Practical Multiple Persistent Fault Analysis

Hadi Soleimany  Nasour Bagheri  **Hosein Hadipour**  Prasanna Ravi  Shivam Bhasin

Sara Mansour

CHES 2022 - Leuven, Belgium

> hsn.hadipour@gmail.com
Outline

1. Introduction and the Research Gap

2. Our Framework for PFA With Multiple Faults

3. A Generic Key Recovery Framework

4. Conclusion
Introduction and the Research Gap
Fault Attacks

⚠️ **Fault attack**: An active side-channel attack [BDL97]:

🔧 **Fault injection**: Disturb the operation of a cryptographic device

🗂️ **Fault analysis**: Analyze the erroneous outputs to retrieve the secret key
Fault Attacks

⚠️ **Fault attack**: An active side-channel attack [BDL97]:

🛠 **Fault injection**: Disturb the operation of a cryptographic device

🔍 **Fault analysis**: Analyze the erroneous outputs to retrieve the secret key
Persisent Fault Attack (PFA)

PFA fault model [Zha+18]:

- The injected faults are persistent until the reset of the device
- The injected faults typically alter the stored algorithm constants
- We can inject the faults before the encryption
- We can collect multiple faulty ciphertexts

\[
\begin{array}{cccccccccccccc}
 x & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & a & b & c & d & e & f \\
 S(x) & 6 & 4 & c & 5 & 0 & 7 & 2 & e & 1 & f & 3 & d & 8 & a & 9 & b \\
 S'(x) & 6 & 4 & c & 5 & 0 & 7 & 2 & 4 & 1 & f & 3 & d & 8 & a & 9 & b \\
\end{array}
\]
Persisent Fault Attack (PFA)

PFA fault model [Zha+18]:

- The injected faults are persistent until the reset of the device
- The injected faults typically alter the stored algorithm constants
- We can inject the faults before the encryption
- We can collect multiple faulty ciphertexts

\[
\begin{array}{cccccccccccc}
 x & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & a & b & c & d & e & f \\
 S(x) & 6 & 4 & c & 5 & 0 & 7 & 2 & e & 1 & f & 3 & d & 8 & a & 9 & b \\
 S'(x) & 6 & 4 & c & 5 & 0 & 7 & 2 & 4 & 1 & f & 3 & d & 8 & a & 9 & b \\
\end{array}
\]
Persisent Fault Attack (PFA)

PFA fault model [Zha+18]:

- The injected faults are persistent until the reset of the device
- The injected faults typically alter the stored algorithm constants
- We can inject the faults before the encryption
- We can collect multiple faulty ciphertexts

\[ S(x) = 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad a \quad b \quad c \quad d \quad e \quad f \]

\[ S'(x) = 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad a \quad b \quad c \quad d \quad e \quad f \]
Persisent Fault Attack (PFA)

PFA fault model [Zha+18]:

- The injected faults are persistent until the reset of the device
- The injected faults typically alter the stored algorithm constants
- We can inject the faults before the encryption
- We can collect multiple faulty ciphertexts

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S(x)$</td>
<td>6</td>
<td>4</td>
<td>c</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>e</td>
<td>1</td>
<td>f</td>
<td>3</td>
<td>d</td>
<td>8</td>
<td>a</td>
<td>9</td>
<td>b</td>
</tr>
<tr>
<td>$S'(x)$</td>
<td>6</td>
<td>4</td>
<td>c</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>f</td>
<td>3</td>
<td>d</td>
<td>8</td>
<td>a</td>
<td>9</td>
<td>b</td>
</tr>
</tbody>
</table>
Core Idea of PFA

| x  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S(x)$ | 6 | 4 | c | 5 | 0 | 7 | 2 | e | 1 | f | 3 | d | 8 | a | 9 | b |

$c = S(x) + k$

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S'(x)$</td>
<td>6</td>
<td>4</td>
<td>c</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>f</td>
<td>3</td>
<td>d</td>
<td>8</td>
<td>a</td>
<td>9</td>
<td>b</td>
</tr>
</tbody>
</table>

$c = S'(x) \oplus k$

- Filter wrong keys: $S'(x) \neq 0xe \Rightarrow k \neq 0xe \oplus c$
- Filter wrong keys: $S'(X[i]) \neq 0xe \Rightarrow K[i] \neq 0xe \oplus C[i]$

