all key bytes without error.



Difference trace for an incorrect, almost correct and correct guess

Several countermeasures have been proposed to protect against the power analysis attacks, including hiding of data by random masking [13] or distribution of critical data over different locations [5, 14]. Other countermeasures seek to make the aligning traces and the averaging of power consumption more difficult. In particular, 'no-op' insertion introducing random waits and randomization of control flow. Of course, due to the limited resources, one needs to carefully balance the improvement of security yielded by these techniques and the resource utilization and performance of the resulting algorithm.

In order to defend the AES implementation described above, we introduce a countermeasure that randomizes the flow of control. The observation is that in the exclusive or with bytes of the key in the `addroundkey` section can be done in any order. The adapted code is as follows:

```
int[] order = permutation of 0..15
data[ order[i] ] = data[ order[i] ] XOR key[ order[i] ]
```

Experiments with the PINPAS tool show that the countermeasure is effective. Repeating the DPA based on 500 traces no longer revealed the key. We expected that simply more traces would suffice for a successful DPA attack. Surprisingly, using 8000 traces (with 8000 traces being 16 times 500 traces) did not prove successful either. Thus, the tool helped not only in providing experimental evidence of the strength of the countermeasure in reducing the vulnerability, but also provided feedback on our engineering intuition. Recall that in the setting of the tool the synthesized traces are noise-free; in physical experiments even more traces need to be collected. However, it should be noted that other DPA attacks are still possible, e.g. in the `subbytes` or `shiftrows` routines.

3.2. RSA and the PINPAS tool

As a second example we discuss the familiar RSA algorithm [22]. Encryption in the RSA algorithm uses exponentiation modulo a large composite n. For our implementation of RSA we assume to have a co-processor available on the card. The co-processor is used for two operations: squaring and multiplication in $\mathbb{Z}/n\mathbb{Z}$. We focus on the common case of 512 bit numbers.

Using the operations provided by the co-processor, exponentiation can be implemented as follows.

```
r = 1;
for( int i = 0; i < 512; i++ )
{ r = SQR( r, n );
  if ( keybit(i) == 1 ) r = MUL( r, m, n );
}
```

An SPA attack on this code is well-known, see e.g. [16, 9]. The multiplication `MUL` in the code above is done only if the current bit of the key equals 1. By inspection of the shape of the power consumption trace, one can distinguish the different rounds in the algorithm. Moreover, one can determine for a round, by examining its length, whether it consists of only a squaring or also includes a multiplication. This way, the key can directly be read from the power trace.

The standard countermeasure to remedy this vulnerability is to introduce a dummy multiplication into the code above for key bits which are zero.

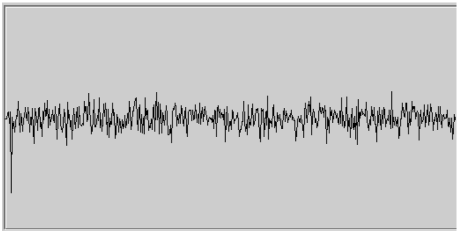
```
if ( keybit(i) == 1 ) r = MUL( r, m, n );
else dummy = MUL( r, m, n );
```

As all rounds are then of equal length, one can no longer decide the value of a key byte by simply looking at the power consumption trace. Unfortunately, as a side-effect of this defense, the vulnerability to DPA attacks has been increased.

In the RSA algorithm the intermediate result for the variable `r` depends on part of the key only. A DPA attack can be launched attacking this value. By guessing e.g. the first byte of the key, the intermediate value after 8 rounds can be predicted for given inputs. As before, a spike in the difference trace indicates a correct guess for the key.

In contrast to the attack on the AES discussed above, the attack on RSA has a cumulative nature. In order to find a byte of the key, all more significant bytes should already have been found. If no byte value results in a significant peak it is likely that one of the previous byte values has been guessed incorrectly.

Experiments with the tool showed that the correct key could be obtained using 125 traces. A complication in the attack is that significant peaks are induced by a wrong guess, such as a shifted key value. However, one can eliminate these peaks by narrowing the window of analysis to the correct round, for example the 8th round for the first byte (MSB) of the key. The availability of the tool helped unraveling the occurrence of peaks at unexpected locations in the difference traces.



