A DFA ON AES BASED ON THE ENTROPY OF ERROR DISTRIBUTIONS

FDTC2012 | Ronan Lashermes, Guillaume Reymond, Jean-Max Dutertre, Jacques Fournier, Bruno Robisson and Assia Tria

9 SEPTEMBER 2012
Introduction

In order to design secure cryptosystems, one has to assess the risks of potential attacks.

We want to discuss about the practical implementation of attacks, more precisely about the fault models.

We want a DFA:

- **General**: can be used with all injection means.
- **Adaptive**: the efficiency increases when the fault model is more restrictive.
- **Simple** to implement.
- **Without prior knowledge** of the fault model…
- **Or with prior knowledge** and higher efficiency.
- Helped by some countermeasures!
OUTLINE

Section 1 – Context

Section 2 – Entropy-based methodology

Section 3 – Improving entropy-based tools
Differential Fault Analysis

- Attacker corrupts one of the intermediate states of the AES.
- Attacker performs a differential cryptanalysis between the correct cipher (C) and the erroneous one (D) to infer information about the secret key.
The fault model is the set of restrictions put on the injected faults. Common examples are:
- Single bit faults \((2^{3 \times 16} = 2^{48}\) authorized faults on the State)
- Single byte faults \(((4 \times 2^8)^4 = 2^{40}\) authorized faults on the State)

Key extraction analyses are:
- Either restrictive (Giraud’s: \(2^{48}\), Piret’s: \(2^{40}\) …)
- Either inefficient: a high number of fault injections is required (Moradi’s: \(2^{127.9}\) …)

We represent a fault model with an error distribution. \((2^{128})\)
CLOCK GLITCHES

- Clock glitches create memorization faults in registers through setup time violations.
- Faults are probabilistic.
- Distributions can be used for all injection means.

'Length' of propagation delay associated to each bit

Clock period
SECTION 2
ENTROPY-BASED METHODOLOGY
In order to work, our analysis needs the following hypotheses:
- The faults are bit-flip.
- The faults are not uniformly distributed.*
- The faults are injected on M9.

* From now on we shall concentrate on individual bytes…

The correct key byte is noted $K_{10}$.

For each realization $i$:
- First a valid encryption is executed ($C_i$).
- Then a fault is injected on M9 and the faulty cipher value is memorized ($D_i$).

* A work based on a similar principle can be found in DFA on DES middle rounds by M. Rivain (CHES 2009)
From $C_i$ and $D_i$ (correct and faulty ciphers)

Given a key guess $s$,

The fault guess $e_{i,s}$ is computed with:

$$M9_{i,s} = SB^{-1}(C_i \oplus s)$$

$$e_{i,s} = M9_{i,s} \oplus SB^{-1}(D_i \oplus s)$$
**RK-table**

We can know construct the Realization/Key hypothesis (RK) table, filled with \((e_{i,s})\).

<table>
<thead>
<tr>
<th>Realization</th>
<th>Key</th>
<th>0</th>
<th>1</th>
<th>...</th>
<th>255</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>(e_{0,0})</td>
<td>(e_{0,1})</td>
<td>...</td>
<td>(e_{0,255})</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>(e_{1,0})</td>
<td>(e_{1,1})</td>
<td>...</td>
<td>(e_{1,255})</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(i_{\text{max}})</td>
<td>(\ldots)</td>
<td>(e_{i_{\text{max}},0})</td>
<td>(e_{i_{\text{max}},1})</td>
<td>...</td>
<td>(e_{i_{\text{max}},255})</td>
</tr>
</tbody>
</table>

This table has two interesting properties:
- **Only one column** (for \(s = K10\)) corresponds to faults actually injected.
- For every wrong key guess, the corresponding column is quasi-random.
Finding the correct column

The uniformity of a distribution is simply determined with Shannon entropy:

\[ H(p_s) = - \sum_{e=0}^{255} p_s(e) \log_2 p_s(e) \]

Decision criterion:

\[ H(p_s) \xrightarrow[i_{\text{max}} \to \infty]{} 8 \text{ if } s \neq K10 \]

\[ H(p_{K10}) \xrightarrow[i_{\text{max}} \to \infty]{} H_{\text{inj}} < 8 \]

Valid only for sets of faults of infinite size
Finding the correct column with a finite number of realizations