Hadi Soleimany, Nasour Bagheri, **Hosein Hadipour**, Prasanna Ravi, Shivam Bhasin, Sara Mansouri
CHES 2022 - Leuven, Belgium
Core Idea of PFA

\[
x \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad a \quad b \quad c \quad d \quad e \quad f \\
S(x) \quad 6 \quad 4 \quad c \quad 5 \quad 0 \quad 7 \quad 2 \quad e \quad 1 \quad f \quad 3 \quad d \quad 8 \quad a \quad 9 \quad b
\]

\[c = S(x) + k\]

\[
x \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad a \quad b \quad c \quad d \quad e \quad f \\
S'(x) \quad 6 \quad 4 \quad c \quad 5 \quad 0 \quad 7 \quad 2 \quad 4 \quad 1 \quad f \quad 3 \quad d \quad 8 \quad a \quad 9 \quad b
\]

\[c = S'(x) \oplus k\]

- Filter wrong keys: \(S'(x) \neq 0xe \Rightarrow k \neq 0xe \oplus c\)
- Filter wrong keys: \(S'(X[i]) \neq 0xe \Rightarrow K[i] \neq 0xe \oplus C[i]\)
Core Idea of PFA

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S(x))</td>
<td>6</td>
<td>4</td>
<td>c</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>e</td>
<td>1</td>
<td>f</td>
<td>3</td>
<td>d</td>
<td>8</td>
<td>a</td>
<td>9</td>
<td>b</td>
</tr>
</tbody>
</table>

\[ c = S(x) + k \]

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S'(x))</td>
<td>6</td>
<td>4</td>
<td>c</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>f</td>
<td>3</td>
<td>d</td>
<td>8</td>
<td>a</td>
<td>9</td>
<td>b</td>
</tr>
</tbody>
</table>

\[ c = S'(x) \oplus k \]

- Filter wrong keys: \(S'(x) \neq 0xe \Rightarrow k \neq 0xe \oplus c\)
- Filter wrong keys: \(S'(X[i]) \neq 0xe \Rightarrow K[i] \neq 0xe \oplus C[i]\)
Core Idea of PFA

\[ c = S(x) + k \]

\[ c = S'(x) \oplus k \]

- **Filter wrong keys**: \( S'(x) \neq 0xe \Rightarrow k \neq 0xe \oplus c \)
- **Filter wrong keys**: \( S'(X[i]) \neq 0xe \Rightarrow K[i] \neq 0xe \oplus C[i] \)
Limits of the Original PFA

PFA requires about 2000 faulty ciphertexts per key \cite{Zha+18}.

\[ N \gtrsim 2000 \Rightarrow |K| = \lambda^{16} \]
Limits of the Original PFA

PFA is very time consuming for multiple faults [Zha+18]

\[ \lambda = 12 \Rightarrow |K| = 12^{16} \approx 2^{57.36} \]
More Limits of PFA and Its Enhanced Versions

- The location of the injected fault is supposed to be known
- For multiple fault injections:
  - We need a known plaintext/ciphertext pair to detect the correct key
- PFA only exploits the fault leakage in the last round
- Enhanced PFA (EPFA) [Xu+21] exploits the fault leakage in multiple rounds
- However, EPFA is not clear about exploiting multiple faults in deeper rounds
- Moreover, EPFA still relies on the assumption of knowing the fault location
More Limits of PFA and Its Enhanced Versions

- The **location of the injected** fault is supposed to be known

- For multiple fault injections:
  - We need a known plaintext/ciphertext pair to detect the correct key

- PFA only exploits the fault leakage in the last round

- Enhanced PFA (EPFA) [Xu+21] exploits the fault leakage in multiple rounds

- However, EPFA is not clear about exploiting multiple faults in deeper rounds

- Moreover, EPFA still relies on the assumption of knowing the **fault location**
More Limits of PFA and Its Enhanced Versions

- The location of the injected fault is supposed to be known.
- For multiple fault injections:
  - We need a known plaintext/ciphertext pair to detect the correct key.
- PFA only exploits the fault leakage in the last round.
  - Enhanced PFA (EPFA) [Xu+21] exploits the fault leakage in multiple rounds.
  - However, EPFA is not clear about exploiting multiple faults in deeper rounds.
  - Moreover, EPFA still relies on the assumption of knowing the fault location.
More Limits of PFA and Its Enhanced Versions

