Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures

Christoph Dobraunig, Maria Eichlseder, Hannes Gross, Stefan Mangard, Florian Mendel, Robert Primas

ASIACRYPT 2018

IAIK - Graz University of Technology
Building cryptographic implementations is challenging:

- Requires usage of proper cryptographic primitives
- But often also the usage of additional defenses...
- Microcontroller
- FPGAs
- ASICs
- ... because of implementation attacks
Building cryptographic implementations is challenging:

- Requires usage of proper cryptographic primitives
Building cryptographic implementations is challenging:

- Requires usage of proper cryptographic primitives
- But often also the usage of additional defenses ...
  - Microcontroller
  - FPGAs
  - ASICs
Building cryptographic implementations is challenging:

- Requires usage of proper cryptographic primitives
- But often also the usage of additional defenses ...
  - Microcontroller
  - FPGAs
  - ASICs
- ... because of implementation attacks
Proper cryptography does not mean practical security
• Proper cryptography does not mean practical security
• Every cryptographic implementation stores a secret
Motivation

- Proper cryptography does not mean practical security
- Every cryptographic implementation stores a secret
- Secrets can be extracted by:
  - Power Analysis
  - Fault Attacks
Fault Attacks
Fault Attacks

• Get physical access to target device:
  • Set plaintexts
  • Observe ciphertexts
Fault Attacks

- Get physical access to target device:
  - Set plaintexts
  - Observe ciphertexts
- Cause erroneous computations via:
  - Clock glitches
  - Voltage glitches
  - Lasers

\[ P \rightarrow \text{ENC} \rightarrow C \rightarrow \text{Differential Fault Attack (DFA)} \]
Fault Attacks

- Get physical access to target device:
  - Set plaintexts
  - Observe ciphertexts

- Cause erroneous computations via:
  - Clock glitches
  - Voltage glitches
  - Lasers

- Observe faulty and correct ciphertext

⇒ Differential Fault Attack (DFA)
Fault Attacks

- Get physical access to target device:
  - Set plaintexts
  - Observe ciphertexts
- Cause erroneous computations via:
  - Clock glitches
  - Voltage glitches
  - Lasers
- Observe faulty and correct ciphertext
- Recover key

⇒ Differential Fault Attack (DFA)
Fault Attacks

- Get physical access to target device:
  - Set plaintexts
  - Observe ciphertexts
- Cause erroneous computations via:
  - Clock glitches
  - Voltage glitches
  - Lasers
- Observe faulty and correct ciphertext
- Recover key

⇒ Differential Fault Attack (DFA)
• Use redundancy to detect faults
• Use redundancy to detect faults
• Fault detected → No ciphertext
Fault Countermeasures - Detection

- Use redundancy to detect faults
- Fault detected $\rightarrow$ No ciphertext
- 2 identical faults necessary for attack
• Use redundancy to detect faults
• Fault detected $\rightarrow$ No ciphertext
• 2 identical faults necessary for attack
  $\rightarrow$ More redundancy, Enc-Dec, etc...
We presented SIFA at CHES 2018:

- Breaks detection countermeasures (any degree of redundancy)
- Breaks infection countermeasures
- Requires just a single fault injection per encryption
- Require no precise knowledge about location and effect of the fault
• We presented SIFA at CHES 2018:
  • Breaks detection countermeasures (any degree of redundancy)
  • Breaks infection countermeasures
  • Requires just a single fault injection per encryption
  • Require no precise knowledge about location and effect of the fault

• We demonstrated applicability to AE schemes at SAC 2018
• We presented SIFA at CHES 2018:
  • Breaks detection countermeasures (any degree of redundancy)
  • Breaks infection countermeasures
  • Requires just a single fault injection per encryption
  • Require no precise knowledge about location and effect of the fault

