Modelling of the Tamper-Proof Devices

Martin Hrubý *
hrubym@fit.vutbr.cz

Petr Peringer*
peringer@fit.vutbr.cz

Zdena Rábová*
rabova@fit.vutbr.cz

Abstrakt This paper deals with modelling the certain security aspects of hardware implementations of cryptographic mechanisms. These implementations are called *tamper-proof devices* and they are typically smart cards or cryptographic coprocessors. The design of most available tamper-proof devices does not deal with various forms of side effects of cryptographic algorithms execution. A side-channel is the term commonly used for places where such undesirable information appears. The volume and usefulness of information we are able to extract depends on the form of the side-channel. However, number of scientific papers proves that it is possible to effectively analyse information from side-channels and use the results for compromise of tamper-proof devices.

Keywords: tamper-proof hardware, cryptographic attacks, cryptography, heterogeneous modelling, HELEF

1 Introduction

We just started a new research project called "Information system security - research of attacks on tamper-resistant cryptographic hardware" at the Department of Intelligent Systems, FIT VUT. The main goal of the project is a research of security aspects of hardware implementations of cryptographic mechanisms.

The project consists of three main areas. We will design and build laboratory instruments for implementation of attacks exploiting side-channels. The attacks will produce experimental data. The second area consists of a design of analytic methods for effective processing of experimental data in such a form that allows reconstruction of sensitive data. The proposed methods will be checked on real devices. The last area deals with the design of countermeasures against side-channel attacks. We will use already known approaches as well as the principles of our analytic methods. The goal is to decrease the amount of useful information in side-channels and determine minimum amount necessary for successful attack.

2 Modelling the tamper-proof devices and attacks

We are going to model several aspects of mentioned devices and interfaces what attackers may use to realise their attacks. There are:

* Brno University of Technology, Faculty of Information Technology, Božetěchova 2, 612 66, Brno, Czech Republic

– Models of hardware and firmware (cards, safe interface, attacker's interface).
– Models of side-channels and their analysis.
– Models of attacks.

At this stage, we have created a model of attack using so called Simple Power Analysis (SPA) [5]. This paper starts our long term research of modelling methods of tamper-proof devices, their possible weaknesses and corresponding attack methods.

Presented model is a simple demonstration showing a top-level abstraction of a card, an attack interface and attacking method. Further models will be already a more detailed inspection of internal signals and events inside the tamper-proof device.

We are going to utilize results of our past research of heterogeneous modelling[6] in this work. Heterogeneous modelling allows us to employ many of existing modelling tools such as the SIMLIB/C++ simulation library, new simulation language called HELEF[1, 2], Prolog, continuous simulation language TKSL/C and others.

At the end, we want to be able to inspect a given tamper-proof device and qualify:

– All information sources exploitable by the attacker so that he can drop down a level of card security.
– Level of general security expressed with some exact experimental or simulation results.
– Necessary steps to increase its level of security.

2.1 Simple Power Analysis (SPA) method

SPA is the most easy attack method (attacks with low cost equipment). The attacker has got a testing instrument which connects a card, and pretends a safe environment of cash-machine (ATM). Card does not have its own electrical supply, so it gets the electricity from the outer environment. Attacker measures the input electrical current flow to analyse which internal instruction of card's CPU just being executed. Some of that operations are characteristic as they cause detectable curves of power consumption[5]. More sophisticated version of SPA is called Differential Power Analysis and is described in [4].

Let us have a smart card for use in most spread ATMs. To demonstrate the basics of our model, we will simplify card-to-ATM communication into just checking the given pin-code for its correctness and returning some status flag.

When the card is given a pin code, it executes a code of the following algorithm:

```
cycles := EEpromread(cycles); // read counter of pin enter
attempts
if (cycles==0) return; // close the communication
pin := EEpromread(pin); // read the on-card pin number
while (cycles>0) { // while the access is allowed
  if (inputpin==pin) break; // correct pin entered
  else { // failure
     cycles := cycles - 1;
     EEpromwrite(cycles); // write the decremented counter
// This operation IS WATCHED!
     return ;
  }
}
```

```
online := true; // return the status flag
// to the interface (card ready)
cycles := 3;
EEpromwrite(cycles); // re-set the counter
```

EEpromwrite operation is that characteristic one. Attacker's interface detects this moment and switches off the supply. Write operation is not finished and so the counter does not change its value. The attacker goes through the pin codes from 0 to 9999 and has an unlimited chance to find the right pin code.

2.2 The HELEF Language

The HELEF language has been originally designed to combine declarative and procedural programming where models are extended with some knowledge-based parts. This mixture of paradigms requires a special way of computing called AIN in the HELEF terminology.

HELEF is a brand-new language and is not completely documented yet. There are some papers [2, 1] about the HELEF language describing the AIN computing paradigm, HELEF style of the object orientation, its use and applications.

The HELEF language combines several programming and modelling concepts such as:

- Declarative and procedural programming.
- Heterogeneous modelling (also call multi-paradigm or multi-formalism modelling).
- Non-class object oriented programming (inspired by the Self [3] language).
- Interactive (rapid prototyping) building of models.
- Persistent models.
- Distributed modelling, backtracking and others.

3 Model of attack based on SPA

In this chapter, we are going to present our simplified model of the inspected card, safe ATM interface, attacker's test interface and the attack method.

