Differential Fault Attacks on AES

F. De Santis
desantis@tum.de

Sichere Implementierung kryptographischer Verfahren WS 2015-2016
Lehrstuhl für Sicherheit in der Informationstechnik
Technische Universität München

13.01.2016

Lehrstuhl für Sicherheit
in der Informationstechnik

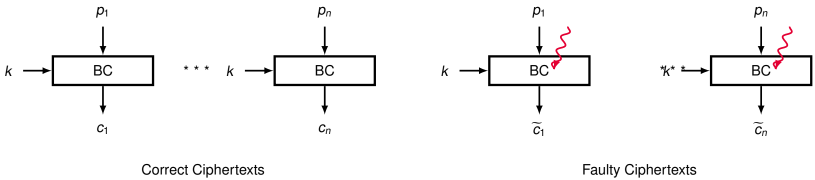
Technische Universität München

Section 1

Background

Differential Fault Attacks (DFA)

- Idea: recover the secret key from the analysis of **correct** and **faulty** computations



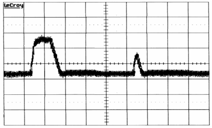
- Let $FP_i = (c_i, \widetilde{c_i})$ be a **faulty pair**, where c_i is the correct ciphertext and $\widetilde{c_i}$ the corresponding faulty ciphertext.

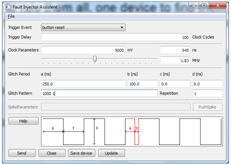
Fault Injection

- Digital circuits operate within specific operation conditions
  - ➥ Supply voltage
  - ➥ Clock frequency
  - ➥ Temperature
  - ➥ ...
- Goal: induce faults during the computation by changing the operating conditions
- Consequences:
  - ➥ instructions are changed/skipped
  - ➥ data are changed
- Challenge: induce faults **reliably** (in a controllable and reproducible way)
- Risks: unwanted behaviours (resets or permanent damage)

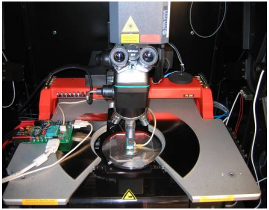
Fault Injection Techniques

- Spike/Glitch injections
  - ➥ Insert spikes in the power supply lines (change V_{dd})
  - ➥ Insert glitches in the clock line / alter clock parameters (period, voltage, ...)





- Optical injections
  - ➥ Use light sources to switch transistors (focused lasers)



Fault Model: Parameters

1. Spatial location (where to inject to fault)
   - ➥ Memory elements
   - ➥ Logic cells (combinational)

2. Temporal control (when and how long inject the fault)
   - ➥ Duration (permanent, transient)

3. Fault type
   - ➥ Stuck at 0/1
   - ➥ Bit flip
   - ➥ Random

4. Extension
   - ➥ Single bit
   - ➥ Multi bit
   - ➥ Word-wise

Section 2

DFA on AES

The AES Encryption Algorithm

Algorithm 1 Pseudocode for the AES $\mathtt{Enc}$ Algorithm

Input: p, k
Output: $c = \mathtt{Enc}(k, p)$

```
rk[0...10] ← KeySched(k)                    # generate round keys
s ← ARK(p, rk[0])                           # key whitening
for r = 1 to 9 do
    s ← ARK(MC(SR(SB(s))), rk[i])           # iterate the round function
end for
c ← ARK(SR(SB(s)), rk[10])                  # last round without MC
return c
```

MixColumns (MC) Operation

Each **column** of the state is viewed as a polynomial with coefficients in $\mathbb{GF}(2^8)$ and multiplied by a fixed polynomial module $x^4 + 1$ leading to the following transformation:

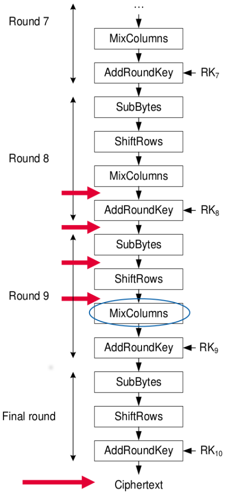
$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}, \quad c \in [0, 3]$$

Fault Injection on AES

Fault model 1

- Location: AES state
- Time: transient between MC in $r = 8$ and MC in $r = 9$
- Type: random
- Extension: single byte

