BZA06 - MNG

Model 2019

Bezpečná zařízení

Část 6 Chybová analýza

Post 18/19

Souhrnné materiály

Ver 0.1

©Petr Hanáček

BZA

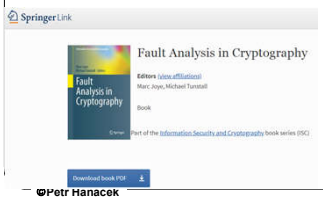
New 2018/19

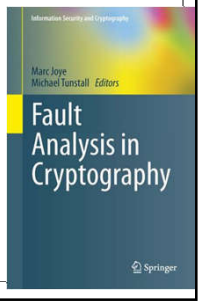
BZA 06
Chybová analýza

Petr Hanáček
Faculty of Information Technology
Technical University of Brno
Božetěchova 2
612 66 Brno
tel. (05) 4114 1216
e-mail: hanacek@fit.vutbr.cz

Fault Analysis in Cryptography

- Marc Joye, Michael Tunstall: Fault Analysis in Cryptography
- Volně dostupná pro studenty VUT



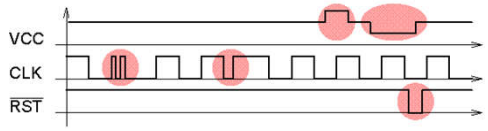


Logické útoky

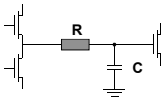
Vpp útok na čip. karty

- Starší čipy potřebují na Vpp kontaktu napětí 15-20 V pro zápis do EEPROM nebo EPROM
- Aktivní útok - odstraněním Vpp zabránit zápisu. Cílem je:
  - Nezapsání čítače chybných zadání PINu
  - Neodečtení impulsu (tlf. karty)
  - Zabránění zablokování karty (Infinite Lives Hack)
- Pasivní útok - monitorováním Vpp zjistit, kdy karta zapisuje (a jaký objem dat)
  - útoky závislé na protokolech
- Obrana
  - nepoužívat karty, které potřebují Vpp

Vyvolání poruchy zákmitem



- Zákmit (glitch) může způsobit neprovedení instrukce
  - spoje mezi hradly tvoří RC zpožďovací členy
  - hodnota RC spoje určuje maximální frekvenci hodin
  - signál RST není někdy vzorkován, což dovoluje částečný reset
  - transistory porovnávají VCC a napětí na kondenzátoru C, což dovoluje i použití zákmitů na VCC



Glitch útok na výstupní cyklus

- Typický program pro výstup dat v čipu:

```
» 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    ...
```

- Útok:
  - Pokud se zákmitem na VCC nebo CLK podaří způsobit neprovedení instrukce 3 nebo 6, program pošle na výstup více dat (potenciálně celou dostupnou paměť procesoru)

BZA

Cíle glitch útoku

- Výstupní cyklus
  - viz. předchozí příklad
- Často používaný - typicky pro zjištění klíčů
- Kontrola hesel a PINů
  - zabránění zápisu nesprávného pokusu
- Kontrola přístupových práv, odpověď protokolu
  - změna výsledku testu přístupového práva
  - změna kladné odpovědi na zápornou a naopak
- Kryptografický algoritmus
  - pomocí chybného výsledku zjistit klíč nebo šifrovaný text
  - viz. dále - Diferenciální chybová analýza
- Ochrana na úrovni programu
  - vyhýbat se algoritmům, jejichž výsledek závisí na jedné instrukci
  - "vymaskuji-li tři bity z bajtu, měl bych otestovat, zda výsledek není větší než 7"

Teplotní útoky

- Cíl útoku:
  - Nastavením teploty mimo povolené meze obvodu způsobit nefunkčnost některých obvodů
  - Vyhřátím nebo vychlazením zařízení v termostatu
- Napadané obvody
  - Obvod pro mazání paměti v okamžiku fyzického útoku (zeroization circuitry)
  - Obvod pro ochranu proti čtení paměti
  - Znemožnění zápisu do paměti EEPROM
  - Součinnost s paměťovou remanencí
    - udržení dat v zálohované SRAM několik sekund po odpojení napájení při fyzickém útoku
- Protiopatření:
  - Senzory teploty. V případě mimomezních teplot smazání citlivých informací.
  - Mazání zálohovaných SRAM musí být aktivní (nikoli pouze odpojení napájení)

Opatření proti glitch útokům

- Ochrana proti RST glitch
  - zpracování RST klopným obvodem
    - » dobrá karty mají
- Ochrana proti VCC glitch
  - kontrola tolerance napájecího napětí
    - » většina karet má, ale použití obvodu závisí na programátorovi
    - » v hodně aplikacích (možnost kolísání napájení) se nepoužívá
- Ochrana proti CLK glitch
  - kontrola frekvence hodinových impulsů
  - příliš nízká frekvence (ochrana proti single stepping útoku)
    - » většina karet má, ne vždy se používá
  - příliš vysoká frekvence (ochrana proti glitch útoku)
    - » má poměrně málo karet

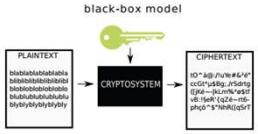
Diferenciální chybová analýza

- DeMillo, Boneh, Lipton, 1996
- Využívá indukovaných chyb v hardware (např. pomocí zákmitu - glitch)
- Nejzranitelnější jsou algoritmy s veřejným klíčem
  - Biham a Shamir rozšířili i na DES
- Poučení:
  - Chyby (byť i dočasné) hardware mohou narušit bezpečnost systému

Intro

Classical Cryptanalysis

- Black-Box Model assumed in classical cryptanalysis:
  - key(s) stored in the device
  - cryptographic operations computed inside the device



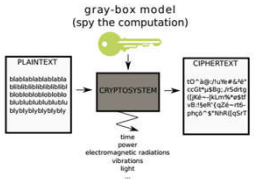
- The attacker has only access to pairs of plaintexts / ciphertexts.

Victor LOMNE - ANSSI / Fault-based Cryptanalysis on Block Ciphers

BZA

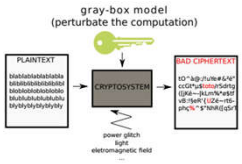
Secure Cipher - Unsecure Implementation (1/2)

- [*Kocher* + 1996] ⇒ exploitation of physical leakages
  - cryptosystems integrated in CMOS technology
  - physical leakages correlated with computed data



Secure Cipher - Unsecure Implementation (2/2)

- [*Boneh* + 1997] ⇒ exploitation of faulty encryptions
  - the attacker can generate faulty encryptions



- the attacker has access to correct & faulty ciphertexts
- New class of attacks ⇒ Fault Attacks (FA)

Fault Zoology (1/2)

- Different ways to generate a fault:
  - electrical glitch on pins (VCC, CLK, I/O, ...)
  - electrical glitch on the die (FBBI)
  - light injection
  - ElectroMagnetic (EM) field injection
- The duration of the fault can be:
  - transient
  - permanent

Fault Zoology (2/2)

- Different effects:
  - modification of operation flow
  - modification of operands
- Different goals:
  - Bypassing a security mechanism
    e.g. PIN verification, file access right control, secure bootchain, ...
  - Generating faulty encryptions/signatures
    ⇒ fault-based cryptanalysis
  - Combined Attacks
    JavaCard based, FA + SCA

Fault Injection Means

Electrical glitch on Power Supply (1/3)

- Principle:
  under/over-power a device during a very short time
- Over-powering cause unexpected electrical phenomenoms inside the IC
  e.g. local shortcuts, ...
- Under-powering slows down the processing of the IC
  e.g. bad memory read/write, ...
- Low/medium-cost attack
  ex. of equipment: custom electronic board, pulse generator, ...

BZA

Global Faults | Local Faults | Other Tools

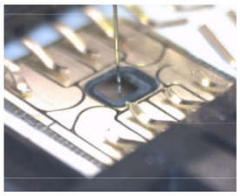
Electrical glitch on Power Supply (2/3)

- Adversary can control:
  - Amplitude of the glitch
  - Duration of the glitch
  - Shape of the glitch
- Generally no control of the fault precision:
  - On a microcontroller running code, modification of the current executed opcode and/or operand(s)
  - On a hardware coprocessor, modification of (some of) the current processed words (e.g. registers)

Global Faults | Local Faults | Other Tools

Electrical glitch on Power Supply (3/3)

- Recent variant [Tobich+ 2012]:
  FBBI: Forward Body Bias Injection
- Consist in putting a needle in contact with the IC silicon through its backside



Global Faults | Local Faults | Other Tools

Tamper the clock (1/2)

- Principle:
  reduce one or several clock period(s)
- slows down the processing of the IC
  e.g. DFF sampling before correct computation of current instruction/combinational logic ...
- Low/medium-cost attack
  ex. of equipment: custom electronic board, signal generator, ...

Global Faults | Local Faults | Other Tools

Tamper the clock (2/2)

- Adversary can control:
  - Duration of the reduced clock period
  - Number of reduced clock period(s)
- Generally no control of the fault precision:
  - On a microcontroller running code, modification of the current executed opcode and/or operand(s)
  - On a hardware coprocessor, modification of (some of) the current processed words (e.g. registers)

Global Faults | Local Faults | Other Tools

Light attacks (1/2)

- Principle:
  inject a light beam into the device to disturb it
- Old school setups were using flash lamp
- Modern setups are based on laser modules
- It requires to open the package of the IC in order the light beam can be injected into the frontside or the backside of the die

Global Faults | Local Faults | Other Tools

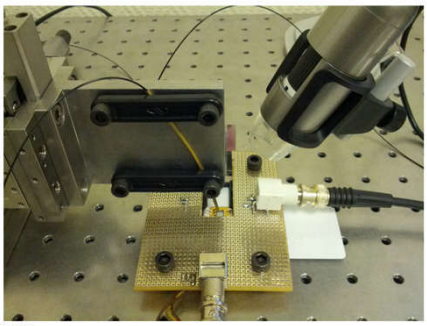
Light attacks (2/2)

- A photoelectric phenomenom transforms light energy into electrical energy, provoking unexpected behaviour of transistors
- On complex ICs with many metal layers, or on secure ICs with a shield, it can be difficult to inject light on the frontside of the IC
- As silicon is transparent to infrared light, backside light injection uses infrared light
  e.g. NIR laser diodes
- Medium/high cost attack

BZA

Global Faults| Local Faults | Other Tools|

Laser Setup example 1 (1/2)



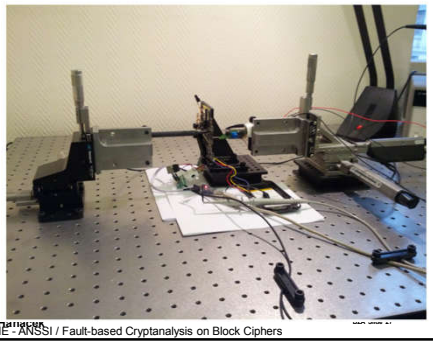
Global Faults| Local Faults | Other Tools|

EMI attacks

- Principle:
  inject an electromagnetic field inside the device to disturb it
- Can be done without removing the package of the IC
- In practice, a glitch of high power is injected into an EM sensor put above the IC
  ex. of equipment: high power pulse generator + EM sensor
- Medium/high cost attack

Global Faults| Local Faults | Other Tools|

ElectroMagnetic Injection Setup example



Global Faults| Local Faults| Other Tools

Synchronization Mean

- In many cases, need of a synchronization mean to trig the fault at the right instant
- A classical way consists in monitoring the power consumption/EM activity of the IC such that finding the side-channel signature of the event one wants disturb
- Several solutions:
  - Using the triggering capabilities of oscilloscopes
  - Using a custom synchronization board, with real-time pattern matching mechanism

Methods

Safe Error Attack (SEA) [Biham+ 1997]

- SEA requires two copies of the target device:
  - a first copy that the adversary can fully control
  - a second copy set at an unknown secret
- SEA requires the ability to encrypt several times the same plaintext
- SEA does not require any faulty ciphertext
- SEA requires two phases:
  - a profiling phase
  - an attack phase

