PINPAS: A TOOL FOR POWER ANALYSIS OF SMARTCARDS

J. den Hartog[1], J. Verschuren[2], E. de Vink[3,4], J. de Vos[2], W. Wiersma[3]

Abstract This paper describes the PINPAS tool, a tool for the simulation of power analysis and other side-channel attacks on smartcards. The PINPAS tool supports the testing of algorithms for vulnerability to SPA, DPA, etc. at the software level. Exploitation of the PINPAS tool allows for the identification of weaknesses in the implementation in an early stage of development. A toy algorithm is discussed to illustrate the usage of the tool.

Keywords: Smartcard, Power Analysis, Side-Channel Analysis, Simulation

Introduction

Developers of smartcard algorithms are confronted, among others, with the question of the vulnerability of their final implementation to power analysis attacks (Kocher, 1996; Fahn and Pearson, 1999; Messerges, 2000). Typically, in a late stage of the development of the smartcard application some evaluating company is consulted. Over a period of several weeks this specialized company assesses the security level of the smartcard. In case a power analysis attack is successfully launched against the smartcard, re-design and patches should improve the actual code. Time-to-market and project deadlines seem not to allow for a different approach.

Central to the situation, in our view, is that power analysis vulnerability is tested on a physical smartcard in a late phase. It is possible to do this inspection earlier. The key observation is that the implementation is not needed in hardware to find an attack. For power analysis it is sufficient that the algorithm is available in software.

[1]Corresponding author, Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands, jhartog@win.tue.nl
[2]TNO-TPD, Delft, The Netherlands
[3]Eindhoven University of Technology, Eindhoven, The Netherlands
[4]LIACS, Leiden, The Netherlands

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35691-4_52

D. Gritzalis et al. (eds.), *Security and Privacy in the Age of Uncertainty*

An instruction level interpreter for the machine language of the smartcard and a physical model of the smartcard will do. The power consumption of an instruction with its data can be represented by a power consumption frame. Then, interpretation of a program leads to a trace of power consumption frames. The trace is in fact an abstraction of the power trace obtained when measuring the power consumption of a smartcard running the program with an oscilloscope.

The set-up sketched above brings several advantages compared to the physical approach: (i) Power analysis can be done in an early stage of the development of the application, say on the code of a skeleton version of the program. (ii) Immediate feedback can be given on the vulnerability to power attacks as time-consuming measurements by an evaluator can be avoided. (iii) As turn-around time has decreased significantly, iterative improvements of the code are feasible and the power analysis can be explored at leisure of the developer.

1. The PINPAS tool

Above we described a scheme for evaluating the vulnerability of a smartcard to power analysis attacks. The PINPAS (Program INferred Power Analysis in Software) tool is being developed to implement this scheme. Currently the tool consists of two major components: a simulator and an analyzer.

One part of the tool provides a simulator for programs to be run on a given smartcard. The simulation not only calculates the input/output behavior of a program, but also models the side-channel information that becomes available during the execution of the program. From this additional information the power consumption frames of the abstract power traces, as mentioned above, are generated. By incorporating only the information that leaks for the particular card, the abstract power trace is an accurate representation of a trace that would otherwise have been obtained physically. The simulator will in general deliver perfect traces without any sort of jitter or noise, although this is not essential in case of a sufficient signal-to-noise ratio; if noise is present more traces may be required but the same attacks can still be mounted.

The other main part of the tool supports the power analysis of the generated traces. Due to the modular software architecture it does not matter which program and simulator were used to generate these traces; only the traces have to be supplied to this part of the tool. In later sections we explain how the PINPAS tool can be used on a toy algorithm and leakage of Hamming weights over the bus. However, the tool has been successfully applied in experiments including DES, TEA and AES.

2. A simple example

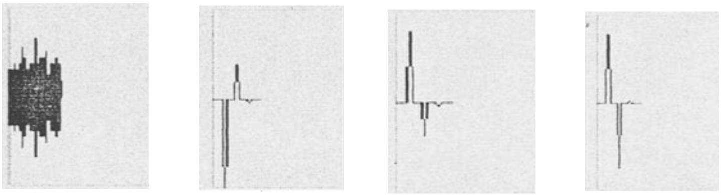
To illustrate the PINPAS tool at work we discuss a toy algorithm, viz. `return Sbox[ input XOR key ]`. The algorithm takes the exclusive or of an 8-bit input and an 8-bit key and calculates their 4-bit signature. The function `Sbox[ x ]` selects one out of four S-boxes based on the first two bits of `x` and uses the remaining six bits as the usual S-box input. Further details of the S-boxes are omitted. Clearly this algorithm, as such, is not secure, but it handles the main ingredients of an algorithm like DES. As platform we assume some vanilla smartcard using some generic processor. The smartcard is assumed to leak information about values transported on the memory bus.

As a first step we translate the algorithm to the assembler code for our smartcard. A straightforward implementation is

```
MOV x,input; XOR x,key; MOV output,Sbox[x]; EXT
```

Here we have used the operations `MOV x,y` to move data to `x` from y, `XOR` for the exclusive or and storage of the result in the first operand, and `EXT` to end the program by returning the value `output`. The implementation of the program can be loaded into the simulator and run to test its functionality.