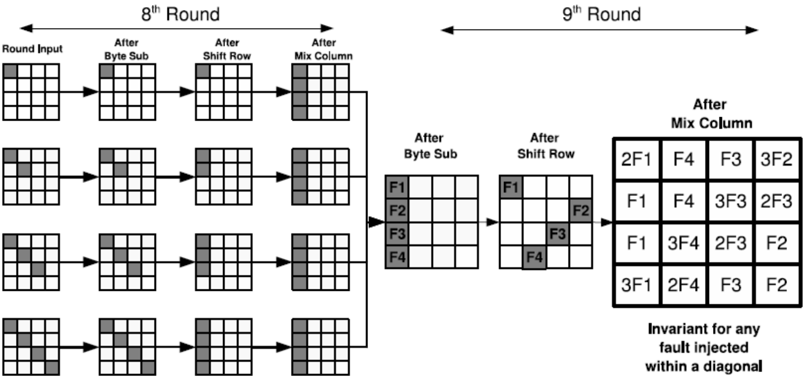
- If a **single byte** is faulted in the state, then 4 **bytes** result faulted in the ciphertext
- 4 **fault injections** are required to recover all the 16 round key bytes



DFA on AES

- Let F_i define the $\oplus$-difference between correct and faulty data
- Fault Model 1

Figure from http://eprint.iacr.org/2009/581.pdf



DFA on AES

Let denote the elements of the invariant after the MC as follows:

$$\begin{cases} a_0 = & 2F1 \\ a_1 = & F1 \\ a_2 = & F1 \\ a_3 = & 3F1 \end{cases}$$

then for the correct key hypothesis it must hold:

$$\begin{cases} a_0 = & 2a_1 \\ a_1 = & a_2 \\ a_3 = & 3a_1 \end{cases}$$

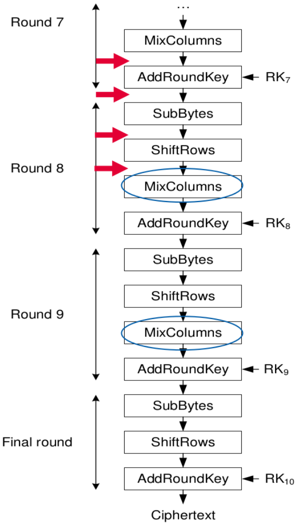
- The DFA computes the MixColumns output at round 9 backwards from the faulty pair and uses the system of equations above to verify the correct key hypothesis
- Multiple faulty pairs can be used during the attack

Fault Injection on AES

Fault model 2

- Location: AES state
- Time: transient between MC in $r = 7$ and MC in $r = 8$
- Type: random
- Extension: single byte

- If a **single byte** is faulted in the state, then **all the 16 bytes** result faulted in the ciphertext
- A **single fault injection** is required to recover all the 16 round key bytes



Section 3

Assignment

Task Description

1. Implement the DFA attack script in Python
   - ➥ Project skeleton @ https://tueisec-sica.sec.ei.tum.de/
     - ✯ `project.nja` project file for Ninja IDE
     - ✯ `main.py` **not** to be modified
     - ✯ `student.py` for implementing the attack
     - ✯ **Two** faulty pairs in CSV format: plaintext, correct ciphertext, faulty ciphertext
     - ✯ `aes_faulty_pairs.py` to generate your own faulty pairs for testing

2. Recover the **last round key** from given faulty pairs
   - ➥ Different for each student @ https://tueisec-sica.sec.ei.tum.de/faulty/

Faulty Pairs Generation

$ aes_faulty_pairs.py -r ROUND -b BYTE -m TYPE -n PAIRS -k KEY -v VERBOSE

- Round number
- Index state byte to fault
- Type (RND for random)
- How many pairs
- Secret key
- Verbose execution

Stairway to Heaven

1. Download the project skeleton
2. Implement the DFA attack
3. Recover the secret key from the given faulty pair
4. Submit the source code and the `key.txt` **before 03.02.2016 23:59:59 CET**

Concluding Remarks

- The assignment is passed if the `key.txt` is correct **and** the `student.py` works correctly on freshly generated faulty pairs
- The usual rules apply (multiple submissions, don't include libraries etc ...)
- Tip: implement independent searches on max 2^{16} key hypothesis
- Tip: the 2^{nd} faulty pair can be used to purge the list of key candidates obtained from the 1^{st} faulty pair (intersection).
- In case you end up with more than one key candidate using two faulty pairs:
  1. Check your script twice
  2. Contact the tutors
- Tip: Plaintexts can be used to verify your `key.txt`