A DFA attack on White-box implementations of AES with external encoding

WhibOx 2019: White-Box Cryptography and Obfuscation, 18-19/05/2019, Darmstadt

Alessandro Amadori, Wil Michiels and Peter Roelse
White-box Cryptography and Side Channel Attacks

A very quick introduction
AES-128 is a block cipher
- 128-bit plaintext
- 128-bit key
- Rearranged bits
- 10 rounds
Attacks in a White-box Scenario

- In a White-box Attack scenario an attacker:
  - has full access to implementation;
  - can modify part of the implementation;
  - can observe the execution of the algorithm;

- Algebraic attacks on source-code generally require:
  - Reverse engineering;
  - De-obfuscation;
  - Attack-strategies based on the implementation;
Side Channel Attacks (DCA/DFA)

- Advantages:
  - Can be automated;
  - Require little-to-no reverse engineering.
- Differential Computational Analysis (DCA) is the software counterpart of Differential Power Analysis (DPA).
- Differential Fault Analysis (DFA) introduces faults during execution.
  - Inject faults at Round 9 (4 faulty output bytes);
  - Set up system:
    \[
    \begin{align*}
    S^{-1}(x_0 \oplus k_0) \oplus S^{-1}(X_0 \oplus k_0) &= 2 \ (S^{-1}(x_1 \oplus k_1) \oplus S^{-1}(X_1 \oplus k_1)) \\
    S^{-1}(x_2 \oplus k_2) \oplus S^{-1}(X_2 \oplus k_2) &= S^{-1}(x_1 \oplus k_1) \oplus S^{-1}(X_1 \oplus k_1) \\
    S^{-1}(x_3 \oplus k_3) \oplus S^{-1}(X_3 \oplus k_3) &= 3 \ (S^{-1}(x_1 \oplus k_1) \oplus S^{-1}(X_1 \oplus k_1))
    \end{align*}
    \]
  - Solve the system to obtain the round key.
External Encodings

- Input or output of the executable may be encoded
  - Composition of random non-linear and linear functions
  - Input is encoded/output is decoded by another party
- Prevent from code-lifting
- Prevent from some algebraic attacks
External encodings as countermeasures to SCA

- “Therefore, DFA attacks on encoded outputs are not feasible either.”
  
  Unboxing the White-box, Sanfelix, Mune, de Haas, BlackHat 2016.

- “Another potential countermeasure against DCA is the use of external encodings. This was the primary reason why we were not able to extract the secret key [...]”


*Photo Courtesy by Lorenz Panny*
Attack WB implementations with simple output
External Encodings with DFA
Our Model

- External encodings proposed by Chow et al.: 128-bit matrix multiplication and non-linear byte encodings.
- Main objective: Use first-order fault injection attack to extract key
- External encoding given by non-linear byte encodings.
Our Assumptions

- No reverse engineering;
- Operations may not be aligned;
- For any S-box in/out $x$ there exists at least 1 location in a *single* execution where we can change $x$ to any of its possible 256 values
  - Masking, internal encodings and embedding
- Adversary can guess with good probability the location of an S-box
  - E.g. Checking if 4 output bytes have been altered
    - Different values for different faults
Before we start off:

a quick thing

- \( E_i() \rightarrow i^{th} \) output byte encoding
- \( \oplus \rightarrow \) bitwise XOR
- \( x_i \rightarrow i^{th} \) correct output byte
- \( X_i \rightarrow i^{th} \) faulty output byte
- \( S() \rightarrow \) AES S-box
- \( MC() \rightarrow \) AES MixColumns
- Ignore Round 10 ShiftRows
Outline of the Attack

• Step 1: Pre-computation

• Step 2: Reconstruction of the 9\textsuperscript{th} round output up to affine bit-functions

• Step 3: Reconstruction of the 9\textsuperscript{th} round output up to affine byte-functions
  • Step 3/4: Reduction of number of variables

• Step 4: Complete reconstruction of the 9\textsuperscript{th} round SubBytes output

• Step 5: Recovery of the 8\textsuperscript{th} round key
Step 1: Pre-computation

• Construct bins of plaintexts $M_0, M_1, \ldots, M_{15}$
  • Necessary to perform Step 2
  • One for every output byte
  • Every $p$ in $M_i$ satisfies the following properties:
    • For all $p$ in $M_i$, $i$th ciphertext output bytes are unique
    • The output values of two other indexes in the same column are fixed

• Example: $M_0 = \{p_0, p_1, \ldots, p_{255}\}$

\[
\begin{align*}
p_0 & \rightarrow c_0 = (0x02, 0x34, 0x56, \ldots) \\
p_1 & \rightarrow c_1 = (0xf4, 0x34, 0x56, \ldots) \\
& \ldots \\
p_{255} & \rightarrow c_{255} = (0xc6, 0x34, 0x56, \ldots)
\end{align*}
\]
**Step 2**