- The **location of the injected** fault is supposed to be known
- For multiple fault injections:
  - We need a known plaintext/ciphertext pair to detect the correct key
- PFA only exploits the fault leakage in the last round
- Enhanced PFA (EPFA) [Xu+21] exploits the fault leakage in multiple rounds
  - However, EPFA is not clear about exploiting multiple faults in deeper rounds
  - Moreover, EPFA still relies on the assumption of knowing the fault location
More Limits of PFA and Its Enhanced Versions

- The location of the injected fault is supposed to be known
- For multiple fault injections:
  - We need a known plaintext/ciphertext pair to detect the correct key
- PFA only exploits the fault leakage in the last round
- Enhanced PFA (EPFA) [Xu+21] exploits the fault leakage in multiple rounds
- However, EPFA is not clear about exploiting multiple faults in deeper rounds
  - Moreover, EPFA still relies on the assumption of knowing the fault location
More Limits of PFA and Its Enhanced Versions

- The location of the injected fault is supposed to be known
- For multiple fault injections:
  - We need a known plaintext/ciphertext pair to detect the correct key
- PFA only exploits the fault leakage in the last round
- Enhanced PFA (EPFA) [Xu+21] exploits the fault leakage in multiple rounds
- However, EPFA is not clear about exploiting multiple faults in deeper rounds
- Moreover, EPFA still relies on the assumption of knowing the fault location
Our Framework for PFA With Multiple Faults
Core Idea

- **V**: Impossible values in the output of faulty S-box
- **D[i]**: Impossible values in the \(i\)th word of ciphertext
- \(D[i] = V \oplus K[i]\) for all \(i \in \{1, \ldots, 15\}\)
- \(V = K[0] \oplus D[0]\)
- \(D[i] = (K[0] \oplus K[i]) \oplus D[0]\)
- \(\delta[i] = K[0] \oplus K[i]\)
- We can derive \(\delta[i]\) from \((D[0], D[i])\)
Core Idea

- \( V \): Impossible values in the output of faulty S-box
- \( D[i] \): Impossible values in the \( i \)th word of ciphertext
- \( D[i] = V \oplus K[i] \) for all \( i \in \{1, \ldots, 15\} \)
- \( V = K[0] \oplus D[0] \)
- \( D[i] = (K[0] \oplus K[i]) \oplus D[0] \)
- \( \delta[i] = K[0] \oplus K[i] \)
- We can derive \( \delta[i] \) from \( (D[0], D[i]) \)

Hadi Soleimany, Nasour Bagheri, **Hosein Hadipour**, Prasanna Ravi, Shivam Bhasin, Sara Mansouri
CHES 2022 - Leuven, Belgium
Core Idea

- $V$: Impossible values in the output of faulty S-box
- $D[i]$: Impossible values in the $i$th word of ciphertext
- $D[i] = V \oplus K[i] \text{ for all } i \in \{1, \ldots, 15\}$
- $V = K[0] \oplus D[0]$
  - $D[i] = (K[0] \oplus K[i]) \oplus D[0]$
  - $\delta[i] = K[0] \oplus K[i]$
- We can derive $\delta[i]$ from $(D[0], D[i])$
Core Idea

- **V**: Impossible values in the output of faulty S-box
- **D[i]**: Impossible values in the *i*th word of ciphertext
- **D[i] = V ⊕ K[i]** for all *i* ∈ {1, ..., 15}
- **V = K[0] ⊕ D[0]**
- **D[i] = (K[0] ⊕ K[i]) ⊕ D[0]**
- **δ[i] = K[0] ⊕ K[i]**
- We can derive **δ[i]** from (**D[0]**, **D[i])**

Hadi Soleimany, Nasour Bagheri, **Hosein Hadipour**, Prasanna Ravi, Shivam Bhasin, Sara Mansouri

CHES 2022 - Leuven, Belgium
Reducing the Number of Key Candidates to $2^8$

Guess $K[0]$ and determine $K[i]$ for all $i \in \{1, \ldots, 15\}$. So, $|K| = 2^8$ for AES!
Reducing the Number of Key Candidates to $2^8$

\[
\begin{align*}
\delta[1] &= K[0] \oplus K[1] \\
\delta[2] &= K[0] \oplus K[2]
\end{align*}
\]

Guess $K[0]$ and determine $K[i]$ for all $i \in \{1, \ldots, 15\}$. So, $|K| = 2^8$ for AES!
Reducing the Number of Key Candidates to $2^8$

Guess $K[0]$ and determine $K[i]$ for all $i \in \{1, \ldots, 15\}$. So, $|K| = 2^8$ for AES!
Reducing the Number of Key Candidates to $2^8$