3.1 Safe interface of ATMs

```
cashm = environment {
   sup := true // power on
   clk := 0
   clk(AINnoevent) = clk + 1 // clock cycles
}
```

3.2 Attacker's interface

```
hack = environment @cashm {
   sup(power>level) = false // switch off if the input
          //  current overflows a specified  level
   // Attack algorithm
   process(defined(card)) { // when card is inserted
      for inputpin := (0, 9999) { // force attack
```

```
            // process waits until the card answers
            // the online status flag
            // or the current is automatically switched off
            sup := true // switch on
            continue = ((sup=false) || online=true)
            if (online=true) halt
        }
    }
}
```

Model starts when `entity card` is inserted to the `environment hack`. Then `input-pin` has got a new value and `sup` becomes `true`. Following sub-models represents structure and behaviour of the inspected tamper-proof device (smart card).

3.3 Model of the smart card

3.3.1 Action

This example should demonstrate a different way of making models in the HELEF simulation language. In this model, a static structure (CPU and the EEprom memory) of the card and its dynamic behaviour are separated.

Presented attack method switches off the power supply in certain moments. We want to model this as a discrete event at this level of abstraction. We show the exact state of the model when the card is automatically disconnected by the attacker. Dynamic part is represented here as a state of memory cells (CPU registers, EEprom memory).

Object `entity onoff` is a pattern of power dependent object (information `sup`). Object `onoff` models a future event which cancels it-selves when sup becomes $false$.

```
onoff=entity {
  local at, todo;
  process(at) { // when  at  is true
    do todo; // execute  todo
  }
  process(sup=false) { // power is down. Remove the whole
         // object off the model
     remove parentStructure;
  }
}
```

3.3.2 The EEprom memory unit

```
EEpromItem = entity { // one cell of EEprom
   process(sup=true) { // action on power-up
      output := nil
      power := 0
      ready := nil
   }
   process(sup=false) {
      output := nil
```

```
      ready := nil
   }
   read = process { // read method
      power := 1 // define its consumption
      ++ entity @onoff { //  plan the event
         at = (clk= ^clk+1) // at next clk step
         todo = process {
             power := 0
             output := value
             ready := true
         }
      }
   }
   // write process  similarly with power := 10
}
```

The `write process` plans an event (derived from `onoff`) which will proceed its `todo` operation in the next `clk` cycle or it will disappear if `sup` changes to `false` suddenly. Write operation causes increase of the power consumption detected by the attacker.

3.3.3 The card - main part

```
card = entity {
    // object pin  (cell of EEprom memory)
    pin = entity @EEpromItem { value := 1234 }
    // object cycles
    cycles = entity @EEpromitem { value := 3 }
    online := false
    // export power object to supervise. structure
    export power;
    // complete power income
    power = pin.power + cycles.power + secret.power + cpupower;
    cpupower := 0
    process(sup=true) {
       // authentization algorithm
    }
}
```

The model of card and the attacker interface is completed by inserting the `entity card` into `environment hack`.

```
   world = environment {
      environment hack {}
      process() {
          ++entity card -> hack
      }
}
```

Card and interface objects are connected using names `inputpin, sup, clk, online`. Except that names, they are totally independent (and re-usable).

Simulation runs in the following steps:

1. Model is completed.
2. Attacker sets the `inputpin` and switches on the supply.
3. Card starts its card-holder authentization (verification) algorithm.
4. If pin is correct, it sets `online:=true` and attacker finishes.
5. Otherwise it attempts to write the EEprom memory, power exceeds the watched level and `sup` goes to `false`. Card is frozen. Attacker process is activated again and continues with the next pin number (step 2).

Card model shows a difference between static and temporary pieces of information included in the model. It models an object carrying some value which is:

- Undefined when power goes up.
- Defined when set by the CPU.
- Not existing when power falls down.

HELEF approach in modelling is very different because the `hack` model does not influence directly the `card` model. Environment `hack` does not include any action which would write a value straight in the card. We model a device with no possible way of doing so.

4 Summary

We presented a beginning of our research in cryptography and information security. Part of this research is pointed at modelling methods and creating models. We are going to use our experience in heterogeneous modelling and the HELEF language.

This work is supported by the Grant Agency of Czech Republic, grant No. GACR 102/04/0871: "Information system security - research of attacks on tamper-resistant cryptographic hardware" and also by the research project CEZ:J22/98:262200012 "Research in Information and Control Systems".

Bibliography

1. Hrubý, M., Rábová, Z.: HELEF: A New Simulation Language, In: Proceedings of International Conference MOSIS '02, Ostrava, CZ, MARQ, 2002, p. 39-47, ISBN 80-85988-71-2
2. Hrubý, M.: Formal Specification of the HELEF Simulation Language, In: Proceedings of 37th Internation Conference MOSIS'03, Ostrava, CZ, 2003, p. 143-148, ISBN 80-85988-86-0
3. Ungar, D., Chambers, C., Chang, B., Holzle, U.: Organizing Programs Without Classes, In: LISP and Symbolic Computation: An International Journal, Netherlands, 1991
4. Coron, J.S., Kocher, P., Naccache, D.: Statistics and Secret Leakage, In: Financial Cryptography (FC2000), vol. 1962 of Lecture Notes in Computer Science, pp. 157-173, Springer-Verlag, 2001
5. Agrawal, D., Archambeault, B., Rao, J. R., Rohatgi, P.: The EM Side-Channel(s): Attacks and Assessment Methodologies, technical report, IBM Watson Research Center
6. Hrubý Martin, Kočí Radek, Peringer Petr, Rábová Zdenka: Tools for Creating of Multimodels, In: Kybernetes: The International Journal of Systems & Cybernetics, roc. 2002, c. 9, GB, p. 1391-1400, ISSN 0368-492X