Differential Fault Attacks on RSA Smartcards

Phuc Vo
School of Electrical Engineering and Computer Science
Oregon State University
voph@engr.orst.edu

Abstract— **Smart cards is a security technology that is becoming more and more popular. As a result, they are targets of criminals who wish to break their encryption in order to abuse what they protect. One form of attacks that can be used is the differential fault attack (DFA). This report will cover what are DFA's, how they can be carried out against a smart card and solutions that can defend such an attack.**

I. Introduction

Smart cards are a technology that provides higher reliability, more memory, and better performance than conventional magnetic strips [1]. As a result, smart cards are becoming more popular in use. They can be used for identity authentication and payment services. Common applications of smart cards include credit cards, mobile phones' SIM card, pay-TV access control and data storage that can contain a person's medical record or personal identity. Because smart cards holds such valuable informations that can be used for economic gains, they are a popular target for attacks. These attacks include invasive attacks like reverse engineering where smart cards are depackaged and completely analyzed and non-invasive attacks like differential power attack, electromagnetic analysis, and differential fault attacks (DFA). This paper will discuss DFA in particular and provide different countermeasures to these attacks.

II. Differential Fault Attacks

Differential fault attacks disturbs the function of the smart card through physical means in order for the smart card to output faulty data. This faulty data can then be used to reveal the secret key of the smart card. Two types of DFA that can be used to break a smart card are glitching and optical fault induction attacks.

A. Glitching

Glitching is an attack done long ago by hackers to break pay-TV smart cards [2]. This method involves applying a glitch (a rapid transient) to the smart card's clock or power source. The smart card's processor can then be made to execute a number of incorrect instructions by varying the duration and precise timing of the glitch. This can cause the secret key to be outputted and checks of passwords and access rights to be skipped over. For example, the following loop is commonly used to output the contents of a limited range in memory 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 aim of glitching attacks is to increment the program counter as usual but modify the conditional branch in line 3 or the decrement of variable a in line 6. The glitching attack can then be repeated such that the entire contents of the memory is outputted.

B. Optical Fault Induction Attack

Another DFA is the optical fault induction attack that was described in [3]. This attack uses a laser to change the state of a memory cell. By exposing an intense light source to CMOS logic, the semiconductor becomes ionized and can cause a new value to be written. The experiment carried out by Skorobogatov used a light from a magnified photoflash lamp to successfully change a bit in a SRAM chip. By manipulating the data in the smart card, faulty data can be outputted. This faulty data can then be used by the Chinese Remainder Theorem (CRT) to find the smart card's secret key.

C. Finding the Secret Key Using CRT

Using the CRT to find the secret key of a public key crytosystem was first discussed in [4]. Devices using public key cryptosystems to generate signature may be attacked to inadvertedly reveal their secret keys. This can be done if the following conditions are true: the message as signed is known, a ceratin type of faulty behavior occurs during signature generation and the device outpus the faulty signature.

RSA uses two primes: p and q and sets $n = pq$. RSA signs a message with the private key, d, by computing

$$s := m^d \bmod n$$

However, computing this exponent consumes a large amount of time and power. A more efficient method that is currently implemented in smart cards is to first compute the following equations:

$$s_p := m_p^d \pmod p$$

$$s_q := m_q^d \pmod q$$

Where $d_p := d \pmod{p-1}$ and $d_q := d \pmod{q-1}$. The Chinese remainder theorem can then be used on these two equations to find s from s_p and s_q.

If an error occurs while computing s_p and the faulty value s'_p is outputted, then a faulty signature s' will also be outputted for the message m. The secret exponent q can be computed by:

$$q = \gcd(s'^e - m \pmod{n}, n).$$

As can be seen from the above equation, the secret exponent q can be found without knowing the private key n.

III. Countermeasures to DFA

There are many ways to make smart cards more resistant to DFA's [5], [6]. This can be done by changing the hardware of the smart card itself or the software ran on the smart card. In general, smart cards should have mechanisms that can prevent glitching attacks, detect errors during runtime or check the results of the computation before outputting the data.

A. Asynchronous and Self-Checking Logic

One method to make a smart card resistant to fault attacks is to use asynchronous and self-checking logic [5]. By removing the clock from smart cards and having it employ self-timed circuits, clock glitching attacks are no longer possible. The self-checking property of *m-of-n* encoded circuits can be use to defend against power glitching and fault injection attacks. Using *m-of-n* encoded circuits provides self-checking properties due to the fact that a codeword of length n is only valid when it contains m 1's. An example of such a coding scheme is the dual-rail encoding (See Table-1).

Table 1: Dual Rail Encoding

A0	A1	meaning
0	0	clear
0	1	logic 0
1	0	logic 1
1	1	alarm

By detecting whether a codeword has an incorrect number of 1's one of two behaviors can occur. In the case of a reduction in the number of 1's, the completion of the program will not be allowed and the circuit will be stalled. If there is an increase in the number of 1's, an error will be detected and an alarm signal can be propagated throughout the circuit causing a system-wide deadlock. This coding scheme will also allow an alarm state to be activated by OR-ing an alarm signal from tamper sensors with the alarm signals from the dual-rail circuit (see Figure 1).