- Comparison with pseudo-random sets.
- \( i_{max} \): number of realizations, \( \mu_{i_{max}}^{rand} \): the mean, \( \sigma_{i_{max}}^{rand} \): the standard deviation.
- \( H(p_s) \) the measured entropy for the key guess \( s \).
- We can express the confidence \( cf \) that an entropy of value \( H \) is not random by:

\[
 cf_{i_{max}}(H) = \frac{\mu_{i_{max}}^{rand} - H}{\sigma_{i_{max}}^{rand}}
\]

- Decision criterion:

\[
 K10 = s \Leftrightarrow cf_{i_{max}}(H(p_s)) > X
\]

- We chose with empirical calibration \( X = 6 \)
$H_{\text{inj}} = 7.76$

$i_{\text{max}} = 500$
$H_{\text{inj}} = 7.76$
Using simulation, the entropy of the injection means may be linked with the attack efficiency.

Attack efficiency is the average minimum number of faults needed to meet the decision criterion.
ENTROPY: SUMMARY

Our DFA is:

- **General**: can be used with all injection means.
- **Adaptive**: the efficiency increases when the fault model is tighter.
- **Simple** to implement.
- **Without prior knowledge** of the fault model…
- **Or with prior knowledge** and higher efficiency.
- Helped by some countermeasures!

It is not particularly **efficient**: can we improve it?

<table>
<thead>
<tr>
<th>Average best attack</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shannon entropy</strong></td>
</tr>
<tr>
<td><strong>Giraud’s</strong></td>
</tr>
</tbody>
</table>

Perfect single bit faults (simulation)
SECTION 3
IMPROVING ENTROPY-BASED TOOLS
Considering a known fault model

- We want to improve the efficiency of the attack by including information of a known model.
- Let $t(e)$ be the expected distribution, we use the relative entropy:

$$RH(p_s, t) = \sum_{e=0}^{255} p_s(e) \log_2 \left( \frac{p_s(e)}{t(e)} \right)$$

<table>
<thead>
<tr>
<th></th>
<th>Average best attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shannon entropy</td>
<td>6.41</td>
</tr>
<tr>
<td>Relative entropy</td>
<td>2.24</td>
</tr>
<tr>
<td>Giraud’s</td>
<td>2.24</td>
</tr>
</tbody>
</table>

Perfect single bit faults (simulation)
How to learn the fault model $t(e)$

- Use the Shannon entropy in a first attack.
- Inject faults on M10 and observe the resulting fault model.
- We have previous knowledge of the system, the injection means, the countermeasure…

- Bertoni’s countermeasure = 1 parity bit
- Thus all odd bit faults are eliminated. This creates non uniformity!
Modeling basic countermeasures

- $d(e)$ is the detection rate for error $e$.
- $D = \sum_{e=0}^{255} p_{K10}(e) d(e)$ is the global detection rate.

Two cases:

- Virtual model with result discrimination: the attacker knows for which realizations the countermeasure was activated. The new “virtual” distribution is:
  
  $$v(e) = \frac{p_{K10}(e)(1 - d(e))}{1 - D}$$

- Virtual model without result discrimination: the attacker does not know for which realizations the countermeasure was activated. The new “virtual” distribution is:
  
  $$w(e) = \frac{1}{256} D + p_{K10}(e)(1 - d(e)) = \frac{1}{256} D + (1 - D)v(e)$$
Bertoni’s with error discrimination
Simulation
\( H(v) = 6.86 \)
Bertoni’s without error discrimination
Simulation
$H(v) = 7.78$
Conclusion

- Our DFA is:
  - **General**: can be used with all injection means.
  - **Adaptive**: the efficiency increases when the fault model is tighter.
  - **Simple** to implement.
  - **Without prior knowledge** of the fault model...
  - **Or with prior knowledge** and higher efficiency.
  - **Helped by some countermeasures!**

- We **loosened the constraints** on the injection means.
- We can find the **key** and the **fault model** in parallel.
- All faults contribute to find the key. The analysis is done by taking into account all faults as a whole.

- **Countermeasures** must create non uniformity.
Conclusions

Perspectives

- Verify that all injection means have non-uniform distribution for injected faults.
- Represent the fault model with something different than a distribution.
- Test this methodology on other algorithms. It should work if we can compute the injected faults with the secret as a parameter.
- Cartography for localized injection means should include a fault entropy evaluation.
Thank you for your attention.

Any questions?