Persistent Fault Analysis on Block Ciphers

Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi and Kui Ren

Zhejiang University

CHES2018, Amsterdam, Netherlands, 09/12/2018
1 Introduction
2 Persistent Fault Attack
3 Persistent Fault Analysis on AES-128
4 PFA on Countermeasures against Fault Analysis
5 Case Study – Rowhammer-based PFA on T-box
6 Conclusion and Future Work
1.1 What are fault attacks

- Active attacks against cryptographic implementations
- FA (Fault Attack) first proposed by Boneh et al in 1996
- Two stages: Fault injection and Fault analysis

adopted from Josep Balasch in IACR Summer School 2015
1. Introduction

1.2 Fault injection (online)

- Categories
  - Non-invasive
  - Semi-invasive
  - Invasive

- Techniques
  - Clock Glitch
  - Voltage Spike
  - EM Pulse
  - Optical Laser

- Very popular form of non-invasive attacks

adopted from Josep Balasch in IACR Summer School 2015
1. Introduction

1.3 Fault model

- Granularity: how many bits are affected (aka fault width)
- Modification (aka fault type)
  - Stuck-at, e.g. zero or one
  - Flip
  - Random
- Control: on the fault location and on timing
  - Precise
  - Loose
  - None
- Duration or effect of the fault
  - Transient
  - Permanent
  - Destructive

Persistent

adopted from Josep Balasch in IACR Summer School 2015
1.4 Countermeasures

- **Hardening hardware**
  - Hide sensitive parts of the chip
  - Add filters and/or security sensors

- **Hardening computations**
  - Information redundancy (Addition of parities, linear codes, Ring embeddings, Infective computations)
  - Hiding countermeasures
  - Branchless implementations
  - Parallel execution or inverse execution

---

adopted from Josep Balasch in IACR Summer School 2015
1.5 Disadvantages of previous works

- Very tight time synchronization on the round calculation and the injection timing
- Very complicated analysis due to the random value and the fault propagation
- May not work if there are countermeasures against fault attacks
1 Introduction

2 Persistent Fault Attack

3 Persistent Fault Analysis on AES-128

4 PFA on Countermeasures against Fault Analysis

5 Case Study – Rowhammer-based PFA on T-box

6 Conclusion and Future Work
2.1 Fault model of PFA

- The adversary can inject faults before the encryption of a block cipher
  - Typically, these faults alter a stored algorithm constant

- The injected faults are persistent
  - The affected constant stays faulty unless refreshed
  - All iterations are computed with the faulty constant

- The adversary is capable of collecting multiple ciphertext outputs
  - A watchdog counter on detected faults is considered out of scope
2. Persistent Fault Attack

2.2 Core idea of Persistent Fault Attack

Three Stages

1. Persistent fault injection
2. Encryption with persistent faults
3. Persistent fault analysis

\[ C' = C'' = C \]
Correct Ciphertexts

\[ C' \neq C \]
Faulty Ciphertexts
2.3 Overview of Persistent Fault Analysis (PFA)

- A statistical analysis on the last round, exploiting three types of fault leakages
- $v$ and $v^*$ are known

![Diagram showing normal encryption and faulty encryption with probability distributions for $Pr(y_j)$ and $Pr(c_j)$](image)
2. Persistent Fault Attack

2.3 Illustration of analysis result

- Counts the number of appearances of possible values for the specific byte in ciphertexts

(a) Extract $k_1$ using the distribution of $c_1$

(b) Extract $k_2$ using the distribution of $c_2$
## 2. Persistent Fault Attack

### 2.4 PFA on the last round of AES

**Algorithm 1**: Pseudo code of PFA on the last round of a general block cipher.

```plaintext
for u = 0; u < L; u++ do
  for t = 0; t < 2^b; t++ do
    Counts[u][t] = 0;
  end
end

for u = 0; u < L; u++ do
  for n = 0; n < N; n++ do
    Counts[u][c_{u,n}]++;
  end
end

for u = 0; u < L; u++ do
  for t = 0; t < 2^b; t++ do
    if Counts[u][t] > 0 then
      Discard candidate k_u = t ⨁ v;
    end
  end
end
```

// \( c_{u,n} \) is \( c_u \) in the \( n \)-th ciphertext
2. Persistent Fault Attack

2.5 Comparison with other fault analysis

(1) The attack is not differential in nature and thus the control over the plaintext is not required.
(2) The adversary does not necessarily need live synchronization.
(3) The fault model remains relaxed.
(4) PFA can also be applied in multiple fault setting.
(5) PFA can bypass some redundancy based countermeasures.
(6) An adversary can always inject the persistent fault before the victim is switched to the sensitive mode.