The correct key guess for the first byte of the key[1]

Due to the mathematical setting of RSA several countermeasures available for block ciphers, such as (non-multiplicative) masking and distribution, do not apply here. Perhaps they are applicable for a completely

[1] The tool has been instructed to only record power consumption values at points relevant to the attack.

different implementation of RSA, but it is hard to see how the co-processor can then be exploited. In the same vein, although random waits can be introduced, it seems likely that in practice, the attacker can recognize and remove these from the power trace. When simulating traces in the tool this is straightforward.

Experiments with the tool indicate that hardware defenses are effective. For example, adding to the co-processor an internal buffer that does not leak for storage of the intermediate result can help reduce the DPA sensitivity. Countermeasures in hardware can be incorporated in the tool, relatively easy, by adapting the Java classes implementing the various machine instructions and adjustment of the power profile. As seen from our discussion above, this co-processor must also implement a dummy multiplication operation which yields the same power signature as a real multiplication, but does not update the internal buffer.

4. Significance of theoretical attacks

In the previous section we have demonstrated how attacks on AES and RSA can be simulated in the PINPAS tool. Here we discuss the practical relevance of the results found using the tool.

4.1. Measured vs. synthesized traces

Power traces generated by the simulation cannot, by their nature, be the same as physically measured traces. However, a reasonable approximation is possible. So, what is a reasonable approximation? From a security standpoint the amount of information leaked in the power consumption is more relevant than, for example, the absolute difference with actually measured values.

To do a correct analysis using the tool it is important that attacks possible on the physical card can also be mimicked in the tool. This means that information leakage in the power consumption trace of a physical card should also be present in the simulated power trace. On the other hand, adding countermeasures cost money, time, performance, etc. Thus, from a practical stand point, having too much information in a simulated power trace is also not too attractive as this may cause a waste of effort defending against attacks which are not really possible anyway.

The power traces synthesized by the tool contain no noise, in contrast to physically measured traces. Experiments, in which random noise has been added to the synthesized power traces, have confirmed that this does not affect the vulnerability to a DPA attack. The absence of noise has several clear advantages in the analysis. One can concentrate on the actual vulnerabilities in the code without a need for advanced signal processing to align traces and reduce noise and jitter. Also, because less traces are required for an attack, the amount of computation is limited. Note that, when testing only with traces obtained from physical measurements one may miss a vulnerability merely due to the fact that an insufficiently many traces were used.

In comparing generated traces versus power traces with noise obtained from physical measurement we observe the following: First, if no attack is found on the generated traces, there will be none on the measure power traces either. Second, if an attack was found, one will likely be able to reconstruct this for measured power traces, possibly requiring more measurements. The absence of a vulnerability for measured traces may very well be due to the signal-to-noise ratio. By using generated traces one can focus on the intrinsic lack of safety of an implementation and devote time to finding the proper countermeasures.

4.2. Hamming weight and other side-channel information

In the simulations used for the power analysis attacks in the previous section, the Hamming weight of manipulated data leaks in the power trace. Different cards may leak different information in their power consumption. The PINPAS tool is rather flexible in the power profile that is used, allowing the simulation to be adapted to the card under consideration. Experience with smartcard hardware has shown that the leakage of information in the power consumption is, in large part, due to activity on the memory bus. Depending on the implementation of the memory bus, the power consumption of the memory bus may be related to changes of the value on the bus rather than to the actual value on the bus. This means that in many cases assuming that the Hamming weight leaks in the power consumption is probably an overestimation of the amount of information that an attacker can obtain through power analysis. A power profile that captures the more restricted information leakage has also been included in the tool.

As with noiseless traces, using Hamming weight leakage is playing it safe: If no attacks are found with this strong form of leakage, it is unlikely that there will be attacks on measured traces. It does mean, however, that the practicality of attacks found in the tool needs to be considered. In other words, if an attack is found, does one need to spend considerable effort to prevent it or is it merely a theoretical attack? To answer this question, the attack can be repeated on a different power consumption profile which contains less information leakage, all the way down to the actual hardware if needed (and a card is available). Having predefined and tested the attacks and potential weak points expedites the testing process on the hardware.