The next step is to evaluate the vulnerability to power analysis of the assembler code. A potential source for DPA is the value `input XOR key`, which can be calculated from the known value of the input and a guessed value of the key. After generating power traces with the simulator, the traces can be split into two groups using the 'trace condition' for the expected power consumption of `XOR x,key`. A difference trace is obtained by subtracting the average traces for each of the two groups. This is done for each possible value of the key. Exhibition of large amplitudes indicates that the correct value of the key has been found.



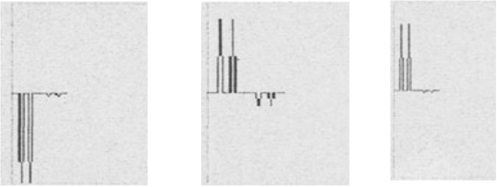
The images above show some of the output produced by the simulator. The first image shows an example of a generated power trace. The second trace shows the difference trace for splitting on the input only. The spikes indicate where the input can still be recognized. The third and fourth image show the difference trace for an incorrect and a correct guess of the key, respectively. The correct guess can be identified because it produces a large downward spike.

3. Flexibility of the PINPAS tool

The PINPAS tool is flexible in the selection of the algorithm that is being investigated, making it easy to check several algorithms for a given platform. One can choose any well-known algorithm, e.g. DES, IDEA, etc., or any proprietary algorithm but one can also switch between different implementations of the same algorithm. This makes it feasible to evaluate software defenses introduced in an implementation to protect against vulnerabilities that have been found. Another aspect of the flexibility of the PINPAS tool lies in the choice of the hardware platform. One can easily switch between different cards, for example cards based on well-known processors, based on specialized processors or even based on processors which have only been designed but not (yet) implemented in hardware. This flexibility in the choice of the card allows testing of an algorithm on several potential cards and assessing vulnerabilities of an algorithm irrespective of the precise platform.

In the remainder of this section we illustrate how both software and hardware defenses can be tested in the tool. This is done by introducing a DPA countermeasure for the vulnerability found in the example algorithm. The DPA attack above is based on the fact that information about the Hamming weights of certain values leaks in the power usage. Several techniques have been proposed in the literature to strengthen smartcard algorithms, e.g. (Chari et al., 1999; Messerges, 2000; Goubin, 2001). One possible defense is to protect important values by masking them with random values. Unmasked values are only used within the registers of the chip, which are assumed not to leak significantly. This defense protects against the attack described above, but is still not sufficiently safe. To render information about Hamming weight of values completely useless one can apply so called dual rail logic. When using dual rail logic the Hamming weight of values is always the same and thus cannot provide the attacker with any useful information. An implementation of this scheme purely in hardware can be efficient and transparent to the algorithm running on the card, but at a price; the hardware resources have to be doubled in size. To limit the cost one could include dual rail operations in the processor but leave the choice of when to invoke these to the programmer, allowing the non-sensitive data to be coded normally.

By using the dual rail operation `XORdr`, which calculates the exclusive or of numbers in dual rail format, we can easily implement our example program as `x = single2dual(input); y = Sbox2[x XORdr key2]; return dual2single(y)`. Here Sbox and key are stored in dual rail format (denoted by `Sbox2` and `key2`). The function `single2dual` converts an 8-bit number to its 16-bit dual rail equivalent. Finally, `dual2single` converts back from the dual rail format. The implementation of this algorithm on our generic smartcard is straightforward. The function `XORdr` is directly mapped to a dual rail operation implemented on the chip.



The first image shows the difference trace for splitting on the input only. The value of the input can only be recognized at the start (first two spikes), after this no leakage occurs. The other two images show that a wrong and a correct guess of the key can no longer be distinguished: The trace for the correct key guess is not significantly different from the traces for incorrect guesses.

4. Conclusion

We have introduced the PINPAS tool which can simulate a smartcards power consumption and a side channel attack on that smartcard. The software simulation done with the PINPAS tool can be profitable in the testing of existing cards and algorithms. The tool is especially useful as an aid in the design of both cards (hardware) and algorithms (software), allowing for an assessment of the risk of side channel attacks in a much earlier stage of development. This way the production cycle and time-to-market of a new smartcard product can be greatly reduced. We have illustrated how the development process can be supported by the PINPAS tool for a toy algorithm which, although being very simple, contains steps comparable to widely accepted encryption standards.

References

Chari, S., Jutla, C., Rao, J., and Rohatgi, P. (1999). Towards sound approaches to counteract power-analysis attacks. In Wiener, M., editor, *Proc. CRYPTO'99*, pages 398–412. LNCS 1666.

Fahn, P. and Pearson, P. (1999). IPA: A new class of power attacks. In Koç, C. and Naccache, D., editors, *Proc. CHES'99*, pages 173–186. LNCS 1717.

Goubin, L. (2001). A sound method for switching between boolean and arithmetic masking. In Koç, C., Naccache, D., and Paar, C., editors, *Proc. CHES 2001*, pages 3–14. LNCS 2162.

Kocher, P. (1996). Timing attacks on implementations of Diffe-Hellman, RSA, DSS, and other systems. In Koblitz, N., editor, *Proc. CRYPTO'96*, pages 104–113.

Messerges, T. (2000). *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms*. PhD thesis, University of Illinois, Chicago.