BZA

Fault Model | Safe Error Attack | DFA | Statistical Fault Attack

Safe Error Attack (SEA) - Sketch

1. Profiling phase
   - Use the device the adversary can fully control
   - For every bit of the master key, find the fault parameters allowing to reset this bit
2. Attack phase
   - Use the device set at an unknown secret
   - Encrypt a plaintext and keep the ciphertext
   - For every bit of the key, encrypt once again the same plaintext, while injecting a fault with parameters of profiling phase for the current bit
   - If both ciphertexts are equal, the current bit is equal to 0, otherwise equal to 1

Fault Model | Safe Error Attack | DFA | Statistical Fault Attack

Differential Fault Analysis (DFA) [Piret+ 2003]

- DFA requires the ability to encrypt two times the same plaintext
- DFA requires to have one or several pairs of correct and wrong ciphertexts corresponding to the same plaintext
  $P_1 \rightarrow (C_1, \widetilde{C_1})$
  $P_2 \rightarrow (C_2, \widetilde{C_2})$
  ...
  $P_N \rightarrow (C_N, \widetilde{C_N})$
- DFA requires to be able to fault only a part of the **State** at a particular position in the encryption
  e.g. one byte of the AES **State** before the last **MixColumns**

Fault Model | Safe Error Attack | DFA | Statistical Fault Attack

Statistical Fault Attack (SFA) [Fuhr+ 2013]

- SFA has the property to work even with a set of faulty ciphertexts corresponding to different unknown plaintexts
  $P_1 \rightarrow \widetilde{C_1}$
  $P_2 \rightarrow \widetilde{C_2}$
  ...
  $P_N \rightarrow \widetilde{C_N}$
- Nevertheless it requires a Fixed Fault Logical Effect
  e.g. stuck-at a fixed value a **State** byte with a good probability

Fault Model | Safe Error Attack | DFA | Statistical Fault Attack

Statistical Fault Attack (SFA) - Sketch (1/2)

1. Collect a set of faulty AES ciphertexts $\widetilde{C_1}, \widetilde{C_2}, \ldots, \widetilde{C_N}$, by injecting a fault on one byte of the **State** after the penultimate **AddRoundKey**. We assume that the fault has a stuck-at effect to an unknown value e:
   $\widetilde{S^1_{ak}} = S^1_{ak}\ AND\ e, \quad e \in [0, 255]$
2. A collection of correct ciphertext bytes $C_1, C_2, \ldots, C_N$ would have an uniform distribution
   Here, due to the stuck-at fault, the collection of faulted ciphertext bytes $\widetilde{C_1}, \widetilde{C_2}, \ldots, \widetilde{C_N}$ has a biaised distribution

Countermeasures

Analog Level | Digital Level | Application to Cryp

(De)synchronization

- A fault injection requires a precise timing to be effective
- Adding temporal randomness makes the timing of the fault harder to set
- Classical ways to add temporal randomness:
  - jittered clock
  - dummy instructions
  - randomize operation flow
  - ...

BZA

IC Package as Countermeasure

- Several kind of fault injection techniques require to expose the die of the IC to perform the attack
  FBBI, laser, ...
- Depending on the type of package, it can be more or less easy to expose the die:
  - smartcard packages are easy to open
  - metallic packages can be mechanically opened
  - epoxy packages require a chemical attack
  - Package-on-Package or 3D IC technology make the chip opening a nightmare

Glitch Detectors

- The historical way to inject a fault in an IC is to under/over-power it during a short time
- Some IC manufacturers add glitch detectors after IC pads, checking that the current signal voltage stays in a defined range
- If a signal voltage goes outside from the defined range, a mechanism triggers an alarm
  e.g. flag set, interruption, reset, ...

Laser Detectors (1/2)

- Laser injection often requires to only disturb a small area of the IC
- It requires to perform a spatial cartography to find hot spots
  CPU/co-processor registers, memory cells or decoders, ...
- Laser detectors that are small dedicated blocks are placed among the other IC cells

Laser Detectors (2/2)

- Different kind of Laser detectors:
  - analog based laser detectors
    e.g. based on photodiodes
  - digital based laser detectors
    e.g. based on custom logic cells
- Laser detectors do not cover the whole suface of the IC, but make the job of the adversary harder

Redundancy

- Redundancy consists in:
  - performing two times an operation
  - comparing results of both operation executions
    ⇒ require a conditionnal test
- From a code theory point-of-view, it corresponds to the most obvious code one can construct
  ⇒ duplication code
- A variant consists in performing the operation and the inverse operation, then checking that the obtained result is equal to the initial data

Examples of Redundancy

- Redundancy can be used in different ways:
  - Sequential redundancy for a software function
  - Sequential or Parallel redundancy for a hardware function
  - Use of redundant logics (Dual Rail logic → SABL, WDDL, STTL, ...)
  - Securization of special registers by duplication or by storing a value and its inverse
    2 flip-flops are necessary to store one bit

BZA

Analog Level | Digital Level | Application to Cr

Error Detection Codes

- Error Detection Codes are efficient tools to check the integrity of data
- ECC can protect linear operations (they are based on linear applications)
- ECC cannot protect non-linear operations
  in particular they are not well suited to protect cryptographic primitives

©Petr Hanáček BZA Slide 43

Victor LOMNE - ANSSI / Fault-based Cryptanalysis on Block Ciphers

Symetrické šifry

©Petr Hanáček BZA Slide 44

DFA

- DFA targets almost whatever algorithm (known or even
- unknown)
- It works on complex bit operations, such as the ones involved in secret key cryptography
- It is demonstrated on DES
  - Reference: "Differential Fault Analysis of Secret Key Cryptosystems", by Eli Biham, Adi Shamir, CRYPTO 1997, LNCS 1294, pp. 513–525.

Differential Fault Analysis of Secret Key Cryptosystems

Eli Biham
Computer Science Department
Technion – Israel Institute of Technology
Haifa 32000, Israel
biham@cs.technion.ac.il

Adi Shamir
Applied Math. and Comp. Sci. Department
The Weizmann Institute of Science
Rehovot 76100, Israel
shamir@wisdom.weizmann.ac.il

©Petr

Source: Koc

DFA on DES

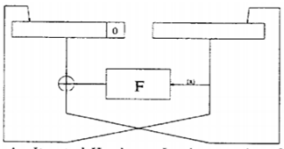


Figure 1. An Iterated Hardware Implementation of DES.

We generate a permanent fault in the least significant bit of the left-half register, either by destroying the bit or by cutting the wire which enters or leaves this cell, and assume that the value of this bit is permanently stuck at zero. Figure 2 describes the values computed by this iterated implementation in the last round.

©Petr Hanáček BZA Slide 46

Source: Koclab.cs.ucsb.edu, Cetin Kaya Koc

DFA on DES

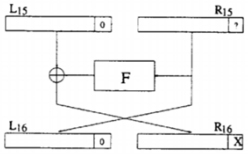


Figure 2. The Last Round in the Iterated Implementation of DES.

For any ciphertext the least significant bit (LSB) of the right half of the ciphertext (L_{16}) is zero, and the LSB of L_{15} is also zero. The LSB of the output of the F-function of the last round equals the LSB of the left half of the ciphertext (X, since the LSB of L_{15} is zero). This bit is the output of S7 in the last round. The input of S7 is formed by XORing 6 known ciphertext bits with six key bits. We can try all the 64 possible values of the six key bits, and discard any value which does not predict the LSB of the right half of the ciphertext. Each ciphertext is used to discard about half of the remaining key values. Thus, given about six ciphertexts (on whose plaintexts the attacker has no information of any kind) the right value of the six key bits can be identified. The other bits of the last subkey can be found by inducing additional faults, or by enumerating and verifying additional key bits.

©Petr Hanáček BZA Slide 47

Source: Koclab.cs.ucsb.edu, Cetin Kaya Koc

Asymetrické šifry

©Petr Hanáček BZA Slide 48

BZA

RSA

- Plain RSA signature is universally forgeable
- Messages should be appropriately padded and hashed
- RSA signature primitive
  - Setup $n = p \cdot q$ with p and q are prime
  - The public and private exponents satisfy $e \cdot d = 1 \pmod{\phi(n)}$
  - Public parameters (e, n)
  - Private parameters (d, n)
- Signature on message m

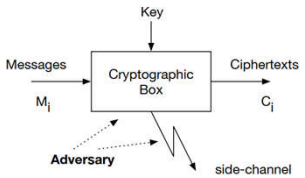
$$s = \mu(m)^d \pmod{n}$$

- Verification of the signature s

$$s^e \stackrel{?}{=} \mu(m) \pmod{n}$$

Source: koclab.cs.ucsb.edu, Cetin Kaya Koc

- The binary method of exponentiation leaks information on private key



Source: koclab.cs.ucsb.edu, Cetin Kaya Koc

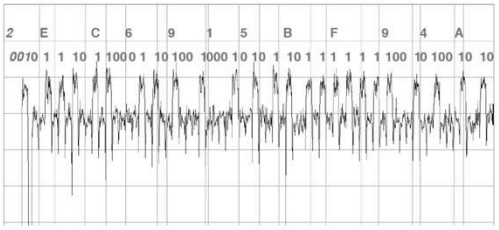
- The binary method is also known as Square-and-multiply algorithm

Input: m, $d = (d_{k-1}, \ldots, d_0)_2$, n
Output: $s = m^d \pmod{n}$
1. $R_0 \leftarrow 1$
2. For $i = k - 1$ downto 0
$R_0 \leftarrow R_0^2 \pmod{n}$
If $d_i = 1$ then $R_0 \leftarrow R_0 \cdot m \pmod{n}$
3. Return R_0

- It performs exponentiation left to right
- 2 Temporary variables R_0 and m
- Susceptible to SPA-type attacks

Source: koclab.cs.ucsb.edu, Cetin Kaya Koc

- The key: 2E C6 91 5B F9 4A



Source: koclab.cs.ucsb.edu, Cetin Kaya Koc

- One way to avoid leakage is to square and multiply at every step

Input: m, $d = (d_{k-1}, \ldots, d_0)_2$, n
Output: $s = m^d \pmod{n}$
1. $R_0 \leftarrow 1$; $R_1 \leftarrow 1$
2. For $i = k - 1$ downto 0
$R_0 \leftarrow R_0^2 \pmod{n}$
$b \leftarrow 1 - d_i$; $R_b \leftarrow R_b \cdot m \pmod{n}$
3. Return R_0

- When $b = 1$ (i.e., $d_i = 0$), there is a dummy multiplication
- The power trace is a regular succession of squares and multiplies
- 3 Temporary variables: R_0, R_1 and m
- Not susceptible to SPA-type attacks
- Susceptible to Safe-Error attacks

Source: koclab.cs.ucsb.edu, Cetin Kaya Koc

- Timely induce a fault into ALU during multiply operation at step i
- Check the output
  - If the result is incorrect (invalid signature or error notification), then the error was effective $\Rightarrow d_i = 1$
  - If the result is correct, then the multiplication was dummy (safe error) $\Rightarrow d_i = 0$
- Re-iterate the attack for another value of i

Source: koclab.cs.ucsb.edu, Cetin Kaya Koc

KONEC

Rezerva

Napěťové útoky

- **Cíle útoku:**
  - Nastavením teploty mimo povolené meze obvodu způsobit nefunkčnost některých obvodů
- **Napadané obvody**
  - Obvod pro mazání paměti (zeroization circuitry)
  - "Security bits", "Security locks", ...
  - Generátory náhodných čísel
- **Protiopatření:**
  - Senzory napětí. V případě mimomezních hodnot resetování.
- **Smazání "lock bitu" v neobvyklých podmínkách**
  - PIC 16C84: nastavení VCC na hodnotu VPP-0.5V a opakovaný zápis tzv. "lock bitu" způsobí jeho jeho smazání bez smazání paměti programu
  - Dallas DS5000, Intel 8051 kompatibilní: krátké výpadky napájecího napětí občas vedou ke smazání "lock bitu"
  - Generátor náhodných čísel v některých mikrokontrolérech dává při poklesu napájecího napětí téměř samé jedničky

Problém

- **Princip RSA:**
  - Klíče: e,d,n
  - Zašifrování: $C = M^e \bmod n$
  - Dšifrování: $M = C^d \bmod n$
- **Možná implementace RSA (binární umocnění)**

```
M := 1;
for i from t downto 0 do
   M := M * M;
   if d_i = 1 then M := M * C;
endfor
```

- **Co je na tom špatně?**

Fault attack assumptions

- Precise bit errors
  - The attacker can cause a fault in a single bit
  - Full control over the timing and location of the fault
- Precise byte errors
  - The attacker can cause a fault in a single byte
  - Full control over the timing but only partial control over the location (e.g., which byte is affected)
- Unknown byte errors
  - The attacker can cause a fault in a single byte
  - Partial control over the timing and location of the fault
- Random errors
  - Partial control over the timing and no control over the location

Source: koclab.cs.ucsb.edu, Cetin Kaya Koc

KONEC
KONEC

Differential Fault Analysis of Secret Key Cryptosystems

Eli Biham
Computer Science Department
Technion – Israel Institute of Technology
Haifa 32000, Israel
biham@cs.technion.ac.il
http://www.cs.technion.ac.il/~biham/

Adi Shamir
Applied Math. and Comp. Sci. Department
The Weizmann Institute of Science
Rehovot 76100, Israel
shamir@wisdom.weizmann.ac.il

Abstract

In September 1996 Boneh, Demillo, and Lipton from Bellcore announced a new type of cryptanalytic attack which exploits computational errors to find cryptographic keys. Their attack is based on algebraic properties of modular arithmetic, and thus it is applicable only to public key cryptosystems such as RSA, and not to secret key algorithms such as the Data Encryption Standard (DES).