Guess $K[0]$ and determine $K[i]$ for all $i \in \{1, \ldots, 15\}$. So, $|K| = 2^8$ for AES!
Reducing the Number of Key Candidates to $2^8$

Guess $K[0]$ and determine $K[i]$ for all $i \in \{1, \ldots, 15\}$. So, $|K| = 2^8$ for AES!
Reducing the Number of Key Candidates to $2^8$

\[ \delta[1] = K[0] \oplus K[1] \]
\[ \delta[2] = K[0] \oplus K[2] \]
\[ \delta[3] = K[0] \oplus K[3] \]
\[ \delta[5] = K[0] \oplus K[5] \]
\[ \delta[6] = K[0] \oplus K[6] \]

\( \downarrow \)

Guess \( K[0] \) and determine \( K[i] \) for all \( i \in \{1, \ldots, 15\} \). So, \( |K| = 2^8 \) for AES!
Guess $K[0]$ and determine $K[i]$ for all $i \in \{1, \ldots, 15\}$. So, $|K| = 2^8$ for AES!
A Generic Key Recovery Framework
Going Deeper Into the Decryption Rounds

- For each key, compute the impossible values of S-box ($K \Rightarrow V$)
- Go deeper into the decryption to filter more wrong keys
- Challenge: the faulty S-box is not invertible
- We use the correct S-box for decryption
- We consider the wrong key assumption
Going Deeper Into the Decryption Rounds

- For each key, compute the impossible values of S-box \((K \Rightarrow V)\)
- Go deeper into the decryption to filter more wrong keys
- Challenge: the faulty S-box is not invertible
- We use the correct S-box for decryption
- We consider the wrong key assumption
Going Deeper Into the Decryption Rounds

- For each key, compute the impossible values of S-box ($K \Rightarrow V$)
- Go deeper into the decryption to filter more wrong keys

⚠️ Challenge: the faulty S-box is not invertible

- We use the correct S-box for decryption
- We consider the wrong key assumption
Going Deeper Into the Decryption Rounds

- For each key, compute the impossible values of S-box \((K \Rightarrow V)\)
- Go deeper into the decryption to filter more wrong keys
- Challenge: the faulty S-box is not invertible
- We use the correct S-box for decryption
- We consider the wrong key assumption
Our Key-recovery Framework

**Input:** Key candidates

**Output:** Master key

1. **for each key candidate** $K$ **do**
2. $V \leftarrow K[0] \oplus D[0]$;
3. $\text{cnt}[K, V] \leftarrow 0$;
4. **foreach** faulty ciphertext **do**
5. **for** $r = R - 1, \ldots, 1$ **do**
6. Compute $Y_r$;
7. **foreach** cell of $Y_r$, i.e., $Y_r[j]$ **do**
8. **if** $Y_r[j] \in V$ **then**
9. Go to line 4
10. $\text{cnt}[K, V] \leftarrow \text{cnt}[K, V] + 1$;
11. **return** key with maximum $\text{cnt}[K, V]$;

\[
p = \left(1 - \frac{|V|}{256}\right)^{16}, \quad \text{cnt}_w = N \sum_{r=1}^{R-1} p^r, \quad \text{cnt}_c = N \sum_{r=1}^{R-1} p^r +
\]
Our Key-recovery Framework

**Input:** Key candidates

**Output:** Master key

1. **for each key candidate** $K$ **do**
   
   2. $V \leftarrow K[0] \oplus D[0]$;
   
   3. $\text{cnt}[K, V] \leftarrow 0$;
   
   4. **foreach faulty ciphertext** do
      
      5. **for** $r = R - 1, \ldots, 1$ **do**
         
         6. Compute $Y_r$;
         
         7. **foreach cell of $Y_r$, i.e., $Y_r[j]$** do
            
            8. **if** $Y_r[j] \in V$ **then**
               
               9. Go to line 4
               
         10. $\text{cnt}[K, V] \leftarrow \text{cnt}[K, V] + 1$;
      
      11. **return** key with maximum $\text{cnt}[K, V]$;

\[
p = \left(1 - \frac{|V|}{256}\right)^{16}, \quad \text{cnt}_w = N \sum_{r=1}^{R-1} p^r, \quad \text{cnt}_c = N \sum_{r=1}^{R-1} p^r + \]