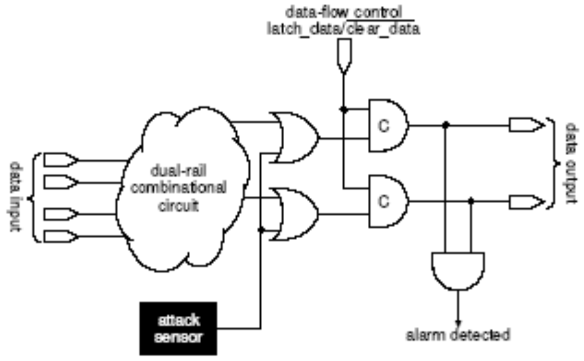

Figure 1: Alarm Insertion and Detection

B. Automatic Integration of Countermeasure

Another technique used on smart cards to resist DFA's is to make the smart card's software more robust by reducing the weakness of the software to a few and known checkpoints. This can be done through a preprocessor that automatically integrates an attack detection system into the source code of the smart card's software [6]. This detection system maintains a history of a subset of successive program points, called flags, that have been passed from the beginning. The detection system will check the consistency of this history regarding the control flow of the program at certain checkpoints determined by the developer. At each checkpoint, the history is checked against the control flow of the program and the detection system determines which controls must be enforced. Depending of the state of the smart card, these controls can accept or reject executions depending on conditions defined by the developer. This is an important feature for critical operations, like instance file creation, that are allowed or forbidden depending on the smart card's state, i.e. personalization, post-issuance, etc.

The preprocessor creates the detection system based on the location and data of directives included into the source code by the developer. The four types of directives are starting points, flags, check points and race declaration. Starting points specifies where in the program must the detection system start recording the history. Locations in the code that need to be considered by the history are indicated by the flag directives. The checkpoint directives designates where in the code must the checkpoints be enforced. The race declaration, along with a dynamic condition, determines whether or not a race, a family of executions, must be accepted or not. The developers specifies in the flag directives the family of accepted executions. During compiling, the preprocessor will replace a starting point with a piece of code that resets the data structure used by the detection system to store the execution history. The flag and checkpoint directives will be replaced by a piece of code that updates this data structure and a piece of code that enforces the control repsectively. Coding that activates or deactivates a race in question replaces the race declarations. All of these pieces of code then form the detection system in the software.

IV. Secure RSA Algorithms

Another way to make smart cards that implements RSA more robust is to make the algorithms used to calculate RSA signatures more secure against DFA's [7], [8], [9]. This scheme seeks to check if the RSA signature faulty before outputting the signature and prevent the RSA secret factors from being leaked out.

A. Shamir's Algorithm

One of the first RSA algorithm proposed to be more scure against DFA's was introduced by Shamir [7]. Let $n = pq$ be the RSA modulus and (e,d) be verification, signature exponent, respectively. Also, let the Chinese remaindering theorem be represented as CRT(). Shamir's algorithms runs as follows:

```
1: Choose a small, random $r$.
2: Compute: $s_{rp} := m^d \bmod rp$.
3: Compute: $s_{rq} := m^d \bmod rq$.
4: Check if $s_{rp} := s_{rq} \bmod rq$
       then s = CRT($s_{rp} \bmod p$,$s_{rq} \bmod q$. )
       else: Output an error signal.
```

The probability an error will go undetected in this algorithm is about $\frac{1}{r}$. However, Shamir's technique to computing the RSA signature is slow because the exponent, d, is generally unknown and $d_p$and d_q are given instead. This would require the algorithm to compute d in order to sign a message. Another theorized weakness to this algorithm is the step where s_{rp} is checked against s_{rq}. If this step were to be skipped, because of a clock glitching attack for example, it is possible the algorithm will output a faulty signature.

B. Infective CRT-RSA

Infective CRT-RSA is another approach to computing RSA signatures but eliminating the possiblity of an attack on the self-checking step in Shamir's algorithm. In Infective CRT-RSA, two integers, $t_1$and t_2 are used as the small moduli like in Shamir's approach. These integers must be of sufficiently large bitlength and satisfies these conditions:

1. $t_1$and t_2 must be coprime
2. gcd(d,$\varphi(t_1) = 1$ and gcd(d,$\varphi(t_2) = 1$
3. $t_1$and t_2 are square-free
4. $t_1$and $t_2 \equiv 3 \bmod 4$
5. $\chi\mathbf{X} = pt_1 \cdot ((pt_1)^{-1} \bmod qt_2$