In this paper, we describe a related attack, which we call *Differential Fault Analysis*, or *DFA*, and show that it is applicable to almost any secret key cryptosystem proposed so far in the open literature. Our DFA attack can use various fault models and various cryptanalytic techniques to recover the cryptographic secrets hidden in the tamper-resistant device. In particular, we have demonstrated that under the same hardware fault model used by the Bellcore researchers, we can extract the full DES key from a sealed tamper-resistant DES encryptor by analyzing between 50 and 200 ciphertexts generated from unknown but related plaintexts.

In the second part of the paper we develop techniques to identify the keys of completely unknown ciphers (such as SkipJack) sealed in tamper-resistant devices, and to reconstruct the complete specification of DES-like unknown ciphers.

In the last part of the paper, we consider a different fault model, based on permanent hardware faults, and show that it can be used to break DES by analyzing a small number of ciphertexts generated from completely unknown and unrelated plaintexts.

1 Introduction

In September 1996 Boneh, Demillo, and Lipton from Bellcore announced an ingenious new type of cryptanalytic attack which received widespread attention[11,5]. This

Copyright (c) 1998, Springer-Verlag

attack is applicable only to public key cryptosystems such as RSA, and not to secret key algorithms such as the Data Encryption Standard (DES)[17]. According to Boneh, "The algorithm that we apply to the device's faulty computations works against the algebraic structure used in public key cryptography, and another algorithm will have to be devised to work against the non-algebraic operations that are used in secret key techniques". In particular, the original Bellcore attack is based on specific algebraic properties of modular arithmetic, and cannot handle the complex bit manipulations which underly most secret key algorithms.

This type of attack on a tamper-resistant device shows that even cryptographic schemes sealed inside such devices might leak information about the secret key. Earlier papers on this subject, including the papers of Anderson and Kuhn[1], and of Kocher[8], had shown that tamper-resistant devices are vulnerable to several types of attacks including attacks against the protocols, attacks using carelessness of the device's programmers, and timing attacks.

In this paper, we describe a new attack, related to Boneh, Demillo, and Liptons' attack, which we call *Differential Fault Analysis*, or *DFA*, and show that it is applicable to almost any secret key cryptosystem proposed so far in the open literature. In particular, we have actually implemented DFA in the case of DES, and demonstrated that under the same hardware fault model used by the Bellcore researchers, we can extract the full DES key from a sealed tamper-resistant DES encryptor by analyzing between 50 and 200 ciphertexts generated from unknown but related plaintexts. In more specialized cases, as few as five ciphertexts are sufficient to completely reveal the key. The power of Differential Fault Analysis is demonstrated by the fact that even if DES is replaced by triple DES (whose 168 bits of key were assumed to make it practically invulnerable), essentially the same attack can break it with essentially the same number of given ciphertexts.

Differential Fault Analysis can break many additional secret key cryptosystems, including IDEA[9], RC5[19] and Feal[21,16,14,15]. Some ciphers, such as Khufu[13], Khafre[13] and Blowfish[20] compute their S boxes from the key material. In such ciphers, it may be even possible to extract the S boxes themselves, and the keys, using the techniques of Differential Fault Analysis. Differential Fault Analysis can also be applied against stream ciphers, but the implementation might differ in some technical details from the implementation described above. At this point we should note that small differences in the fault models might crucially affect the capabilities and the complexities of the attacks.

Differential fault analysis is not limited to finding the keys of known ciphers: We introduce an asymmetric fault model which makes it possible to find the secret key stored in a tamper-resistant cryptographic device even when nothing is known about the structure and operation of the cryptosystem. A prime example of such a scenario is the SkipJack cryptosystem, which was developed by the NSA, has unknown design, and is embedded as a tamper-resistant chip inside the commercially available Fortezza PC cards. We have not tested this attack on SkipJack, but we believe that it is a realistic threat against some smartcard applications which were not specifically designed to counter it.

Moreover, we show that in most interesting cases we can extract the exact structure

Copyright (c) 1998, Springer-Verlag

of an unknown DES-like cipher sealed in the tamper-resistant device, including the identification of its round functions, S boxes, and subkeys. If the attacker can only encrypt with the tamper-resistant device with a fixed key (e.g., in PIN verifying devices), still the attacker can identify the operation of the cipher with this particular key, which lets him encrypt and decrypt under this key.

The transient fault model used in all these attacks has not been demonstrated in physical experiments, and was questioned by representatives of the smartcard industry. We thus suggest a practical attack based on a different fault model which will hopefully be less controversial. In this model we cut one wire or permanently destroy a single memory cell in the smartcard using a narrow laser-beam. This model allows us to mount a *pure* ciphertext only attack which finds the key with only a few ciphertexts generated from random unrelated unknown plaintexts. The attack is thus applicable even when the smartcard chooses the message it wants to encrypt, and even it if uses counters or random bits to foil differential attacks on related plaintexts.

2 The Attack on DES

Todays' computers are extremely reliable. Still, it is possible for interested parties to intentionally induce faults into some kinds of computations. Although this is almost impossible to do to a remote computer or to a mainframe, it can be done to smartcards.

In many applications (like electronic money, identification systems, or access control) smartcards are used as secure extensions of the host, and enable their owners to apply cryptographic computations without knowing the hosts' secret keys. It is assumed that the smartcards are tamper-resistant, and thus even the owner of a smartcard cannot open it or reverse-engineer it in order to reveal the secret keys kept inside the smartcard.

However, due to the simplicity of the smartcards, and the ability of its owner to control the environment, the owner of the smartcard can force the smartcard to malfunction in many ways, such as changing the power supply voltage, changing the frequency of the (external) clock, and applying radiation of many kinds.

Boneh, Demillo, and Lipton suggested using faults induced by the card owner in order to deduce the private keys in number theoretic public key cryptosystems.

We use the following fault model, similar to the model used by Boneh, Demillo, and Lipton: The smartcard is assumed to have random transient faults in its registers, with some small probability of occurrence in each bit, so that during each encryption/decryption there appears a small number of faults (typically one) during the computation, and each such fault inverts the value of one of the bits, either from zero to one or from one to zero.

More accurately, we assume that during each faulty computation there occurs one (or a few) faults, at random times during the computations, and with random choice of the registers and positions in the registers. Both the bit location and the exact timing of the fault are unknown to the attacker.

This model lets the attacker induce errors at some random position during the DES encryption, i.e., at some random bit of one of the registers in a random round. For

Copyright (c) 1998, Springer-Verlag

the sake of simplicity we assume that the only affected registers are those keeping the right half of the data (the registers keeping the left half do not affect the results till they are exchanged with the right half), i.e., we assume that faults may occur in one of the $16 \cdot 32 = 512$ bits of the right halves of the 16 rounds, and the fault position is unknown to the attacker.

In the attack, the attacker uses the smartcard to encrypt some (possibly unknown) plaintext twice. The attacker compares the two results, and if they differ, he assumes that a fault occurred during exactly one of the two encryptions. As a result, he obtains two ciphertexts derived from the same (unknown) plaintext and key, where one of the ciphertexts is correct and the other is the result of a computation corrupted by a single bit error during the computation.

In the first step of the attack we identify the round in which the fault occurred. This identification is very simple and effective: If the fault occurred in the right half of round 16, then only one bit in the right half of the ciphertext (before the final permutation) differs between the two ciphertexts. The left half of the ciphertext can differ only in output bits of the S box (or two S boxes) to which this single bit enters, and the difference must be related to non-zero entries in the difference distribution tables[4] of these S boxes. In such a case, we can guess the six key bits of each such S box in the last round, and discard any value which disagrees with the expected differences of these S boxes. On average, about four possible 6-bit values of the key remain for each active S box.

If the faults occur in round 15, we can gain information on the key bits entering more than two S boxes in the last round: the difference of the right half of the ciphertext equals the output difference of the F function of round 15. We guess the single bit fault in round 15, and verify whether it can cause the expected output difference, and also verify whether the difference of the right half of the ciphertext can cause the expected difference in the output of the F function in the last round (i.e., the difference of the left half of the ciphertext XOR the fault). If successful, we can discard possible key values in the last round, according to the expected differences. We can also analyse the faults in the 14'th round in a similar way. We use counting methods in order to find the key. In this case, we count for each S box separately[1], and increase the counter by one for any pair which suggests the six-bit key value by at least one of its possible faults in either the 14'th, 15'th, or 16'th round. The right value of the key is expected to be counted more frequently than any wrong value, and thus can be identified.

We have implemented the algorithmic part of this attack on a personal computer. Our analysis program found the whole last subkey given between 50 and 200 ciphertexts, by simulating random single-faults in all the rounds.

This attack finds the last subkey. Once this subkey is known, we can proceed in two ways: We can use the fact that this subkey contains 48 out of the 56 key bits in order to guess the missing 8 bits in all the possible $2^8 = 256$ ways. Alternatively, we can use our knowledge of the last subkey to peel off the last round (and remove faults that we already identified), and analyse the preceding rounds with the same data using

[1] Counting on more than one consecutive S box should reduce the number of required ciphertexts.

Copyright (c) 1998, Springer-Verlag

the same attack. This latter approach makes it possible to attack triple DES (with 168 bit keys), or DES with independent subkeys (with 768 bit keys).

This attack still works even under more general assumptions on the fault locations, such as faults inside the function F, more than one fault during encryption, or even faults in the key or the key scheduling algorithm.

To check these claims, we have implemented a variant of this attack in which the faults may occur also in the inputs of the F function, rather than only in the right half of the data (i.e., the faults do not affect the following rounds directly), and found that the number of ciphertexts required to find the key is about the same as in the original attack, although the faults may occur in twice as many positions.

If the attacker can induce the faults in a chosen position or a chosen time during encryption, he can improve his results by a large factor. For example, if the attacker can cause the faults to appear uniformly in the last two, three, or four rounds of the DES encryption (rather than in all the 16 rounds), our attack requires only about 10 ciphertexts! If the attacker can choose the exact position of the fault, this number can be further reduced to about 3 ciphertexts.

2.1 Discussion

We described a new attack on ciphers using transient hardware faults. Smartcard designers can try to counter this attack by computing the encryption function twice, and outputting the ciphertext only if the results are identical. This solution is however insufficient: the probability that the same fault occurs during both encryptions may not be sufficiently small. In the attack there are only 512 possible positions for a fault. When computing twice with a single random fault in each of the two encryptions, there is a probability of $1/512$ to generate the same fault in both computations. Since the device outputs the doubly-corrupted results, the attacker receives the same data as in the original attack, if he tries 512 as many encryptions. Thus, instead of generating about 200 ciphertexts in the original attack, the device performs about 100000 encryptions.