| (1) It needs higher number of ciphertexts as compared to DFA |
| (2) Persistent faults can be detected by some built-in health test mechanism. |
3. PFA on AES-128

3.1 AES implementations

- S-box Implementation
- T-box Implementation

**Table 1:** Different implementations of AES-128 encryptions.

<table>
<thead>
<tr>
<th>Type</th>
<th>Lookups in each round</th>
<th>Table size</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>$R_{1-10}$: $S$</td>
<td>$S$:256B</td>
<td>Typical S-box implementation</td>
</tr>
<tr>
<td>I2</td>
<td>$R_{1-10}$: $T_0, T_1, T_2, T_3$</td>
<td>$T_i$:1KB</td>
<td>Typical T-box implementation</td>
</tr>
<tr>
<td>I3</td>
<td>$R_{1-9}$: $T_0, T_1, T_2, T_3$</td>
<td>$T_i$:1KB</td>
<td>Code can be found in rijndael-amd64.S in the library Libgcrypt 1.6.3</td>
</tr>
<tr>
<td></td>
<td>$R_{10}$: $T_0', T_1', T_2', T_3'$</td>
<td>$T_i':1KB$</td>
<td></td>
</tr>
</tbody>
</table>


3.2 PFA on vulnerable S-box implementation

(a) Extract $k_1$ using the distribution of $c_1$

(b) Extract $k_2$ using the distribution of $c_2$
3.3 Practical result v.s. Theoretical estimation

- $\phi_t(n)$ is calculated by the equation, coupon collector’s problem.
- $\phi(n)$ is calculated by the code
- $\phi(n)$ is close to $\phi_t(n)$
  - $\phi_t(n) \leq 16$ when $n \approx 1240$
  - $\phi(n) \leq 16$ when $n \approx 1360$
  - $\phi_t(n) \leq 1$ when $n \geq 1405$
  - $\phi(n) \leq 1$ when $n \geq 2148$
3.4 Sample size distributions for full key recovery

- The full key attacks are conducted $\xi=1000$ times.

- $N_f$ denotes the number of ciphertexts that is required when the adversary can extract all 16 key bytes for the first time.

- $1678 \leq N_f \leq 3504$
- $N_f \approx 2281$ on average
OUTLINE

1. Introduction
2. Persistent Fault Attack
3. Persistent Fault Analysis on AES-128
4. PFA on Countermeasures against Fault Analysis
5. Case Study – Rowhammer-based PFA on T-box
6. Conclusion and Future Work
4. PFA on Countermeasures against FA

4.1 Dual Modular Redundancy (DMR)

- Time redundancy v.s. Space redundancy
- Two modules: Module 1 and Modules 2
  - Redundant Encryption based DMR (REDMR)
  - Inversive Decryption based DMR (IDDMR)
- PFA is naturally against REDMR
4.2 Three types based on the reaction

- NCO: No ciphertext output
- ZVO: Zero value output
- RCO: Random ciphertext output

- REDMR
  - If both the modules use shared memory, *i.e.*, common lookup tables
  - All three countermeasures will fail

- IDDMR
  - A stronger countermeasure (two different lookup tables)
4.3 PFA on S-box (I1) with NCO/ZVO

- $p$, the probability that one plaintext can bypass IDDMR

\[ p = \left(1 - \frac{1}{256}\right)^{160} \approx 0.5346 \]

- Only $p \times N$ ciphertexts can be used in attacks
- The adversary requires $N/p \approx 1.8706 \times N$ encryptions (equivalent to REDMR)
- $\xi = 1000$
- $3042 \leq N_f \leq 7141$
- $N_f \approx 4234$ on average
- If $n \geq 7200$, the success rate is 100%
4. PFA on Countermeasures against FA

4.4 PFA on S-box (I1) with RCO

- No impossible values, however, the slight probability difference can still be detected.

\[
Pr(y = v) = 0 \times p + \frac{1}{256} \times (1 - p) = \frac{0.4654}{256}
\]

\[
Pr(y = v^*) = \frac{2}{256} \times p + \frac{1}{256} \times (1 - p) = \frac{1.5346}{256}
\]

\[
Pr(y \neq v \land y \neq v^*) = \frac{1}{256} \times p + \frac{1}{256} \times (1 - p) = \frac{1}{256}
\]

(a) Extract \( k_1 \) using the distribution of \( c_1 \)

(b) Extract \( k_2 \) using the distribution of \( c_2 \)
4.5 PFA on AES-128 with RCO using thresholds

- Two thresholds to differentiate the abnormal cases
- Apply PFA on S-box (I1) and T-box (I2) implementation

\[
\begin{align*}
\tau_1 &= 0.9 \times \frac{1.5346}{256} \\
\tau_2 &= 1.1 \times \frac{0.4654}{256}
\end{align*}
\]
OUTLINE