• We demonstrated applicability to AE schemes at SAC 2018

• What about power analysis countermeasures?
SIFA on AES in Pictures

- **ROUND 10**: Key Add 10, Shift Rows, Sub Bytes, Mix Columns, Key Add 8, Mix Columns, Shift Rows
- **ROUND 9**: Key Add 9, Shift Rows, Sub Bytes, Mix Columns, Key Add 9
- **ROUND 8**: Key Add 10, Shift Rows, Sub Bytes, Mix Columns, Key Add 10

Diagram:
- Input (P)
- Intermediate states (...)
- Output (C)

Robert Primas — IAIK - Graz University of Technology
SIFA on AES in Pictures

ROUND 10        ROUND 9          ROUND 8

KEY ADD 10
SHIFT ROWS
SUB BYTES
KEY ADD 9
SHIFT ROWS
SUB BYTES
MIX COLUMNS
KEY ADD 8
MIX COLUMNS
SHIFT ROWS
:
P1...N
CCCCCN
?
P1...N
CCCCCN

Robert Primas — IAIK - Graz University of Technology
SIFA on AES in Pictures

ROUND 10        ROUND 9          ROUND 8

KEY ADD 10
SHIFT ROWS
SUB BYTES
KEY ADD 9
SHIFT ROWS
SUB BYTES
MIX COLUMNS
KEY ADD 8
MIX COLUMNS
SHIFT ROWS
S

P1...N

SHIFT ROWS
MIX COLUMNS
KEY ADD 8
SUB BYTES
SHIFT ROWS
MIX COLUMNS
KEY ADD 9
SUB BYTES
SHIFT ROWS
KEY ADD 10

P1...N

CCCCCN

P1...N

CCCCCN

Robert Primas — IAIK - Graz University of Technology
SIFA on AES in Pictures

ROUND 10
SHIFT ROWS
SUB BYTES
KEY ADD 9
SHIFT ROWS
SUB BYTES
MIX COLUMNS
KEY ADD 8
MIX COLUMNS
SHIFT ROWS

ROUND 10
SHIFT ROWS
SUB BYTES
KEY ADD 10
SHIFT ROWS
SUB BYTES
MIX COLUMNS
KEY ADD 10
MIX COLUMNS
SHIFT ROWS

ROUND 9
SHIFT ROWS
SUB BYTES
KEY ADD 9
SHIFT ROWS
SUB BYTES
MIX COLUMNS
KEY ADD 9
MIX COLUMNS
SHIFT ROWS

ROUND 8
SHIFT ROWS
SUB BYTES
KEY ADD 8
SHIFT ROWS
SUB BYTES
MIX COLUMNS
KEY ADD 8
MIX COLUMNS
SHIFT ROWS

P_{1...N}

Robert Primas — IAIK - Graz University of Technology
SIFA on AES in Pictures

ROUND 10
SHIFT ROWS
SUB BYTES
KEY ADD 9
SHIFT ROWS
SUB BYTES
MIX COLUMNS
KEY ADD 8
MIX COLUMNS
SHIFT ROWS

ROUND 9
SUB BYTES
SHIFT ROWS
MIX COLUMNS
KEY ADD 9
SHIFT ROWS

ROUND 10
SHIFT ROWS
SUB BYTES
KEY ADD 10
SHIFT ROWS
SUB BYTES
MIX COLUMNS
KEY ADD 8
MIX COLUMNS
SHIFT ROWS

P1...N
Cn

Robert Primas — IAIK - Graz University of Technology
What about fault countermeasures?
*only correct computations are considered*
Ineffective Faults on AND

Robert Primas — IAIK - Graz University of Technology
Ineffective Faults on AND

\[
\begin{array}{ccccccc}
\text{ROUND 10} & \text{ROUND 9} & \text{ROUND 8} \\
\text{KEY ADD 10} & \text{SHIFT ROWS} & \text{SUB BYTES} \\
\text{KEY ADD 9} & \text{SHIFT ROWS} & \text{SUB BYTES} \\
\text{MIX COLUMNS} & \text{KEY ADD 8} & \text{MIX COLUMNS} \\
\text{SHIFT ROWS} & & \end{array}
\]

\[Ciphertext\]

\[
\begin{array}{ccc}
X & Y & Z \\
\text{Bitflip} & & \\
\sim & 0 & 1
\end{array}
\]