In many cryptographic implementations, the key scheduling algorithm precomputes all the subkeys in advance, or computes the subkeys from the key every single encryption. In such cases, it is easy to find ciphertext pairs whose differences result only from one faulty subkey bit, when the faults affect the subkeys. In the attack we should assume that the difference is caused by one faulty subkey bit, or by several subkey bits caused by one fault during the key scheduling algorithm. The number of ciphertexts required for such analysis is expected to remain about the same as in the attack we described.

DFA can also be combined with other types of cryptanalytic attacks: for example, when the cipher is computed by software in the device, the faults might affect the program counter or the loop index. In such cases, we can apply more or fewer rounds than required.

In some implementations, the DES key scheduling is applied with two 28-bit shift registers, C and D, as in its original definition. The key rotates every round, and is restored to its original state at the end of encryption, since the total number of

Copyright (c) 1998, Springer-Verlag

shifts during the 16 rounds is 28. If the faults affect the shifts of these registers, then in the following encryptions the key is changed to a *related* key. Related key cryptanalysis[3], or differential related key cryptanalysis[7] might be applied with DFA in such cases. We expect that linear cryptanalysis[12] can also be combined with DFA in some cases (in a similar way to differential-linear cryptanalysis[10]), especially when the identification of the fault position is highly reliable (or when the fault positions might be chosen by the attacker).

Variants of DFA attacks can in some cases also derive the keys of modes of operation in which only part of the ciphertext is known to the attacker. This is similar to the situation studied in [18] for differential cryptanalysis of the CFB mode of operation.

3 Breaking Unknown Cryptosystems

In this section, we introduce a variant of DFA that can find the secret keys of unknown cryptosystems, even if they are sealed inside tamper-resistant devices, and nothing is known about their design. In this attack, we assume a slightly different fault model: the main assumption behind this fault model is that the cryptographic key is stored in an asymmetric type of memory, in which induced faults are much more likely to change a one bit into a zero than to change a zero bit into a one (or the other way around). CMOS registers seem to be quite symmetric, but most types of non-volatile memory exhibit some degree of asymmetry. For example, a one bit in an EEPROM cell is stored as a small charge on an electrically isolated gate. If the fault is induced by external radiation (e.g., ultraviolet light), then the charges are more likely to leak out of the gate than to be forced into the gate.

To make the analysis simpler, we assume that we can apply a low level physical stress to the tamper-resistant device when it is disconnected from power, whose only possible effect is to occasionally flip one of the one bits in the key register to a zero. The plausibility of this assumption depends on numerous physical and technical considerations, which are beyond the scope of this paper.

We further assume that we are allowed to apply two types of cryptographic functions to the given tamper-resistant device: We can supply a plaintext m and use the current key k stored in the non-volatile memory of the device to get a ciphertext c, or we can supply a new n-bit key k' which replaces k in the non-volatile memory.

The cryptanalytic attack has two stages: In the first stage, we keep the original unknown secret key k stored in the tamper-resistant device, and use it to repeatedly encrypt a fixed plaintext m_0. After each encryption, we disconnect the device from power and apply a gentle physical stress. The resultant stream of ciphertexts is likely to consist of several copies of c_0, followed by several copies of a different c_1, followed by several copies of yet another c_2, until the sequence stabilizes on c_f. Since each change is likely to be the result of one more key bit flipping from one to zero (thus changing the current key k_i into a new variant k_{i+1}), and since there are about $n/2$ one bits in the original unknown key k, we expect f to be about n/2, and c_f to be the result of encrypting m_0 under the all-zero key k_f.

In the second stage of the attack, we work our way backwards from the known

Copyright (c) 1998, Springer-Verlag

all-zero key k_f to the unknown original key k_0. Assuming that we already know some intermediate key k_{i+1}, we assume that k_i differs from k_{i+1} in a single bit position. If we knew the cryptographic algorithm involved, we could easily try all the possible single bit changes in a simple software simulation on a personal computer, and find the (almost certainly unique) change which would give rise to the observed ciphertext c_i. However, we do not need either a simulator or knowledge of the cryptographic algorithm, since we are given the real thing in the form of a tamper-resistant device into which we can load any key we wish, to test out whether it produces the desired ciphertext c_i. We can thus proceed deterministically from the known k_f to the desired k_0 in $O(n)$ stages, trying $O(n)$ keys at each stage. The attack is guaranteed to succeed if the fault model is satisfied, and its total complexity is at most $O(n^2)$ encryptions.

This seems to be the first cryptanalytic attack which makes it possible to find the secret key of a completely unknown cryptosystem in polynomial time (quadratic time in our case). It relies on a particular fault model which is stronger than the transient fault model described above, and which has not been experimentally verified so far. In the full version of this paper we'll discuss numerous extensions of the attack, including the analysis of more complicated fault models in which the sequence of corrupted keys forms a biased random walk in the space of 2^n possible keys.

4 Reconstructing Unknown Ciphers

In this section we show how to reconstruct the full structure of unknown ciphers hidden in tamper-resistant devices. We assume that the attacked cipher is a DES-like cipher, which encrypts by applying some initial permutation and then applying a round function several times. The round function applies some function F to the right half of the data, XORs the result to the left half, and exchanges the roles of the halves. Finally, some final permutation is performed.

In our attack we reconstruct the cipher in several steps. In each step we receive some additional information on the unknown structure of the cipher: we start from the final permutation and continue backwards through the rounds.

Note that the representation of the ciphers is not unique, and thus we cannot identify the exact original definition of the cipher. Instead, we actually find a family of representations, of which the original representation is a member. We can then choose any member of the family as an equivalent representation of the cipher.

In the first step of the attack we study some information on the final permutation: We identify which ciphertext bits come from the right half and which from the left half of the last round, i.e., which bits affect which in the last round. We encrypt several plaintexts several times, and collect pairs of ciphertexts consisting of the real ciphertext of the plaintext and a faulty ciphertext resulting from some fault during encryption of the same plaintext. We identify the faults which occur in the last round (or two) by counting the bits differing between the real ciphertext and the faulty ciphertext. We use the pairs in which the number of differing bits in the ciphertexts is smaller than a threshold (typically about a quarter of the blocksize), in which case, it is almost certain that the fault occurred in the last round, or in the preceding one.

Copyright (c) 1998, Springer-Verlag

Each fault in the last round differs in one bit in the right half and several bits in the left half. Therefore, the bits of the left half differ more frequently than the bits of the right half. We can thus identify the bits of the right half as those which differ in the least number of pairs.

Once we identify the bits of the right half and the bits of the left half, we can observe which bits of the left half are affected by each bit of the right half via the function F in the last round. In DES-like ciphers the F function is composed of S boxes, and each S box takes a few of the bits of the right half, and affects a few of the bits of the left half. We can easily identify the number of S boxes, and the input and output bits of each S box: we just choose all the pairs which differ by only one bit of the right half, and for each such bit we find the set of all the bits of the left half which differ in those pairs. This information suffices to find the number of S boxes, their sizes, and to identify the input and output bits of each S box.

Then, we reconstruct the content of the S boxes, with the specific unknown key mixed into the input of the S boxes and some unknown value mixed into their output (i.e., we can identify the table $T(x)$, where $T(x) = S(x \oplus k) \oplus u$, where k is the (actual) subkey, and u is derived from the right half of the preceding round). This can be done since the pairs give us the difference $T(x) \oplus T(y)$, for two known values x and y. This reconstruction is very effective. For example, if the unknown cipher is DES, we miss information on only $6 + 4 = 10$ bits out of the $64 * 4 = 256$ unknown bits of each S box, and if the unknown cipher is LOKI[6], we miss information on only $12 + 8 = 20$ bits out of the $2^{12} * 8 = 32768$ bits of each S box of LOKI. These missing bits do not reduce the success of the attack since we actually find all the information we need for peeling off the last round: the subkeys are already mixed into the S boxes and the extra constants are counted naturally as parts of the subkeys when analyzing the preceding rounds. This way we can fully analyze the whole cipher, and receive its full description with the specific key mixed into the S boxes.

Till now we identified the S boxes up to XOR with some unknown constants, some of which are subkeys. We can further identify the S boxes and the subkeys by analyzing encryptions under several keys and comparing the differences between the retrieved tables T. In DES and LOKI the key scheduling algorithm and the S boxes can be easily identified by such comparisons. In these ciphers, where the key scheduling actually select key bits into the subkey (rather than making a more complex computation), the selection pattern of the key bits can be identified by analyzing the effect of different keys on the T boxes. Then, the effect of the subkeys on the T boxes can be removed, resulting with the original S boxes and the key scheduling algorithm.

In some ciphers, the function F is not a composition of several S boxes as in DES and LOKI, but may compute more complex operations. In this case we can model the function F by a most general structure: for each possible subkey, the F function is modeled as an arbitrary function. In this case we can view the F function as applying one huge S box. The number of input and output bits of this huge S box is only half the block size, and thus even though the identification of this whole S box requires much work (about 2^{36} in the case of a 16-round 64-bit DES-like cipher), it can still be done for any particular key. The effect of the key on this S box can be removed in most cases after repeating this computation for several keys.

Copyright (c) 1998, Springer-Verlag

We have implemented major parts of this attack on a personal computer, using DES as the unknown cipher. Our implementation required only about 500 faulty ciphertexts to identify the bits of the right half, and up to 5000 faulty ciphertexts to identify the input and output bits of the S boxes, without reconstructing their actual entries. Their complete reconstruction would require about 10000 faulty ciphertexts. Note that the complexity of this S box reconstruction crucially depends on the size of the S boxes: in DES there are 64 entries in each S box, and thus about 64 faults in the input of an S box in different ciphertexts suffice to reconstruct the S box (except the 10 missing bits). In LOKI there are 4096 entries in each S box, and thus the number of required faulty ciphertexts is much larger. If we view the F function as one large S box, the number of entries of this huge S box is 2^{32}, and thus the number of required faulty ciphertexts in this case is huge, but still practical.

5 Non-Differential Fault Analysis

The main criticism against differential fault analysis was the transient fault model that was claimed to be unrealistic. Although we still believe that an attack based on radiation-induced faults is possible, we decided to devise a more practical fault model that will hopefully be less controversial.

The fault model considered in this section assumes that we can cut a wire or destroy a memory cell in a register using a narrow laser beam directed into a carefully chosen location in the silicon. As a result, during computation the values entering the affected location can be considered to be permanently stuck at a fixed value, and can no longer affect the rest of the computation. A similar fault model is considered in [2] where it is used in conjunction with micro-probing to mount other attacks on smartcard-based cryptosystems.

5.1 The Basic Attack

Based on this fault model we develop a *pure* ciphertext only attack, which, unlike all other ciphertext only attacks, does not make any assumption on the statistical distribution of the plaintexts. Moreover, the ciphertext should only be received after the permanent fault had already been generated; non-faulty ciphertexts are not required at all in this attack. Therefore, this attack will still work if faulty smartcards, which already have the appropriate faults due to natural reasons, are given to the attacker.

We describe this attack against a smartcard implementation of DES. We assume that DES is implemented in hardware as a single round, which is used 16 times as is described in Figure 1. We generate a permanent fault in the least significant bit of the left-half register, either by destroying the bit or by cutting the wire which enters or leaves this cell, and assume that the value of this bit is permanently stuck at zero. Figure 2 describes the values computed by this iterated implementation in the last round.

For any ciphertext the least significant bit (LSB) of the right half of the ciphertext (L_{16}) is zero, and the LSB of L_{15} is also zero. The LSB of the output of the F-function

Copyright (c) 1998, Springer-Verlag

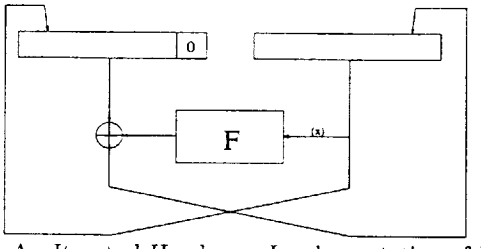


Figure 1. An Iterated Hardware Implementation of DES.