1 Introduction
2 Persistent Fault Attack
3 Persistent Fault Analysis on AES-128
4 PFA on Countermeasures against Fault Analysis
5 Case Study – Rowhammer-based PFA on T-box
6 Conclusion and Future Work
5. Case Study: Rowhammer-based PFA

5.1 Background of Rowhammer techniques and shared libraries

- **Rowhammer vulnerability**
  - Appeared in 2014
  - Frequent DRAM access leads to disturbance errors
  - **Hardware intrinsic**, difficult to prevent
  - Can be triggered from software (js, network)
  - Can gain the privileges of **ring0** without authorizations

- **Shared library**
  - Multiple processes shared one lib
  - Dynamic load
  - Read only at **ring3** (user mode)
  - **Libgcrypt**, OpenSSL, Crypto++, etc
5. Case Study: Rowhammer-based PFA

5.2 Overview of Rowhammer-based PFA on shared libraries

- Four steps
  - Profiling (optional)
  - Allocation
  - Positioning
  - Hammering

- Implication
  - One bit in the table will be flipped
  - Hammering one random bit in any $T'_0$ is possible
  - The bit flip is persistent for both encryptions and all rounds and viewed as faulty for careless users
## 5.3 Setup of our Rowhammer experiments

- **Lenovo ThinkPad x230 laptop**
  - Intel(R) Core(TM) i5-3320M at 2.60GHz
  - two Samsung DDR3 modules, 2GB at 1333MHz
  - Linux OS is Ubuntu 12.04 LTS, kernel version of 3.2.0-79 generic

- **Libgcrypt v1.6.3**
  - Compiled as shared library
  - GCC 4.6.3, No optimization

- **T-box implementation (I3)**
  - AES T-table T0 starts at the offset 000d6710h
  - T\textsubscript{0} is followed by the corresponding element of T’\textsubscript{0}

<table>
<thead>
<tr>
<th>Address</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>Value 6</th>
<th>Value 7</th>
<th>Value 8</th>
<th>Value 9</th>
<th>Value 10</th>
<th>Value 11</th>
<th>Value 12</th>
<th>Value 13</th>
<th>Value 14</th>
<th>Value 15</th>
<th>Value 16</th>
<th>Value 17</th>
<th>Value 18</th>
<th>Value 19</th>
<th>Value 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>000d6710h</td>
<td>C6</td>
<td>63</td>
<td>63</td>
<td>A5</td>
<td>63</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>F8</td>
<td>7C</td>
<td>7C</td>
<td>84</td>
<td>7C</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000d6720h</td>
<td>EE</td>
<td>77</td>
<td>77</td>
<td>99</td>
<td>77</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>F6</td>
<td>7B</td>
<td>7B</td>
<td>8D</td>
<td>7B</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000d6730h</td>
<td>FF</td>
<td>F2</td>
<td>F2</td>
<td>0D</td>
<td>F2</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>D6</td>
<td>6B</td>
<td>6B</td>
<td>BD</td>
<td>6B</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000d6740h</td>
<td>DE</td>
<td>6F</td>
<td>6F</td>
<td>B1</td>
<td>6F</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>91</td>
<td>C5</td>
<td>C5</td>
<td>54</td>
<td>C5</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000d6750h</td>
<td>60</td>
<td>30</td>
<td>30</td>
<td>50</td>
<td>30</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>02</td>
<td>01</td>
<td>01</td>
<td>03</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000d6760h</td>
<td>CE</td>
<td>67</td>
<td>67</td>
<td>A9</td>
<td>67</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>56</td>
<td>2B</td>
<td>2B</td>
<td>7D</td>
<td>2B</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000d6770h</td>
<td>E7</td>
<td>FE</td>
<td>FE</td>
<td>19</td>
<td>FE</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>B5</td>
<td>D7</td>
<td>D7</td>
<td>62</td>
<td>D7</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000d6780h</td>
<td>4D</td>
<td>AB</td>
<td>AB</td>
<td>E6</td>
<td>AB</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>EC</td>
<td>76</td>
<td>76</td>
<td>9A</td>
<td>76</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.4 Results of Hammering

- Successfully inject one bit to any of $T'_0$, $T'_1$, $T'_2$, $T'_3$
  - Occur 5, 4, 6, 5 times to $T'0$, $T'1$, $T'2$, $T'3$, in 90.80, 57.75, 49.83, 59.6 minutes respectively

- Ranging from 3 up to 230 minutes for the first 20 experiments
  - Facilitated with profiling