Futhermore, the following pre-computations must be done:

```
Compute $d_p := d \bmod \varphi(p \cdot t_1)$
Compute $d_q := d \bmod \varphi(q \cdot t_2)$
Compute $e_{t_1}$, such that $d \cdot e_{t_1} := 1 \bmod \varphi(t_1)$
Compute $e_{t_2}$, such that $d \cdot e_{t_2} := 1 \bmod \varphi(t_2)$
```

The Infective CRT-RSA algorithm then runs as follows:

```
1: Compute: $s_p := m^{d_p} \bmod p \cdot t_1)$
2: Compute: $s_q := m^{d_q} \bmod p \cdot t_2)$
3: Compute: $s :=$ CRT$(s_p, s_q) \bmod N \cdot t_1 \cdot t_2$
4: Compute: $c_1 := (m - s^{e_{t_1}} + 1) \bmod t_1$
5: Compute: $c_2 := (m - s^{e_{t_2}} + 1) \bmod t_2$
6: Compute: $sign := s^{c_1 \cdot c_2} \bmod N$
```

This approach protects every single computation step of the signature algorithm by using the two integers t_1 and t_2 to compute $s_p$ands_q. By combining s mod $N \cdot t_1 \cdot t_2$ through CRT, infective computations can be done to calculate the signature. Because this method uses infective computations, the final signature will be false modulo p and q. As a result, single points of failures can be avoided.

C. Secure Modular Functions

A generalization and improvement of Shamir's technique was proposed in [9]. This approach saw that any modular function, f, can be calculated in a similar way to Shamir's algorithm. That is, this realations was found:

$$f(x) \bmod p = [f(x) \bmod rp] \bmod p$$

This relation can be used to effectively check two half exponentiations seperately and check them for errors. This relation can be used for RSA signatures by the following steps:

```
1: Choose two random numbers: $r_1$ and $r_2$
2: Compute $S_{11} := P(m)^d p \bmod r_1 p$
3: Compute $S_{12} := P(m)^d p \bmod r_1$
4: Compute $S_{21} := P(m)^d q \bmod r_2 q$
5: Compute $S_{22} := P(m)^d q \bmod r_2$
6: Check if both $S_{11} := S_{12} \bmod r_1$ or $S_{21} := S_{22} \bmod r_2$
       If they are equal, then $S = CRT(S_{11} \bmod p,\ S_{21} \bmod q)$
       Else, output an error signal.
```

The function P is a padding function. The probability an error passes through this method undetected is $\frac{1}{r}$. By checking two half exponentiations, the time it takes to output the signature or an error signal is much less than Shamir's approach.

V. Conclusion

Because of their rising popularity, smart cards have become a target of many different and ingenious attacks. One of the most clever and less costly of these attack are the differential fault attacks. These attacks include glitching attacks on the clock or power supply of a smart and optical induction attacks on the logic device in the smart cards processor or memory. The purpose of these attacks are to disrupt the operation of the smart card and cause it to reveal its secret key. To defend against these attacks, there are many hardware and software countermeasures that are proposed and used. This includes using a self-timed and asynchronous circuit, implementing an error detection system in the software of the smart card and changing the RSA algorithm on the smart card to make them more robust against DFA's. It should be mentioned that the countermeasure presented in this paper are not necessarily the most secure techniques because their weaknesses are still being investigated. However, they do serve as a basis on which more robust smart cards can be built.

References

[1] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert, “Fault attacks on RSA with CRT: Concrete results and practical countermeasures,” .

[2] R. Anderson and M. Kuhn, “Tamper resistance - a cautionary note,” in *Proceedings of the Second Usenix Workshop on Electronic Commerce*, 1996, pp. 1–11.

[3] S. Skorobogatov and R. Anderson, “Optical fault induction attacks,” CHES 2002, LNCS 2523, pp. 212.

[4] M. Joye, A. K. Lenstra, and J. Quisquater, “Chinese remaindering based cryptosystems in the presence of faults,” *Journal of Cryptology*, vol. 12, no. 4, pp. 241–245, 1999.

[5] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor, “Improving smart card security using self-timed circuits,” in *Asynchronous Circuits and Systems*, 2002, pp. 211–218, Eighth International Symposium on Advanced Research in Asynchronous Circuits and Systems, Computer Laboratory, University of Cambridge.

[6] M. Akkar, L. Goubin, and O. Ly, “Automatic integration of counter-measures against fault injection attacks,” Pre-print found at `http://www.labri.fr/Perso/ ly/index.htm`, 2003.

[7] A. Shamir, “How to check modular exponentiation,” Presented at the rump session of EUROCRYPT’97, 11-15th May 1997, Konstanz, Germany.

[8] M. Otto J. Blőmer and J.P. Seifert, “A new CRT-RSA algorithm secure against bellcore attacks,” in *Proceedings of the 10th ACM conference on Computer and Communications Security*, 2003, pp. 311–320.

[9] P. Paillier M. Joye and S. Yen, “Secure evaluation of modular functions,” in *International Workshop on Cryptology and Network Security*, 2001, pp. 227–229.