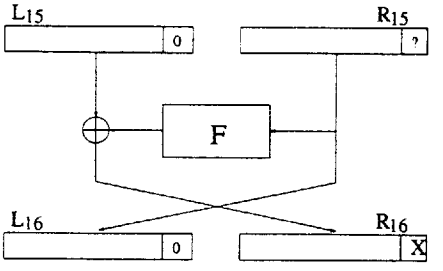


Figure 2. The Last Round in the Iterated Implementation of DES.

of the last round equals the LSB of the left half of the ciphertext (X, since the LSB of L_{15} is zero). This bit is the output of S7 in the last round. The input of S7 is formed by XORing 6 known ciphertext bits with six key bits. We can try all the 64 possible values of the six key bits, and discard any value which does not predict the LSB of the right half of the ciphertext. Each ciphertext is used to discard about half of the remaining key values. Thus, given about six ciphertexts (on whose plaintexts the attacker has no information of any kind) the right value of the six key bits can be identified. The other bits of the last subkey can be found by inducing additional faults, or by enumerating and verifying additional key bits.

5.2 Attacks on Unrolled Implementations

The attack becomes easier if DES is implemented as 16 separate hardware rounds. In this case, a permanent fault induced in one round would not directly affect the other rounds. In this attack, it suffices to destroy the LSB of register L_{15}. In particular, if we destroy the whole L_{15} in this case, we can find the key with only two ciphertexts. We can even reduce the number of ciphertexts required to find 32 bits of the last subkey uniquely to exactly one (rather than only in average) by also destroying the first and sixth input bits entering the S boxes (after they are XORed with the subkey). In this case, the F function becomes reversible, and thus its remaining inputs can be computed uniquely from the outputs.

In an alternative attack, we can destroy all the bits of the subkey registers, except in the first two rounds. Given the ciphertext, remove the last 14 rounds by decrypting

Copyright (c) 1998, Springer-Verlag

with zero subkeys. Equivalently, we can destroy the outputs of the F-function in rounds 3-16, so that the outputs become zero, or even cut the cipher after the 14'th round. In all these cases, the problem is reduced to attacking a two-round DES, which can be done using one ciphertext.

We can even attack the independent-key variant of any iterated cipher. We simply remove a round at a time and prepare the required encrypted data for later analysis. After all the rounds are removed, we find the subkey of the last removed round using the prepared data, and use this found subkey in attacking the second-to-last removed round, etc., till we find all the subkeys. Note that we get two equations on each F-function from two consecutive removed rounds, and thus we get sufficient information for finding the 48-bit subkeys, although each equation leaks only 32 bits.

5.3 Additional Attacks on Iterated Implementations

All the above attacks find the key bits *horizontally*, i.e., a subkey at a time. In iterated hardware implementations we can do the following *vertical* attack which finds the first bits of all the subkeys, then the second bits of all the subkeys, etc. In this attack we first permanently destroy the plaintext to be zero, and encrypt the zero plaintext under the unknown key. In the main step of the attack, we cut the wire leaving the last bit of the subkey in the iterated hardware implementation of the rounds, and encrypt the zero plaintext under the resultant subkeys. Then, we cut the second-to-last wire, etc. When all the subkey bits had been cut, we get the zero plaintext encrypted under the zero subkeys. We can now find the first bits of the subkeys (total of 16 bits in a 16-round cipher) by trying all their possible values (2^{16} trials), and comparing the resulting ciphertexts to the last received ciphertext. Then, we try the values of the second bits of the subkeys, etc, till we find all the bits of the subkeys. For DES in particular, we can use the fact that the key is divided into two halves. We can cut the first 23 bits of the subkey before they are XORed to the right half of the data. Then, we cut the 24'th bit. Then again we cut the next 23 bits. Given the ciphertexts of the destroyed (i.e., 0) plaintext before and after each of these events (four ciphertexts), we can exhaustively search for the 16 key bits which still affect the computation after the last event. When found, we can complete the 12 key bits of the right half of the key, and then search for the 16 key bits of the right half which affect the encryption after the first event. Finally, we complete the remaining 12 key bits.

An additional attack against iterated implementations of DES ignores the data rather than the key. In this attack we cut the 32-bit data input of the F-function (this position is denoted by (x) in Figure 1), such that the output of the F-function becomes dependent only of the subkey and independent of the plaintext. If the actual plaintext is unknown, we also destroy the plaintext register, i.e., set it to zero permanently. The key scheduling algorithm of DES divides the key into two halves, each of them affects only four S boxes in each round (S1–S4, and S5–S8). As a result, 32 bits of the ciphertext depend only on one 28-bit half of the key, and the other 32 bits of the ciphertext depend only on the other 28-bit half of the key. Thus, given the ciphertext of the zero plaintext (or similarly of any plaintext) under the modified cipher, we can easily search for the two halves of the key (2^{28} trials for each, each trial costs about 1/4 encryption).

Copyright (c) 1998, Springer-Verlag

5.4 Remark

Note that in this model it is easy to apply a chosen plaintext attack without choosing any plaintext — we just destroy the plaintext register. In such a case, even a fault tolerant design in which the smartcard encrypts the plaintext several times and compares the results, will not detect any difference between the ciphertexts.[2] Moreover, if the verification is done by decrypting the ciphertext, the result register can also be destroyed, and thus will always be the same as the plaintext register.

6 Acknowledgements

We would like to gratefully acknowledge the pioneering contribution of Boneh, Demillo, and Lipton, whose ideas were the starting point of our new attack.

References

[1] Ross Anderson, Markus Kuhn, *Tamper Resistance - a Cautionary Note*, proceedings of the Second Usenix Workshop on Electronic Commerce, pp. 1–11, November 1996.

[2] Ross Anderson, Markus Kuhn, *Low Cost Attacks on Tamper Resistant Devices*, proceedings of the 1997 Security Protocols Workshop, Paris, April 7–9, 1997.

[3] Eli Biham, *New Types of Cryptanalytic Attacks Using Related Keys*, Journal of Cryptology, Vol. 7, No. 4, pp. 229–246, 1994.

[4] Eli Biham, Adi Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

[5] Dan Boneh, Richard A. Demillo, Richard J. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'97, pp. 37–51, 1997.

[6] Lawrence Brown, Josef Pieprzyk, Jennifer Seberry, *LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of AUSCRYPT'90, pp. 229–236, 1990.

[7] John Kelsey, Bruce Schneier, David Wagner, *Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'96, pp. 237–251, 1996.

[2] Designers should be very careful with such designs, since under some fault models, the comparison of the results might leaks information about the key, which wouldn't be leaked otherwise.

Copyright (c) 1998, Springer-Verlag

[8] Paul C. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'96, pp. 104–113, 1996.

[9] Xuejia Lai, James L. Massey, Sean Murphy, *Markov Ciphers and Differential Cryptanalysis*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'91, pp. 17–38, 1991.

[10] Susan K. Langford, Martin E. Hellman, *Differential-linear cryptanalysis*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'94, pp. 17–25, 1994.

[11] John Markoff, *Potential Flaw Seen in Cash Card Security*, New York Times, September 26, 1996.

[12] Mitsuru Matsui, *Linear Cryptanalysis Method for DES Cipher*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'93, pp. 386–397, 1993.

[13] Ralph C. Merkle, *Fast Software Encryption Functions*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'90, pp. 476–501, 1990.

[14] Shoji Miyaguchi, *FEAL-N specifications*, technical note, NTT, 1989.

[15] Shoji Miyaguchi, *The FEAL cipher family*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'90, pp. 627–638, 1990.

[16] Shoji Miyaguchi, Akira Shiraishi, Akihiro Shimizu, *Fast Data Encryption Algorithm FEAL-8*, Review of electrical communications laboratories, Vol. 36, No. 4, pp. 433–437, 1988.

[17] National Bureau of Standards, *Data Encryption Standard*, U.S. Department of Commerce, FIPS pub. 46, January 1977.

[18] Bart Preneel, Marnix Nuttin, Vincent Rijmen, Johan Buelens, *Cryptanalysis of the CFB Mode of the DES with a Reduced Number of Rounds*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'93, pp. 212–223, 1993.

[19] Ronald L. Rivest, *The RC5 Encryption Algorithm*, proceedings of Fast Software Encryption, Leuven, Lecture Notes in Computer Science, pp. 86–96, 1994.

[20] Bruce Schneier, *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*, proceedings of Fast Software Encryption, Cambridge, Lecture Notes in Computer Science, pp. 191–204, 1993.

[21] Akihiro Shimizu, Shoji Miyaguchi, *Fast Data Encryption Algorithm FEAL*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'87, pp. 267–278, 1987.

Copyright (c) 1998, Springer-Verlag

A Survey of Differential Fault Analysis against Classical RSA Implementations

Alexandre Berzati, Cécile Canovas-Dumas, and Louis Goubin

1 Introduction

Since the advent of side channel attacks, classical cryptanalysis is no longer sufficient to ensure the security of cryptographic algorithms. In practice, the implementation of algorithms on electronic devices is a potential source of leakage that an attacker can use to completely break a system [29, 15, 21]. The injection of faults during the execution of cryptographic algorithms is considered as an intrusive side channel method because secret information may leak from malicious modifications of a device's behavior [13, 11]. In this context, the security of public key cryptosystems [13] and symmetric ciphers in both block [11] and stream modes [23] has been challenged. Recently, some interesting results have been obtained by attacking public key cryptosystems. More precisely, several papers demonstrated that the perturbation of public elements may induce critical flaws in implementations of public key cryptosystems [10, 14, 25].

We propose here a survey of the applications of fault attacks against different RSA implementations, classical or sophisticated. After a presentation of the RSA cryptosystem and its classical implementation, we review chronologically the attacks and some of the proposed countermeasures. Although first attackers focused their efforts to exploit perturbations of secret elements or related operations (see Sect. 3), recent works adressed the security of public elements (see Sect. 4). This new trend is all the more interesting since public elements are usually handled in a less secure way than private ones. Furthermore it may lead to powerful attacks regardless to the kind of induced perturbations.

Alexandre Berzati · Cécile Canovas-Dumas
CEA Leti, Grenoble, France

Louis Goubin
Versailles Saint-Quentin-en-Yvelines University, France

2 RSA Implementations

The celebrated *RSA* has been invented in 1977 by Ronald Rivest, Adi Shamir and Leonard Adleman [36]. This method has been the first one instantiating the principle of public key cryptography, introduced earlier, as a concept, by Whitfield Diffie and Martin Hellman [20]. Nowadays, RSA is certainly the most widely used algorithm to ensure the security of communications or transactions.

The RSA security relies on the Integer Factorization Problem (IFP). The best known method for solving an instance of this problem is the Number Field Sieve which complexity is sub-exponential with respect to the size of the RSA field [30]. To the best of our knowledge, the larger RSA modulus ever factored by this method is a 768-bit one [26]. As a consequence, this method can not be used today to factor practical sized moduli (i.e. 1024 or 2048 bits).

In practice, the RSA can be declined in both standard and CRT modes. The standard mode is the straightforward way for implementing RSA. Its principle is recalled below. The CRT mode of RSA, where CRT stands for *Chinese Remainder Theorem*, is usually used to perform efficient signatures in embedded systems. Further details about CRT-RSA implementations can be found at [35, 31].

2.1 Standard RSA

2.1.1 Key Generation.

Let N, the public modulus, be the product of two large prime numbers p and q. The length of N, denoted by n, also stands for the RSA length. Let e be the *public exponent*, coprime to $\varphi(N) = (p-1)\cdot(q-1)$, where $\varphi(\cdot)$ denotes Euler's totient function. Then the pair $K_p = (N, e)$ is the RSA public key, that is spread over a network. The public key exponent e is linked to the *private exponent* d by the following equation:

$$e \cdot d \equiv 1 \mod \varphi(N) \tag{1}$$

This exponent is also called private key K_s, that is kept secret by its owner.