- It takes about 461 and 1367 minutes
  - Without profiling

<table>
<thead>
<tr>
<th>ID</th>
<th>Attack time(min)</th>
<th>Location of flip</th>
<th>Data before injection</th>
<th>Data after injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>$T'_0[235]$</td>
<td>e900 0000</td>
<td>e900 0000</td>
</tr>
<tr>
<td>2</td>
<td>38</td>
<td>$T'_1[208]$</td>
<td>0070 0000</td>
<td>0050 0000</td>
</tr>
<tr>
<td>3</td>
<td>53</td>
<td>$T'_2[100]$</td>
<td>0000 4300</td>
<td>0000 4100</td>
</tr>
<tr>
<td>4</td>
<td>81</td>
<td>$T'_3[67]$</td>
<td>0000 001a</td>
<td>0000 0018</td>
</tr>
<tr>
<td>5</td>
<td>230</td>
<td>$T'_0[18]$</td>
<td>e900 0000</td>
<td>e800 0000</td>
</tr>
<tr>
<td>6</td>
<td>102</td>
<td>$T'_1[131]$</td>
<td>00ec 0000</td>
<td>00cc 0000</td>
</tr>
<tr>
<td>7</td>
<td>77</td>
<td>$T'_2[172]$</td>
<td>0000 9100</td>
<td>0000 9000</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>$T'_3[34]$</td>
<td>0000 0093</td>
<td>0000 0091</td>
</tr>
<tr>
<td>9</td>
<td>104</td>
<td>$T'_0[230]$</td>
<td>08c0 0000</td>
<td>0860 0000</td>
</tr>
<tr>
<td>10</td>
<td>49</td>
<td>$T'_1[126]$</td>
<td>0000 8300</td>
<td>0000 7300</td>
</tr>
<tr>
<td>11</td>
<td>86</td>
<td>$T'_2[101]$</td>
<td>0000 004d</td>
<td>0000 004c</td>
</tr>
<tr>
<td>12</td>
<td>75</td>
<td>$T'_3[55]$</td>
<td>0000 009a</td>
<td>0000 001a</td>
</tr>
<tr>
<td>13</td>
<td>17</td>
<td>$T'_0[221]$</td>
<td>0000 c100</td>
<td>0000 8100</td>
</tr>
<tr>
<td>14</td>
<td>44</td>
<td>$T'_1[67]$</td>
<td>001a 0000</td>
<td>0018 0000</td>
</tr>
<tr>
<td>15</td>
<td>53</td>
<td>$T'_2[147]$</td>
<td>0000 00dc</td>
<td>0000 00d8</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>$T'_3[108]$</td>
<td>0000 0050</td>
<td>0000 0010</td>
</tr>
<tr>
<td>17</td>
<td>41</td>
<td>$T'_0[252]$</td>
<td>0000 0f00</td>
<td>0000 0b00</td>
</tr>
<tr>
<td>18</td>
<td>62</td>
<td>$T'_1[140]$</td>
<td>0000 6400</td>
<td>0000 4400</td>
</tr>
<tr>
<td>19</td>
<td>47</td>
<td>$T'_2[13]$</td>
<td>0017 0000</td>
<td>0097 0000</td>
</tr>
<tr>
<td>20</td>
<td>85</td>
<td>$T'_3[168]$</td>
<td>e200 0000</td>
<td>e200 0000</td>
</tr>
</tbody>
</table>

1 461(w/o profiling) $T'_3[75]$ 0000 00b3 0000 0013
2 1367(w/o profiling) $T'_0[163]$ 000a 0000 0002 0000
5.5 Results of Analysis

- REDMR
- One injection can recover four bytes.
  - 4000 ciphertexts are collected
- At least four injections are required
- 8200 ciphertexts are required to recover the full key
  - 2050 ciphertexts per row
OUTLINE

1 Introduction

2 Persistent Fault Attack

3 Persistent Fault Analysis on AES-128

4 PFA on Countermeasures against Fault Analysis

5 Case Study – Rowhammer-based PFA on T-box

6 Conclusion and Future Work
6.1 Conclusion

- We propose persistent fault analysis
  - A novel attack on general block ciphers
  - Can defeat mainstream countermeasures against fault attacks
  - Can be used in different fault attacks with persistence
  - Different implementations
  - Different analysis strategies

- We conduct several evaluations
  - The attack is practically conducted in a shared library setting to target AES-128 in cryptographic library Libgcrypt
6.2 Future work

- More formal proofs on the theoretical estimation based on probabilities
  - Analog to Coupon Collector’s Problem
- Revisit the case for key scheduling
- Search for more scenarios where the persistence holds
- Countermeasures design (Counter or health check)
- And more
Thank you very much!