Hadi Soleimany, Nasour Bagheri, **Hosein Hadipour**, Prasanna Ravi, Shivam Bhasin, Sara Mansouri

CHES 2022 - Leuven, Belgium
Our Key-recovery Framework

**Input:** Key candidates

**Output:** Master key

1. **for each key candidate** $K$ **do**
   
   2. $V \leftarrow K[0] \oplus D[0]$;
   
   3. $\text{cnt}[K, V] \leftarrow 0$;
   
   4. **foreach** faulty ciphertext **do**

   5. **for** $r = R - 1, \ldots, 1$ **do**

   6. Compute $Y_r$;

   7. **foreach** cell of $Y_r$, i.e., $Y_r[j]$ **do**

   8. **if** $Y_r[j] \in V$ **then**

   9. Go to line 4

10. $\text{cnt}[K, V] \leftarrow \text{cnt}[K, V] + 1$;

11. **return** key with maximum $\text{cnt}[K, V]$;

\[ p = \left(1 - \frac{|V|}{256}\right)^{16}, \quad \text{cnt}_w = N \sum_{r=1}^{R-1} p^r, \quad \text{cnt}_c = N \sum_{r=1}^{R-1} p^r + \]

Hadi Soleimany, Nasour Bagheri, Hosein Hadipour, Prasanna Ravi, Shivam Bhasin, Sara Mansouri

CHES 2022 - Leuven, Belgium
Our Key-recovery Framework

Input: Key candidates
Output: Master key

1 for each key candidate $K$ do
   2 $V \leftarrow K[0] \oplus D[0]$;
   3 $\text{cnt}[K, V] \leftarrow 0$;
   4 foreach faulty ciphertext do
      5 for $r = R-1, \ldots, 1$ do
         6 Compute $Y_r$;
         7 foreach cell of $Y_r$, i.e., $Y_r[j]$ do
            8 if $Y_r[j] \in V$ then
               9 Go to line 4
            10 $\text{cnt}[K, V] \leftarrow \text{cnt}[K, V] + 1$;
   11 return key with maximum $\text{cnt}[K, V]$;

\[
p = \left(1 - \frac{|V|}{256}\right)^{16}, \quad \text{cnt}_w = N \sum_{r=1}^{R-1} p^r, \quad \text{cnt}_c = N \sum_{r=1}^{R-1} p^r +
\]
Our Key-recovery Framework

Input: Key candidates
Output: Master key

1 for each key candidate $K$ do
   2 $V \leftarrow K[0] \oplus D[0]$;
   3 $\text{cnt}[K, V] \leftarrow 0$;
   4 foreach faulty ciphertext do
      5 for $r = R - 1, \ldots, 1$ do
         6 Compute $Y_r$;
         7 foreach cell of $Y_r$, i.e., $Y_r[j]$ do
            8 if $Y_r[j] \in V$ then
               9 Go to line 4
            10 else $\text{cnt}[K, V] \leftarrow \text{cnt}[K, V] + 1$;
   11 return key with maximum $\text{cnt}[K, V]$;

\[
p = \left(1 - \frac{|V|}{256}\right)^{16}, \quad \text{cnt}_w = N \sum_{r=1}^{R-1} p^r, \quad \text{cnt}_c = N \sum_{r=1}^{R-1} p^r + N \sum_{r=1}^{R-1} rp^r(1 - p)
\]

Hadi Soleimany, Nasour Bagheri, Hosein Hadipour, Prasanna Ravi, Shivam Bhasin, Sara Mansouri
CHES 2022 - Leuven, Belgium
Experimental Verification

$\lambda = 6$, $N = 1526$, $|K| = 256$

Exp: $cnt_w = 3197.91$, $cnt_c = 6086.93$

The: $cnt_w = 3197.89$, $cnt_c = 6983.73$
Experimental Verification

\[ \lambda = 6, \ N = 1526, \ |K| = 256 \]

Exp: \( \text{cnt}_w = 3197.91, \ \text{cnt}_c = 6086.93 \)

The: \( \text{cnt}_w = 3197.89, \ \text{cnt}_c = 6983.73 \)
Conclusion
Our Main Contributions

- We removed the assumption of knowing the fault location in PFA
- Our new technique decreases the number of key candidates by a factor of $\approx 2^{50}$
- We exploit the fault leakages in deeper rounds (until the first round)
- Our new technique reduces the number of required ciphertexts (refer to our paper)

Thanks for your attention!

https://github.com/hadipourh/faultyaes