2.1.2 RSA Encryption.

Let m be the plaintext that will be ciphered with the RSA algorithm. Then, the cipher operation relies on the following modular exponentiation involving the public exponent e:

$$C \equiv m^e \mod N \tag{2}$$

Here C denotes the ciphertext. After sending this cyphertext throughout a network, the expected receiver of the message may want to recover the original message by decrypting C. Then, this operation boils down to compute:

$$\tilde{m} \equiv C^d \mod N \tag{3}$$

If no error occurs during computation, transmission or decryption of C, then we expect to get:

$$\tilde{m} \equiv m \mod N \tag{4}$$

One can notice that performing a decryption implies the knowledge of the private key. This ensures that only its owner, and so the expected receiver, is able to recover m.

2.1.3 RSA Signature Scheme.

As for the decryption, the RSA signature S of a message m consists in a modular exponentiation to the power d:

$$S \equiv m^d \mod N \tag{5}$$

To check the validity of the received signature (S,m), the associated public key is used to ascertain that:

$$m \stackrel{?}{\equiv} S^e \mod N \tag{6}$$

In general, this is not really the plaintext m that is signed but a hash $\dot{m}$. This operation is performed by using a hash function $\mathscr{H}$ such that $\dot{m} = \mathscr{H}(m)$. Then, the sent signature is $(S,\dot{m})$ where $S \equiv \dot{m}^d \mod N$. Furthermore, the usage of a hash function is generally combined with a padding scheme (*e.g.* RSA-OAEP [3] for the RSA encryption and RSA-PSS [4] for the signature). For the different fault attacks presented in this paper, we assume that all RSA decryptions or signatures are performed with some hash and/or deterministic padding function.

2.2 Modular exponentiation methods

Modular exponentiation is one of the core operations to implement asymmetric cryptography. Indeed for both RSA decryption and RSA signature schemes, one has to compute a modular exponentiation of the input message as efficiently as possible. For a naive implementation of this operation, we may write:

$$m^d \mod N \equiv \underbrace{m \cdot m \cdot \ldots \cdot m}_{\text{d times}} \mod N \tag{7}$$

The computational complexity of the algorithm above is exponential with respect to the exponent length, which is not acceptable to implement the modular exponentiation. However, a method as intuitive as the previous one, but much smarter, consists in expressing the exponent in a binary basis: $d = \sum_{i=0}^{n-1} 2^i \cdot d_i$. Then, we can perform a modular exponentiation by scanning the bits of the exponent:

$$m^d \mod N \equiv \prod_{i=0}^{n-1} m^{2^i d_i} \mod N \tag{8}$$

As a consequence, the power is no longer performed by multiplying d times a message m by itself but by multiplying, at most $log_2(d)$ times, square powers of m. Hence the cost of the exponentiation methods declined from this principle, also called *binary exponentiations*, is linear with respect to the exponent length which is much more efficient than the naive approach. The next section briefly introduces common implementations of binary exponentiations

2.2.1 *"Right-To-Left"* Exponentiation.

The first method to perform a modular exponentiation is to scan the exponent bits from least to most significant ones. That is why it is also referred as the *"Right-To-Left"* method. In practice, this method is the most intuitive since it consists in computing consecutive square powers of the input message and, depending on the current exponent bit value, multiplying these powers to the accumulation register. As a consequence, this algorithm exactly transcripts (8). The complete algorithm is detailed below.

Algorithm 1: *"Right-To-Left"* Modular Exponentiation

Input: $m, d = \sum_{i=0}^{n-1} 2^i \cdot d_i, N$
Output: $S = m^d \mod N$
$A \leftarrow 1$;
$B \leftarrow m$;
for $i = 0$ **to** $n-1$ **do**
  if $d_i = 1$ **then**
    $A \leftarrow A \cdot B \mod N$;
  end
  $B \leftarrow B^2 \mod N$;
end
return A;

2.2.2 *"Left-To-Right"* Exponentiation.

It is also possible to scan the exponent bits from most to least significant bits. The associated method is not surprisingly called *"Left-To-Right"* exponentiation. This other method differs from the dual one by an accumulation register withdrawal and a different execution flow. Indeed, each iteration begins with the execution of a square that can be followed, depending on the current exponent bit value, by a multiplication. This method is also detailed in Algorithm 2.

Algorithm 2: *"Left-To-Right"* Modular Exponentiation

Input: $m, d = \sum_{i=0}^{n-1} 2^i \cdot d_i, N$
Output: $S = m^d \bmod N$
$A \leftarrow 1$;
for $i = (n-1)$ **to** 0 **do**
  $A \leftarrow A^2 \bmod N$ **if** $d_i = 1$ **then**
    $A \leftarrow A \cdot m \bmod N$;
  end
end
return A;

2.2.3 Other variants.

Although both previously presented algorithms are quite efficient to compute modular exponentiations, their implementation may leak information on the exponent. Indeed, the execution of a multiplication directly depends on the current value of the exponent bits. This makes smart card implementation of modular exponentiation algorithm potentially vulnerable to side channel attacks, such as DPA. A basic solution to thwart this flaw is to make the execution independent from the exponent value. Such a variant has been already proposed by J.-S. Coron at CHES 1999 [19] and is also known as the *Square & Multiply Always* algorithm. In this case, one square and one multiplication are sequentially executed at each iteration, whatever the value that the current exponent bit may take.

The performance of the exponentiation algorithms can also be improved. For example, we can derive from the previous algorithms the *Sliding-Window Exponentiation* used in the `OpenSSL` library. The principle of this variant is to scan the exponent by, at most, k-bit windows. Hence, after precomputing some odd powers of the input message m, this method significantly reduces the number of iterations for a modular exponentiation, and so the performance.

Finally, some variants allow to improve both security and performance. This is the case of the Montgomery exponentiation algorithm [32]. Obviously, it exists other efficient implementations of the modular exponentiation, but we advise an interested reader to take a look at [27] for more details.

3 Classical Fault Analysis of Standard RSA Implementations

Since the introduction of fault attacks at the end of the nineties, the security against perturbation of CRT-based implementation of RSA has not been the sole implementation mode targeted [13]. Indeed, the security of standard RSA implementations have been also challenged, leading to various attack methodologies. Among these fault attacks, we have chosen to distinguish to main categories. The first one deals

with the perturbation of intermediate computations. The fault attacks that belong in this category take advantages of the perturbation of intermediate values or of the execution flow. The second category we have distinguished gather attacks based on exploiting modifications of RSA public elements. This trend is quite recent but has ever led to successful applications against various standard RSA implementations.

The next paragraph details the different attacks that we have identified from our state-of-the-art study.

3.1 Perturbation of intermediate computations

3.1.1 Register faults.

Bellcore researchers not only introduced the concept of fault attacks [5] but they also showed how this new concept can be applied to many public key cryptosystems, including standard RSA, and their various implementations. In more details, they explained in [13] how to exploit fault injections during the execution of a standard RSA signature to recover the private exponent. The fault model they considered, the so-called *register fault*, is a transient or permanent bit-flip induced in the memory area containing the current value of the exponentiation algorithm. Under this model, they showed that the perturbation of standard RSA signature, implemented with the *"Right-To-Left"* exponentiation, may leak some secret information. The principle of the attack is described in the next paragraph

General Methodology.

The fault attack against standard RSA signature proposed by Bellcore researchers can be split in two parts. The first one is *"on-line"* and consists in gathering sufficiently many message/faulty signature pairs $(m_i, \hat{S}_i)$ by inducing permanent faults, one per faulty signature, on the register that contains a intermediate values. Then, in the second part, the attackers tries to analyze the collected faulty signatures to recover the whole secret key. Hence, this part of the attack is completely *"off-line"*. The principle of the analysis is recalled below. Let S_i be the correct signature and let $\varepsilon(m_i) = \pm 2^b$ be the mathematical representation of a bit-flip with $0 \leqslant b < n$. Since this fault is supposed to be permanent, then corresponding faulty signature can be expressed as:

$$\hat{S}_i \equiv \left(\left(\prod_{j=0}^{t-1} m_i^{2^j d_j} \right) \pm 2^b \right) \cdot \prod_{j=t}^{n-1} m_i^{2^j d_j} \tag{9}$$

$$\equiv S_i \pm 2^b \cdot \prod_{j=t}^{n-1} m_i^{2^j d_j} \mod N \tag{10}$$

$$\text{or } S_i \equiv \hat{S}_i \pm 2^b \cdot m_i^{d_{[t]}} \mod N \tag{11}$$

with $d_{[t]} = \sum_{j=t}^{n-1} 2^j d_j$. If we use the signature's verification operations, the previous equation becomes:

$$m_i \equiv (\hat{S}_i \pm 2^b . m_i^{d_{[t]}})^e \mod N \tag{12}$$

One can notice that this equation is interesting because it only depends on the message m_i and the corresponding faulty signature $\hat{S}_i$ (*i.e.* the knowledge of the correct signature is no longer required). Moreover, the fault injection has isolated a part $d_{[t]}$ of the private exponent d. This is precisely this part of exponent that the attacker has to determine with a *guess-and-determine* approach.

The whole exponent is gradually recovered, from most to least significant bits, by repeating the previous analysis on different faulty signatures. In each analysis, w bits of exponent are retrieved. In more details, for each analysis, the attacker has to simultaneously guess the values of the part of exponent isolated $d_{[t]}$ and the fault induced $\pm 2^b$ such that (12) is satisfied. According to [13], the whole private exponent d can be determined by this way, with a probability greater than $\frac{1}{2}$, from $(n/w)\ log\ (2n)$ message-signature pairs. In this case, the attack complexity is about $O((2^w n^3 log^2(n))/w^3)$ exponentiations. We can additionally remark that this fault attack was later generalized to *"Left-To-Right"*-based exponentiations by J. Blömer and M. Otto [34].

3.1.2 Faults on the private exponent.

This attack was published by F. Bao *et al.* in [1] and then in [2]. The principle is to induce a transient error during the decryption, that produces the same effect as a bit modification of the private exponent. In practice, such an effect can be obtained by flipping a bit of the private exponent, or by corrupting the evaluation made just before the conditional branch in the classical implementations of the modular exponentiation (see Sect. 2.2). The following paragraph only describes the attack for a bit error on the exponent d.

General methodology.

Let m be a plaintext and C the corresponding ciphertext obtained from a RSA encryption (see Sect. 2). In case of a faulty computation, the deciphered text $\hat{m}$ is:

$$\hat{m} \equiv C^{\hat{d}} \bmod N$$

The fault is exploited by dividing the erroneous result by a correct one: $\hat{m} \cdot m^{-1}$. The induced error can be modeled as a *bit-flip* of the t-th bit of d. Therefore, we have:

$$\hat{m} \equiv C^{\sum_{i=0, i\neq t}^{i=n-1} 2^i \cdot d_i + 2^t \bar{d}_t} \bmod N$$

That implies, either $\hat{m} \cdot m^{-1} \equiv C^{-2^t} \bmod N \Rightarrow d_t = 1$,
or $\hat{m} \cdot m^{-1} \equiv C^{2^t} \bmod N \Rightarrow d_t = 0$. This method can be repeated until we obtain enough information on the private exponent. Note that this attack is also suitable in case of a multiple error model [1]. Moreover the principle can be adapted to attack cryptosystems based on discrete logarithm (DSA, El-Gamal, ...). Finally, this attack strategy has been later extended and generalized by M. Joye *et al.* [24], who describe an improved attack relying on the mere knowledge of the faulty deciphered text.

3.1.3 Exploiting safe-errors.

Classical fault attacks often exploit the difference between a correct and a faulty output to deduce some secret information. However, M. Joye and S.-M. Yen noticed that some secret information may leak even if a fault causes no effect on the final result of the computation [38]. This is why this particular kind of fault attacks is also called *"safe-error"* attack. The attacker must be able to perform some perturbation of which he knows the probably effect with a good repeatability. Among these attacks, two categories are usually distinguished: *C-safe-error attacks* that target dummy operations [39] and *M-safe-error attacks* that target registers allocations [38]. In order to illustrate the principle of safe-error attacks in the context of RSA, we have chosen to detail the *C-safe-error attacks* against the *Square & Multiply Always* exponentiation [19].