*only correct computations are considered*
Ineffective Faults on AND

*only correct computations are considered
Ineffective Faults on AND

Also works with:

- Other instructions: LOAD, STORE, XOR
- Other fault types: Random, Stuck-at, Skip

*only correct computations are considered
Power Analysis
• Circuits leak information via side-channels, e.g. power consumption
Circuits leak information via side-channels, e.g. power consumption
CMOS circuits draw power almost only in case of “events”
Circuits leak information via side-channels, e.g. power consumption.
CMOS circuits draw power almost only in case of “events”.
Correlation between processed data and power consumption.
Power Analysis

- Circuits leak information via side-channels, e.g. power consumption
- CMOS circuits draw power almost only in case of "events"
- Correlation between processed data and power consumption
- Problematic if processed data contains secrets
• Make power consumption independent of processed data
  - Requires hardware support (filters, noise generators)
• Make power consumption independent of processed data
  – Requires hardware support (filters, noise generators)
• Make processed data independent of the actual data
  + “Masking” can be done on algorithmic level
Masking Idea

- Split a value $x$ into multiple “shares” s.t.:
  - The XOR-sum over all $x_i$ equals $x$
  - The distribution of each $x_i$ is independent from $x$
• Split a value $x$ into multiple “shares” s.t.:
  • The XOR-sum over all $x_i$ equals $x$
  • The distribution of each $x_i$ is independent from $x$

• Linear function $f$ is performed separately
Masking Idea

- Split a value $x$ into multiple “shares” s.t.:
  - The XOR-sum over all $x_i$ equals $x$
  - The distribution of each $x_i$ is independent from $x$
- Linear function $f$ is performed separately
- Nonlinear functions $g$ need more attention:
  - $g^*$ works on all shares
  - $g^*$ avoids direct combinations of shares
• Split a value $x$ into multiple “shares” s.t.:
  • The XOR-sum over all $x_i$ equals $x$
  • The distribution of each $x_i$ is independent from $x$
• Linear function $f$ is performed separately
• Nonlinear functions $g$ need more attention:
  • $g^*$ works on all shares
  • $g^*$ avoids direct combinations of shares
• Applied to AES →

Does our attack still work?
• Faulting single shares in linear functions does not work...

\[ f + f \]
Faulting single shares in linear functions does not work...

Can faulting single shares in non-linear functions lead to a bias in the unshared value?
• Faulting single shares in linear functions does not work...
• Faulting all shares would work but is boring...
• Faulting single shares in linear functions does not work...
• Faulting all shares would work but is boring...
• Can faulting single shares in non-linear functions lead to a bias in the unshared value?
*masked AES, only correct computations are considered
Faults on Masked AND

\[ X_0 \times Y_0 \times Y_1 \times X_1 + R + Z_0 + Z_1 \]

*only correct computations are considered*
Faults on Masked AND

[Diagram of a circuit with nodes and labels X0, Y0, Y1, X1, Z0, Z1, and R, showing the logic operations AND and OR with notes that only correct computations are considered.]
**Faults on Masked AND**

*only correct computations are considered*
Faults on Masked AND

*only correct computations are considered
Faults on Masked AND

Also works with:

- Other types of faults
- Higher-order masking
- Threshold Implementations

*only correct computations are considered
Target: First-order masked AES by Schwabe and Stoffelen et al.

- Publicly available
Target: First-order masked AES by Schwabe and Stoffelen et al.

- Publicly available
- ARM Cortex M4, ASM optimized
Target: First-order masked AES by Schwabe and Stoffelen et al.

- Publicly available
- ARM Cortex M4, ASM optimized

→ Originally CTR mode encryption, we only use it as block cipher
Target: First-order masked AES by Schwabe and Stoffelen et al.