- Inject faults at round 9;
- As for DFA, set up the system:

\[
g_0^{-1}(x_0) \oplus g_0^{-1}(X_0) = 2 ( g_1^{-1}(x_1) \oplus g_1^{-1}(X_1) )
\]

\[
g_2^{-1}(x_2) \oplus g_2^{-1}(x_2) = g_1^{-1}(x_1) \oplus g_1^{-1}(x_1)
\]

\[
g_3^{-1}(x_3) \oplus g_3^{-1}(x_3) = 3 ( g_1^{-1}(x_1) \oplus g_1^{-1}(X_1) )
\]

- \(g_i^{-1}(x_i) = S^{-1}(E_i^{-1}(x_i) \oplus k_i)\)

- The output of \(g_i^{-1}\) is the input of Round 10.
Step 2 (cont.)

• Using a theorem from the BGE attack, if we have functions $g_i(\oplus_\alpha(g^{-1}_i(.)))$, we can derive a non-linear function $g_i$
  • $g_i = g_i \circ g_i^{-1}$
  • $g_i$ is an affine unknown function

• $g_0^{-1}(x_0) \oplus g_0^{-1}(X_0) = 2(g_1^{-1}(x_1) \oplus g_1^{-1}(X_1))$

• To provide a correct construction:
  • one byte must assume all possible values
  • an output byte must stay fixed

• We use the bin $M_i$
  • We inject all byte values for every plaintext in $M_i$

• Why a second fixed byte?
Step 2 (cont.)

• Faults must be introduced for every plaintext.
  • The same S-box must be affected
  • Possible execution misalignments for different plaintexts

• This is where the second fixed byte comes in action:
  • Comparing faulty outputs on fixed bytes:
    • It is possible to check if two injections affected the same S-Box
  
  • No information about which S-box
    • Not necessary
Step 3

- Inject faults at Round 9
- Consider the set of equations

\[
g_0^{-1}(x_0) \oplus g_0^{-1}(X_0) = 2(g_1^{-1}(x_1) \oplus g_1^{-1}(X_1))
\]
\[
g_2^{-1}(x_2) \oplus g_2^{-1}(X_2) = g_1^{-1}(x_1) \oplus g_1^{-1}(X_1)
\]
\[
g_3^{-1}(x_3) \oplus g_3^{-1}(X_3) = 3(g_1^{-1}(x_1) \oplus g_1^{-1}(X_1))
\]

\[x_i = g_i^{-1}(x_i)\]

\[g_i^{-1}(x_i) = G_i^{-1}(x_i \oplus b_i)\]

Using another Theorem of BGE attack, if we have a function \(G_i \circ \gamma \circ G_i^{-1}\) we derive a linear function \(g_i\)

- \(G_i = g_i \circ \lambda_i^{-1}\)
- \(\lambda_i^{-1}\) is an unknown non-zero factor
Step 3 (cont.)

- We need to construct a function of the form $G_i \circ \gamma \circ G_i^{-1}$
  - $\gamma$ is a particular known constant (derived from MC coefficients)

- We inject faults affecting 2 different S-boxes in different executions
  
  $G_i^{-1}(x_0 \oplus X_0) = 2(G_i^{-1}(x_1 \oplus X_1))$
  
  $G_i^{-1}(x_0 \oplus X_0) = 2^{-13}(G_i^{-1}(x_1 \oplus X_1))$

- $G_i^0(2(G_i^{-1}(.)))$ and $G_i^0(2^{-13}(G_i^{-1}(.)))$

- $\gamma$ is unknown but computable! (check the eigenvalues).
- For some indexes, we can infer the targeted S-Boxes.
- Any pair of positions and output bytes works!

- We construct an encoded output of Round 9 $y_i$ such that
  
  $y_i = g_i^{-1}(x_i)$
  
  $y_i = \lambda_i y_i \oplus b_i$
  
  $y_i$ is the non-encoded output of Round 9
Step 3/4

Knowing that:

- $G_i = g_i \circ \lambda_i^{-1}$,
- $y_i = g_i^{-1}(x_i)$ and
- 

\[
\begin{align*}
G_0^{-1}(x_0 \oplus X_0) &= 2(G_1^{-1}(x_1 \oplus X_1)) \\
G_2^{-1}(x_2 \oplus X_2) &= G_1^{-1}(x_1 \oplus X_1) \\
G_3^{-1}(x_3 \oplus X_3) &= 3(G_1^{-1}(x_1 \oplus X_1))
\end{align*}
\]