General methodology.

The purpose of the *Square & Multiply Always* exponentiation is to make its execution independent from the exponent value. Hence, the conditional branch is withdrawn (see Sect. 2.2) and an extra dummy multiplication is added such that, regardingless of the value of the exponent, a square and a multiplication are always executed at each iteration. The principle of the *C-safe-error attacks* proposed by S.-M. Yen *et al.* [39] is to exploit faults induced while dummy multiplications are performed. To achieve this, the attacker has to inject a fault during a RSA signature and, if the signature remains *"error-free"*, it means that a dummy multiplication has been infected. Therefore, the attacker can deduce that the exponent bit value handled while the perturbation was provoked is zero. Otherwise, the result would be incorrect and the exponent bit value equals to one. The attacker has to repeat the attack at each iteration to gradually recover the whole private exponent. One can notice

that this attack does not apply upon classical exponentiation algorithm but upon a variant expected to defeat *Simple Power Analysis*. As a consequence, checking only the output may not be enough to protect and implementation of a cryptographic algorithm against faults. Furthermore, designers of secure solution may be careful not to add new vulnerability while trying to defeat other ones.

4 Exploiting Perturbations of RSA public modulus

Although the issue of exploiting malicious modifications of public elements was addressed in the context of Elliptic Curve based cryptosystems [16], it took a half decade before seeing the first application to RSA. Indeed, the first fault attack against public key elements is due to J.-P. Seifert with a method for corrupting RSA signature's verification [37, 33]. This fault attack aims to corrupt signature verification mechanism by modifying the value of the public modulus N. Nevertheless, no information about the private exponent d is revealed with this fault attack.

Whether it is necessary or not to protect RSA public elements was an open question until Brier *et al.* attack proposal for recovering the whole private key. This attack, inspired from Seifert's one [37, 33], was published in [14] and reviewed in [17]. It makes it possible to extract the private key using a modulus perturbation. As in Seifert's attack, the fault on the modulus is induced before executing the exponentiation. Hence, if the faulty modulus has a small divisor r, the attacker will be able to solve an instance of the Discrete Logarithm Problem from the corresponding faulty signature and obtain $d \mod r$.

A new fault attack against *"Right-To-Left"*-based implementations of the core RSA exponentiation [8], completed by the attack of the dual implementation [7] has been presented lately. Contrary to previous attacks, authors assumed that the fault is injected during the execution of an RSA signature. Then, from the knowledge of a correct and a corresponding faulty signature, the attacker guesses-and-determines simultaneously the faulty modulus and the part of the private exponent that has been isolated by the fault injection. To recover the whole exponent, the attacker has to repeat the analysis for sufficiently many signatures perturbated at different moments of the execution.

In the next paragraphs, we will detail the different fault attacks published against RSA public elements.

4.1 Modifying N before a signature to solve small DLP.

Although J.-P. Seifert – with its attack proposal to corrupt a RSA signature verification mechanism [37, 33] – first addressed the issue of exploiting RSA signatures performed under a faulty public modulus N, the first analysis leading to a complete secret key recovery is due to E. Brier *et al.* [6]. The main idea of their attack is to an-

alyze faulty signatures performed under a faulty modulus N to recover small parts of the private exponent d (*i.e.* 20 to 30 bits from each faulty signatures). All the parts of exponent extracted from different faulty RSA signatures are finally combined with the Chinese Remainder Theorem to build the full private exponent.

General Methodology.

This fault attack can be split into two distinct phases. The first one is *"on-line"* and consists in gathering K pairs message/faulty signatures $(m_i, S_i)_{1 \leqslant i \leqslant K}$ computed with faulty moduli $\hat{N}_i \neq N$ such that:

$$S_i = \dot{m}_i{}^d \bmod \hat{N}_i \tag{13}$$

The authors assumed that the values of the different faulty $(\hat{N}_i)_{1 \leqslant i \leqslant K}$ are not known by the attacker. However, they have also supposed that theses values are uniformly distributed over the n-bit long integers [14, 17]. From this set of faulty signatures, the attacker performs an *"off-line"* phase which consists in recovering the private exponent by parts. The proposed analysis is declined in different variants depending the choice of the attacker to generate, or not, a table of possible values for faulty moduli. But, generating such a table, also referred as the *dictionary of moduli* [14], requires to choose a fault model. Finally, one can notice that the number of signatures to gather (*i.e.* the parameter K) depends on the method chosen to perform the analysis. Both methods are detailed below.

Analysis without dictionary.

This first method is used when the attacker is not able to identify a fault model from the set of gathered faulty signatures or, if the identified model induce a dictionary too large to be handled (*i.e.* more than 2^{32} entries). In this case, it is not possible to retrieve faulty moduli used to perform the faulty signatures. To overcome this difficulty, the authors give a mean to find some factors p^a of $\hat{N}_i$ thanks to the equation (14) that it satisfied under some conditions [1] with probability 1 if $p^a \mid \hat{N}_i$ and $\frac{1}{r}$ otherwise (where r is a small prime that divides the multiplicative order of $\dot{m}_i$).

$$S_i \equiv \dot{m}_i{}^{d \mod \varphi(p^a)} \mod p^a \tag{14}$$

Then, for such a p^a, if $\varphi(p^a)$ is divisible by a small enough prime r_k (*i.e.* 20 to 30-bit long), the attacker can take advantage of the bias (see [14, Proposition 1] for details) with a counting method that enables to determine parts of the private exponent $d_k = d \bmod r_k$ by solving small discrete logarithms on the faulty signatures gathered. Hence, when $R = \prod_k r_k$ is bigger than N (and obviously bigger than $\varphi(N)$), it is possible to use the Chinese Remainder Theorem to build the whole pri-

[1] p is a prime number such that $p \nmid \dot{m}_i$ and $p \nmid S_i$

vate exponent d from the recovered parts.

One can notice that the advantage of this method is that it may lead to a full key recovery regardingless of the faults injected on the different moduli. According to [14, 17], the implementation of this methodology enables to recover 512-bit private exponents (1024-bit in case of small public exponent e[2]) by gathering about 25000 faulty signatures. On the other hand, 60000 faulty signatures are enough to recover 1024-bit secret exponents (2048-bit in case of small public exponent e).

Analysis with a dictionary.

The attack performance can be improved if the attacker is able to identify and validate a fault model from the faulty signatures collected during the *"on-line"* phase. From this fault model and the knowledge of N, the attacker can establish a list of possible values for the faulty moduli. This list is also called *dictionary of moduli*. Once the dictionary is generated, the attacker tries to guess the pairs $(m_i, \hat{S}_i)$ that result from a computation with one of the dictionary entry. Each successful guess, or *"hit"*, brings a certain amount of information on the private exponent d. In terms of performance, this approach is very interesting since, according to [14, 17], 28 *"hits"* and only 1100 faulty signatures are enough for recovering a 1024-bit RSA private exponent. Moreover, by finding these *"hits"* with a statistical approach, it is possible to extract the private exponent with a number of faults equal to the number of *"hits"* (which is proved to be optimal). In this case 28 faulty signatures may suffice to recover a 1024-bit RSA private exponent.

Although fault attacks has been considered as a powerful way to attack implementations of cryptographic algorithms, the presented attacks highlighted that even non-critical elements, such as public keys, have to be protected against perturbations. Whereas public elements are not supposed to reveal secret information, their perturbation may be a source of leakage leading to the corruption of a signature verification mechanism [37, 33] or worse, to a full private key recovery [14, 17]. However, the use of an exponent randomization technique may be an effective way for defeating such attacks. Furthermore, these attacks only apply for perturbations that occur before performing the core exponentiation of the RSA signature. The attack presented in the next section proved that this claim is no longer true since the perturbation of public elements during the signature can also be exploited.

[2] When e is small, the authors take advantage of the RSA equation $e \cdot d \equiv 1 \mod \varphi N$ to determine the most significant part of d. Indeed, knowing that $\varphi N = N + 1 - p - q \approx N$ for its most significant part, then $d \approx \frac{1+k\cdot(N+1)}{e}$ with $k < e$. Hence, if e is small (*e.g.* $e = 2^16+1$), the most significant part of d can be directly deduced from the previous relation completed by an exhaustive search on k.

4.2 Exploiting faults on N during a RSA signature.

In J.P Seifert and E. Brier *et al.*'s proposals [37, 14], authors exploited perturbations of the public modulus provoked before the core exponentiation so that the whole signature is performed with a faulty modulus $\hat{N}$. The attack presented by A. Berzati *et al.* [8] extended the fault model by enabling an attacker to exploit faults injected of a *"Right-To-Left"* based exponentiation. The modification of N was supposed to be a transient random byte fault modification. It means that only one byte of N is changed to a random value. Moreover, they also assumed that the value of the faulty modulus $\hat{N}$ is not known by the attacker. However, the time location of the fault is a parameter known by the attacker and used to perform the cryptanalysis. This fault model has been chosen for its simplicity and practicability in smart card context [22, 12]. Furthermore, it can be easily adapted to 16-bit or 32-bit architectures. This attack was later generalized to *"Left-To-Right"* based implementations including Montgomery or *Sliding-Window* exponentiations. Finally, the authors proved at CHES 2010 that the exponent randomization method – also referred as exponent blinding – suggested by P. Kocher [28] may not be efficient for protecting RSA implementations against faults on public key elements.

General Methodology.

In order to detail the general methodology, we assume that the implementation of the modular exponentiation is *"Right-To-Left"*-based (see Sect. 2.2.1) and that an attacker is able to transiently modified one byte of the public modulus N at the t-step of the exponentiation. If we denote by A_t and B_t the internal register values, then for a fault occurring while B_t is computed, we have:

$$\hat{S}_t \equiv A_t \cdot \hat{B}_t^{d_t} \cdot \ldots \cdot \hat{B}_t^{2^{(n-1)-t} \cdot d_{n-1}} \bmod \hat{N} \tag{15}$$

$$\equiv A_t \cdot \hat{B}_t^{\frac{d_{[t]}}{2^t}} \bmod \hat{N} \tag{16}$$

where $d_{[t]} = \sum_{i=t}^{n-1} 2^i \cdot d_i$ Hence, the fault injection splits the computation into a correct and a faulty part. The main consequence is that a part of the private exponent, namely $d_{[t]}$ is isolated by the fault injection. In practice this part of exponent is composed of a known (already determined) part and part to guess. So, if the part to guess is small enough, it is possible to *guess-and-determine* it from a faulty/correct signature pair $(\hat{S}_t, S_t)$. Therefore, since the faulty modulus is also unknown by the attacker, the attacker has to choose a candidate value $\hat{N}'$ (built according to the fault model) and another candidate value for the part of $d_{[t]}$ to guess. Then he computes from the correct signature $A'_t \equiv S_t \cdot \dot{m}^{-d'_{[t]}} \bmod N$ and:

$$S'_{(d'_{[t]}, \hat{N}')} \equiv A'_t \cdot \left(\dot{m}^{2^{t-1}} \bmod N \right)^{2 \cdot \frac{d'_{[t]}}{2^t}} \bmod \hat{N}' \tag{17}$$

Then, if this re-built faulty signature satisfies (18) then the pair of candidate values $(\hat{N}', d'_{[t]})$ is the expected one with high probability. Otherwise, the attacker has to choose another candidate pair and perform this test again.

$$S'_{(d'_{[t]};\hat{N}')} \equiv \hat{S}_t \bmod \hat{N}' \tag{18}$$

The whole exponent is gradually recovered by cascading the previous analysis with signatures faulted at different moments of the execution and using the knowledge of already found parts. The performance of this fault attack mainly depends on the number of exponent bits w, that the attacker can extract from correct/signature pair. This parameter is a trade-off between fault number and performance and so, has to be carefully chosen. Authors estimated that for $w = 4$ (which ensure a reasonable execution time), the number of faulty signatures to gather for recovering a 1024-bit private exponent is about 250 [8]. Finally, one can notice that a similar analysis can be performed when the first operation infected by the fault is a multiplication (*i.e.* the computation of A_t is infected first). The details of this variant are provided in [8].