The attack on the AES in the previous section relies in an essential way on leakage of Hamming weights. This means that this attack does work directly on cards with a more restricted form of information leakage. The attack may still provide useful information if the attacker can determine part of the value on the bus before the attacked instruction. Intimate knowledge of the smartcard architecture and ad-hoc analysis of the smartcard, along with heuristics, can be used to obtain the previous value on the bus. If one can safely assume, that the attacker is not able to obtain this information, using a power profile with less information leakage is appropriate. Tests in the tool using such a power profile confirmed, that the described attack no longer works in this setting. However, other attacks on AES can still be mounted. For example, by guessing one byte of the key, the value of the output of the first non-linear transformation (S-box) can be predicted. By looking at only a single bit of this predicted value the attack will still work using the adapted power profile. Note that, a defense using randomization of control flow, such as introduced in the previous section, can also be used to help to protect against this attack.

The SPA attack on RSA using timing of rounds only uses the general shape of the power graph and easily translates to an attack on physical traces. The DPA attack on the version of RSA strengthened against the timing attack also used the fact that Hamming weights leak, but not in an essential way; instead of predicting a byte of the intermediate value, a single bit could be used. This attack requires more traces, but will still work with the power profile with less information leakage.

In conclusion, one can use a liberal leakage profile for an initial analysis of an implementation. If vulnerabilities are found either countermeasures can be introduced or attacks can be re-tested with a more restrictive power profile. In this way, one can make precise which assumptions about the power consumption are essential for the safety of the particular implementation.

5. Concluding remarks

Above we have illustrated the usability of the PINPAS tool for the case of AES and RSA. It was explained how implementations of these two algorithms can be run on the tool and side-channel vulnerabilities can be identified. Next it was discussed how the effectiveness of the countermeasures can be assessed. Finally the relevance of attacks on synthesized traces for the security of a smartcard was treated.

We have argued, that it is advantageous to have a software tool for the analysis of side-channel vulnerabilities available. The discussed attacks are relatively straightforward. However, the approach is flexible and suited to examine, e.g., higher-order strategies. For the particular case of the strengthened AES implementation discussed above, summing up the power consumption at all locations in the power trace where the particular instruction can be loaded in the randomized execution will reveal the key using the same number of traces. For RSA the results for multiple bits or bytes can be combined to strengthen the correlation and amplifying the spikes. It should be noted, that such refined attack scenarios are also necessary in order to keep up with the developments of unconventionally wired smartcard (glue logic) and associated a-typical power profile. More generally, the tool can be deployed for higher-order scenarios on implementations using masking or duplication [5, 14, 1] to test, e.g., the increase in the number of power traces needed. Currently, the points of attack are selected manually, but it would be an interesting add-on to the tool to do this automated (most likely, through integration with other tools for security analysis, such as dependency checkers and high-level program verifiers). A long-term goal of the PINPAS project is the construction of a code optimizer that eliminates side-channel vulnerabilities automatically. For the development of a theory of secure program transformations, building on the notion of non-interference (see, e.g., [12, 23, 21]), the experiments performed with the tool play a crucial role.

In the previous sections we mainly focused on power analysis. However, also for other flavors of side-channel attacks the PINPAS tool can be exploited fruitfully. For example, successful timing attacks have been conducted. Also, the general idea of injecting hardware faults into the smartcards execution [4, 3] can easily be mimicked in the tool, much more precisely and practically as compared to a physical set-up. The virtual machine can be instructed to temporarily step outside its main interpretation cycle and perform a state transformation according to a faulty instruction (as physically could have been enforced by a power glitch, light flash, card tear or other means of card stressing). Due to the modular architecture of the tool this can be implemented straightforwardly, e.g. along the lines of [11] for AES and [4] for RSA. Thus, also for fault analysis, the clean setting of the tool helps to avoid time-consuming and labourious experiments (with, occasionally, annihilating consequences for the smartcard used) while still obtaining a clear indication of the particular vulnerabilities of the algorithm on the card.

Recent work [25, 24] reports on enhancing crypto-analytical collision attacks [10, 2] with side-channel information resulting in successful attacks on DES and AES. The crux in this hybrid approach is that high correlation of power traces indicate that a collision has occurred. Practical experiments support the view that collision attacks constitute a powerful technique, that is insensitive to various countermeasures. It turns out that relatively few power traces are needed. However, having total control, as in the software setting of the PINPAS tool advocated in this paper, may be advantageous in collecting further experimental evidence for the reach of collision attacks.