- Publicly available
- ARM Cortex M4, ASM optimized

→ Originally CTR mode encryption, we only use it as block cipher
→ Originally no fault countermeasures, we added “perfect” fault detection
For each individual instruction in the masked Sbox:

- Simulated fault: Single bitflip in the result
For each individual instruction in the masked Sbox:

- Simulated fault: Single bitflip in the result
- 2000 faulted Sbox computations, random inputs
For each individual instruction in the masked Sbox:

- Simulated fault: Single bitflip in the result
- 2000 faulted Sbox computations, random inputs
- Check if correct outputs are non-uniform, i.e. if key recovery would work
For each individual instruction in the masked Sbox:

- Simulated fault: Single bitflip in the result
- 2000 faulted Sbox computations, random inputs
- Check if correct outputs are non-uniform, i.e. if key recovery would work
For each individual instruction in the masked Sbox:

- Simulated fault: Single bitflip in the result
- 2000 faulted Sbox computations, random inputs
- Check if correct outputs are non-uniform, i.e. if key recovery would work

⇒ 52 % of instruction are “susceptible” to single bitflips
For each individual instruction in the masked Sbox:

- Simulated fault: Randomized 8 bits of the result
- 2000 faulted Sbox computations, random inputs
- Check if correct outputs are non-uniform, i.e. if key recovery would work

⇒ 70% of instruction are “susceptible” to random faults
Simulated Faults on Masked AES with Fault Detection

Exact numbers for one of the susceptible instructions

<table>
<thead>
<tr>
<th>Fault Effect</th>
<th># Ineffective Faults</th>
<th># Faulted Encryptions</th>
<th># Recoverable Key Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip one bit</td>
<td>194</td>
<td>386</td>
<td>32</td>
</tr>
<tr>
<td>Set one bit to zero</td>
<td>214</td>
<td>428</td>
<td>32</td>
</tr>
<tr>
<td>Randomize one bit</td>
<td>574</td>
<td>763</td>
<td>32</td>
</tr>
<tr>
<td>Flip one byte</td>
<td>192</td>
<td>2,940</td>
<td>128</td>
</tr>
<tr>
<td>Set one byte to zero</td>
<td>192</td>
<td>3,129</td>
<td>128</td>
</tr>
<tr>
<td>Randomize one byte</td>
<td>602</td>
<td>1,808</td>
<td>128</td>
</tr>
<tr>
<td>Instruction skip</td>
<td>400</td>
<td>45,527</td>
<td>128</td>
</tr>
</tbody>
</table>
Target: POC higher-order masked AES by Rivain et al.

- Setup: Clock glitches on ATXmega 128D4
Target: POC higher-order masked AES by Rivain et al.

- Setup: Clock glitches on ATXmega 128D4
  → We set masking order to 10
Target: POC higher-order masked AES by Rivain et al.

- Setup: Clock glitches on ATXmega 128D4
  → We set masking order to 10
  → We added “perfect” fault detection
Target: POC higher-order masked AES by Rivain et al.

- Setup: Clock glitches on ATXmega 128D4
  - We set masking order to 10
  - We added “perfect” fault detection

⇒ About 1000 faulted encryptions required
⇒ Thousands of possible fault locations
A word on other countermeasures

- Self Destruction
- Frequent Re-keying
- Multi Party Computation
SIFA is quite powerful...

- Works for many ciphers and AE schemes
SIFA is quite powerful...

- Works for many ciphers and AE schemes
- Breaks both fault and power analysis countermeasures
SIFA is quite powerful...

- Works for many ciphers and AE schemes
- Breaks both fault and power analysis countermeasures
- Attacker does not need to hit specific bits/bytes
SIFA is quite powerful...

- Works for many ciphers and AE schemes
- Breaks both fault and power analysis countermeasures
- Attacker does not need to hit specific bits/bytes
- Attacker does not need to know how the faults influence the computation