Application to "Left-To-Right" based exponentiations.

After exploiting public key perturbation during execution of *"Right-To-Left"* implementations of RSA signatures, A. Berzati *et al.* later generalized their attack to "Left-To-Right" based exponentiations. Although both dual algorithms are very similar, the generalization of the previous fault attack was not so easy. To illustrate the difficulties induced by this generalization they have considered a classical *"Left-To-Right"* exponentiation (see Sect.2.2.2) and an attacker able to perturbate the modulus N during its execution as in the previous fault model but, t steps before the end of the execution. Denoting by A_{n-t} the internal register before the perturbation of N, the faulty signature $\hat{S}_t$ can be expressed as below:

$$\hat{S}_t = {A_{n-t}}^{2^t} \cdot \dot{m}^{d_{[t]}} \bmod \hat{N} \tag{19}$$

and, this time $d_{[t]} = \sum_{i=0}^{t-1} 2^i \cdot d_i$ denotes the t-bit least significant part of d. By observing (19), one can notice that, contrary to the *"Right-To-Left"* case, t cascaded squares are induced by the fault injection. Hence, extending the previous analysis supposes that the attacker is able to compute modular square roots (which is a difficult problem in RSA rings). Fortunately, since these additional squares are computed modulo $\hat{N}$, it is possible to efficiently compute square roots when $\hat{N}$ is prime (or B-smooth) using the probabilistic Tonelli & Shanks algorithm [18]. To validate the consistency of considering only prime faulty moduli, the authors provided a complete theoretical analysis based on number theory. Hence they estimated that, according to the fault model and for a 1024-bit RSA, 1 faulty modulus over 305 will be prime [7]. Moreover, they observed that, although two square roots may be computed from a given quadratic residue, the number of t-th square root is not 2^t but is bounded by the bigger power of two dividing $\hat{N} - 1$. Since this power is usually quite

small (*i.e.* smaller than 5 in their experiments), they concluded that, in practice, the computation of square roots does not prevent an attacker from using a guess-and-determine approach declined from *"Right-To-Left"* analysis and recovering parts of d when *"Left-To-Right"*-based exponentiations are targeted.

In the previous paragraph, we have detailed a proposal for attacking both dual implementations of the modular exponentiation [8, 7]. In both cases, the attacker takes advantage of the fault that occurs during the execution of an RSA signature. However, the previous results show that *"Right-To-Left"*-based exponentiations seem to be easier to attack than *"Left-To-Right"* ones. Moreover, this attack methodology has been later reused to successfully defeat the randomized exponent countermeasure [9]. Although this countermeasure seems to be efficient against perturbations of public elements that occur before the computation of signatures [14], A. Berzati *et al.* showed at CHES 2010 that signatures partially infected by a faulty public modulus are still exploitable when the private exponent is blinded [9]. However, the authors suggest to use the Probabilistic Signature Scheme with RSA (RSA-PSS) [4] for defeating their attacks. Eventually, this work completes the state-of-the-art and highlights the need for protecting RSA public elements against perturbations, even during the computation of signatures.

5 Conclusion

The study of the fault injection in RSA implementations shows that a large panel of different attacks exist. Of course the popularity of RSA is widely accountable, but the variety of the proposed implementations, even secured ones, leads to different fault exploitations. If first instance of fault attacks has led to very powerful applications, especially for CRT-RSA where one fault may suffice, standard RSA implementations seems to be more difficult to attack. Indeed, for such implementations, the goal of the attacker is not to factor the modulus but gradually recovering the private exponent by bit windows or by residues. The vulnerability of this kind of implementations is the modular exponentiation that scans the private exponent bit by bit. In the both cases, the conditional checks must be avoided or secured and the public elements have to be protected as the others values. One can conclude that implementations that parcel the secret elements for the computation, are by construction vulnerable to fault attacks. The countermeasures as masking techniques lead to confuse the isolated parts, but they may be not enough efficient, as proven by the number of attacks that nevertheless exploit the residual vulnerabilities.

References

1. Bao, F., Deng, R.H., Han, Y., Jeng, A.B., Narasimhalu, A.D., Ngair, T.H.: Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In: B. Christianson, B. Crispo, T.M.A. Lomas, M. Roe (eds.) Security Protocols, *Lecture Notes in Computer Science*, vol. 1361, pp. 115–124. Springer (1998)
2. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. Proceedings the IEEE **94**(2), 370–382 (2006)
3. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: A. De Santis (ed.) Advances in Cryptology – EUROCRYPT '94, *Lecture Notes in Computer Science*, vol. 950, pp. 92–111. Springer (1995)
4. Bellare, M., Rogaway, P.: The exact security of digital signatures. In: Advances in Cryptology – EUROCRYPT '96, *Lecture Notes in Computer Science*, vol. 1070, pp. 399–416. Springer (1996)
5. Bellcore: New threat model breaks crypto codes. Press Release (1996)
6. Berzati, A.: Attaques par canaux cachés sur les implémentations logicielles d'algorithmes cryptographiques. Master's thesis, Grenoble INP & Université Joseph Fourier Grenoble I (2007)
7. Berzati, A., Canovas, C., Dumas, J.G., Goubin, L.: Fault attacks on rsa public keys: Left-to-right implementations are also vulnerable. In: M. Fischlin (ed.) Topics in Cryptology – CT-RSA 2009, *Lecture Notes in Computer Science*, vol. 5473, pp. 414–428. Springer (2009)
8. Berzati, A., Canovas, C., Goubin, L.: Perturbating RSA public keys: An improved attack. In: E. Oswald, P. Rohatgi (eds.) Cryptographic Hardware and Embedded Systems – CHES 2008, *Lecture Notes in Computer Science*, vol. 5154, pp. 380–395. Springer (2008)
9. Berzati, A., Canovas-Dumas, C., Goubin, L.: Public key perturbation of randomized RSA implementations. In: S. Mangard, F.X. Standaert (eds.) Cryptographic Hardware and Embedded Systems – CHES 2010, *Lecture Notes in Computer Science*, vol. 6225, pp. 306–319. Springer (2010)
10. Biehl, I., Meyer, B., Müller, V.: Differential fault analysis of elliptic curve cryptosystems. In: M. Bellare (ed.) Advances in Cryptology – CRYPTO 2000, *Lecture Notes in Computer Science*, vol. 1880, pp. 131–146. Springer (2000)
11. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: B.S. Kaliski Jr. (ed.) Advances in Cryptology – CRYPTO '97, *Lecture Notes in Computer Science*, vol. 1294, pp. 513–525. Springer (1997)
12. Blömer, J., Otto, M.: Wagner's attack on a secure CRT-RSA algorithm reconsidered. In: L. Breveglieri, I. Koren, D. Naccache, J.P. Seifert (eds.) Fault Diagnosis and Tolerance in Cryptography – FDTC 2006, *Lecture Notes in Computer Science*, vol. 4236, pp. 13–23. Springer (2006)
13. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. Journal of Cryptology **14**(2), 101–119 (2001). Earlier version published in EUROCRYPT '97
14. Brier, É., Chevallier-Mames, B., Ciet, M., Clavier, C.: Why one should also secure RSA public key elements. In: L. Goubin, M. Matsui (eds.) Cryptographic Hardware and Embedded Systems – CHES 2006, *Lecture Notes in Computer Science*, vol. 4249, pp. 324–338. Springer (2006)
15. Brumley, D., Boneh, D.: Remote timing attacks are practical. Computer Networks **48**(5), 701–716 (2005)
16. Ciet, M., Joye, M.: Elliptic curve cryptosystems in the presence of permanent and transient faults. Designs, Codes and Cryptography **36**(1), 33–43 (2005)
17. Clavier, C.: De la sécurité des cryptosystèmes embarqués. Ph.D. thesis, Université de Versailles Saint-Quentin-en-Yvelines (2007)
18. Cohen, H.: A Course in Computational Algebraic Number Theory, *Graduate Texts in Mathematics*, vol. 138. Springer (1993)

19. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Ç.K. Koç, C. Paar (eds.) Cryptographic Hardware and Embedded Systems − CHES '99, *Lecture Notes in Computer Science*, vol. 1717, pp. 292–302. Springer (1999)
20. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **IT-22**(6), 644–654 (1976)
21. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Ç.K. Koç, D. Naccache, C. Paar (eds.) Cryptographic Hardware and Embedded Systems − CHES 2001, *Lecture Notes in Computer Science*, vol. 2162, pp. 251–261. Springer (2001)
22. Giraud, C.: DFA on AES. In: H. Dobbertin, V. Rijmen, A. Sowa (eds.) Advanced Encryption Standard − AES (AES 2004), *Lecture Notes in Computer Science*, vol. 3373, pp. 27–41. Springer (2005)
23. Hoch, J.J., Shamir, A.: Fault analysis of stream ciphers. In: M. Joye, J.J. Quisquater (eds.) Cryptographic Hardware and Embedded Systems − CHES 2004, *Lecture Notes in Computer Science*, vol. 3156, pp. 240–253. Springer (2004)
24. Joye, M., Quisquater, J.J., Bao, F., Deng, R.H.: RSA-type signatures in the presence of transient faults. In: M. Darnell (ed.) Cryptography and Coding, *Lecture Notes in Computer Science*, vol. 1355, pp. 155–160. Springer (1997)
25. Kim, C.H., Bulens, P., Petit, C., Quisquater, J.J.: Fault attacks on public key elements: Application to DLP-based schemes. In: S.F. Mjølsnes, S. Mauw, S.K. Katsikas (eds.) Public Key Infrastructure (EuroPKI 2008), *Lecture Notes in Computer Science*, vol. 5057, pp. 182–195. Springer (2008)
26. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., te Riele, H.J.J., Timofeev, A., Zimmermann, P.: Factorization of a 768-bit RSA modulus. In: T. Rabin (ed.) Advances in Cryptology − CRYPTO 2010, *Lecture Notes in Computer Science*, vol. 6223, pp. 333–350. Springer (2010)
27. Koç, Ç.K.: High-speed RSA implementation. Tech. Rep. TR 201, RSA Laboratories (1994)
28. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: N. Koblitz (ed.) Advances in Cryptology − CRYPTO '96, *Lecture Notes in Computer Science*, vol. 1109, pp. 104–113. Springer (1996)
29. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: M.J. Wiener (ed.) Advances in Cryptology − CRYPTO '99, *Lecture Notes in Computer Science*, vol. 1666, pp. 388–397. Springer (1999)
30. Lenstra, A.K., Lenstra Jr, H.W. (eds.): The Development of the Number Field Sieve, *Lecture Notes in Mathematics*, vol. 1554. Springer (1993)
31. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1997)
32. Montgomery, P.L.: Speeding up the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177), 243–264 (1987)
33. Muir, J.A.: Seifert's RSA fault attack: Simplified analysis and generalizations. In: P. Ning, S. Qing, N. Li (eds.) Information and Communications Security (ICICS 2006), *Lecture Notes in Computer Science*, vol. 4307, pp. 420–434. Springer (2006)
34. Otto, M.: Fault attacks and countermeasures. Ph.D. thesis, Institut für Informatik, Universität Paderborn (2004)
35. Quisquater, J.J., Couvreur, C.: Fast decipherment algorithm for RSA public-key cryptosystem. Electronics Letters **18**(21), 905–907 (1982)
36. Rivest, R.L., Shamir, A., Adleman, L.M.: Method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
37. Seifert, J.P.: On authenticated computing and RSA-based authentication. In: V. Atluri, C. Meadows, A. Juels (eds.) 12th ACM Conference on Computer and Communications Security (CCS 2005), pp. 122–127. ACM Press (2005)
38. Yen, S.M., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on Computers **49**(9), 967–970 (2000)
39. Yen, S.M., Kim, S., Lim, S., Moon, S.J.: A countermeasure against one physical cryptanalysis may benefit another attack. In: K. Kim (ed.) Information Security and Cryptology − ICISC 2001, *Lecture Notes in Computer Science*, vol. 2288, pp. 269–294. Springer (2002)