References

1. M.-L. Akkar and L. Goubin. A generic protection against high-order differential power analysis. In T. Johansson, editor, *Proc. FSE 2003*, pages 192–205. LNCS 2887, 2003.
2. E. Biham. How to decrypt or even substitute DES-encrypted messages in 2^{28} steps. *Information Processing Letters*, 84:117–124, 2002.
3. E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In B.S. Kaliski Jr., editor, *Proc. Crypto'97*, pages 513–525. LNCS 1294, 1997.
4. D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Furny, editor, *Proc. EuroCrypt'97:*, pages 37–51. LNCS 1233, 1997.
5. S. Chari, C.S. Jutla, J.R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. Wiener, editor, *Proc. Crypto'99*, pages 398–412. LNCS 1666, 1999.
6. J.-S. Coron, P. Kocher, and D. Naccache. Statistics and secret leakage. In Y. Frankel, editor, *Proc. Financial Cryptography 2001*, pages 157–173. LNCS 1962, 2001.
7. J. Daemen and V. Rijmen. *The design of Rijndael*. Springer Series on Information Security and Cryptography. Springer, 2002.
8. J. den Hartog, J. Verschuren, E. de Vink, J. de Vos, and W. Wiersma. PINPAS: a tool for power analysis of smartcards. In D. Gritzalis, P. Samarati, and S. Katsikas, editors, *Proc. SEC 2003*, page 5pp. Wolters-Kluwer, 2003. IFIP WG 11.2 Small Systems Security.
9. J.-F. Dhem, P.-A. Leroux F. Koeume, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In J.-J. Quisquater and B. Schneier, editors, *Proc. Smart Card Research and Applications*, pages 167–182. LNCS 1820, 1998.
10. H. Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11:253–271, 1998.
11. P. Dusart, G. Letourneux, and O. Vivolo. Differential fault analysis on AES. Technical Report 2003–01, LACO, Université de Limoges, 2003.
12. J. A. Goguen and J. Meseguer. Security policy and security models. In *Proc. of the 1982 Symposium on Security and Privacy*, pages 11–20, Oakland, 1982. IEEE.
13. J.D. Golić and C. Tymen. Multiplicative masking and power analysis of AES. In B.S. Kaliski Jr, C.K. Koç, and C. Paar, editors, *Proc. CHES 2002*, pages 198–212. LNCS 2523, 2003.

14. L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In C.K. Koç and D. Naccache, editors, *Proc. CHES'99*, pages 158–172. LNCS 1717, 1999.
15. G. Hollestelle, W. Burgers, and J. den Hartog. Power analysis on smartcard algorithms using simulation. Technical Report CSR 04–22, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, 2004. 40pp.
16. P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Proc. CRYPTO'96*, pages 104–113, 1996.
17. P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M.J. Wiener, editor, *Proc. CRYPTO'99*, pages 388–397. LNCS 1666, 1999.
18. T.S. Messerges. *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms*. PhD thesis, University of Illinois, Chicago, 2000.
19. NIST. *FIPS 197: Announcing the Advanced Encryption Standard*. NIST, 2001.
20. J.-J. Quisquater and D. Samyde. Electro-magnetic analysis (EMA) measures and counter-measures for smart cards. In I. Attali and T. Jensen, editors, *Proc. E-Smart Card Programming and Security*, pages 200–210. LNCS 2140, 2001.
21. F. Martinelli R. Focardi, R. Gorrieri. Non-interference for the analysis of cryptographic protocols. In U. Montanari, J.D.P. Rolim, and E. Welzl, editors, *Proc. ICALP 2000*, pages 354–372. LNCS 1853, 2000.
22. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communication of the ACM*, 21:120–126, 1978.
23. P.Y.A. Ryan and S.A. Schneider. Process algebra and non-interference. *Journal of Computer Security*, 9:75–103, 2001.
24. K. Schramm, G. Leander, P. Felke, and C. Paar. A collision-attack on AES combining sidechannel- and differential attacks. In *Proc. CHES 2004*. LNCS, 2004.
25. K. Schramm, T. Wollinger, and C. Paar. A new class of collision attacks and its application to DES. In *Proc. FSE 2003*, pages 206–222. LNCS 2887, 2003.