We construct a dependency among $\lambda_i$

- $\lambda_0^{-1}(y_0 \oplus Y_0) = 2 (\lambda_1^{-1}(y_1 \oplus Y_1))$
- $\lambda_2^{-1}(y_2 \oplus Y_2) = \lambda_1^{-1}(y_1 \oplus Y_1)$
- $\lambda_3^{-1}(y_3 \oplus Y_3) = 3 (\lambda_1^{-1}(y_1 \oplus Y_1))$

- $\lambda_1^{-1} = c_1 \lambda_0^{-1}$, $\lambda_2 = c_2 \lambda_0^{-1}$, $\lambda_3 = c_3 \lambda_0^{-1}$.
- $c_1, c_2, c_3$ are computable.
We obtain an “encoded” S-Box output of round 9 \((z_0, z_1, ..., z_{15})\) from \((y_0, y_1, ..., y_{15})\) by reverting AES operations (without considering key addition).

Inject faults at Round 8:

\[
\begin{align*}
S^{-1}(\lambda_0^{-1}z_0 \oplus \beta_0) \oplus S^{-1}(\lambda_0^{-1}Z_0 \oplus \beta_0) &= 2(S^{-1}(\lambda_4^{-1}z_1 \oplus \beta_1) \oplus S^{-1}(\lambda_4^{-1}Z_1 \oplus \beta_1)) \\
S^{-1}(\lambda_8^{-1}z_2 \oplus \beta_2) \oplus S^{-1}(\lambda_8^{-1}Z_2 \oplus \beta_2) &= S^{-1}(\lambda_4^{-1}z_1 \oplus \beta_1) \oplus S^{-1}(\lambda_4^{-1}Z_1 \oplus \beta_1) \\
S^{-1}(\lambda_{12}^{-1}z_3 \oplus \beta_3) \oplus S^{-1}(\lambda_{12}^{-1}Z_3 \oplus \beta_3) &= 3(S^{-1}(\lambda_4^{-1}z_1 \oplus \beta_1) \oplus S^{-1}(\lambda_4^{-1}Z_1 \oplus \beta_1))
\end{align*}
\]

The unknowns are \(\lambda_i^{-1}\) and \(\beta_i\)

- They contain the remaining randomness
Step 4 (cont.)

- Exhaustive search is unfeasible,
  - $2^{64}$ operations
- We use a MITM approach with hash tables:
  - $S^{-1}(\lambda_{4}^{-1}z_{1} \oplus \beta_{1}) \oplus S^{-1}(\lambda_{4}^{-1}Z_{1} \oplus \beta_{1})$ in every equation

- Consider
  
  
  $2^{-1}(S^{-1}(\lambda_{0}^{-1}z_{0} \oplus \beta_{0}) \oplus S^{-1}(\lambda_{0}^{-1}Z_{0} \oplus \beta_{0})) = S^{-1}(\lambda_{4}^{-1}z_{1} \oplus \beta_{1}) \oplus S^{-1}(\lambda_{4}^{-1}Z_{1} \oplus \beta_{1})$

- For all $\lambda$ and $\beta$ we compute $S^{-1}(\lambda z_{1} \oplus \beta) \oplus S^{-1}(\lambda Z_{1} \oplus \beta)$
  - Store them in an Hash Table
- For all $\lambda$ and $\beta$ we compute $2^{-1}(S^{-1}(\lambda z_{0} \oplus \beta) \oplus S^{-1}(\lambda Z_{0} \oplus \beta))$
  - Check if we have a match in the hash table
  - If yes: $(\lambda, \beta, \lambda, \beta)$ is a solution
  - $(\lambda_{0}^{-1}, \beta_{0}, \lambda_{4}^{-1}, \beta_{1})$ must belong to the set of solutions

- We apply this process for $\omega$ faults
Step 4 (cont.)

- Higher $\omega \rightarrow$ more accuracy
  - $\omega = 8$ only one solution is found (in about 5 min)

- If injecting at the wrong spot: No solution for the system.

- After retrieving all the $\lambda_i^{-1}$ and the $\beta_i$:
  - We are able to decode the output of the Round 9 S-box.
  - From encoded Round 9 S-Box output $(z_0, z_1, \ldots, z_{15})$ compute $z_i = \lambda_i^{-1}z_i \oplus \beta_i$
• From the decoded Round 9 S-box output \((z_0, z_1, \ldots, z_{15})\) compute the non-encoded Round 8 S-Box output \((w_0, w_1, \ldots, w_{15})\) as in Step 4.

• Inject faults at Round 7: set up and solve the standard equations

\[
S^{-1}(w_0 \oplus k_0) \oplus S^{-1}(w_0 \oplus k_0) = 2(S^{-1}(w_{13} \oplus k_{13}) \oplus S^{-1}(W_{13} \oplus k_{13}))
\]
\[
S^{-1}(w_{10} \oplus k_{10}) \oplus S^{-1}(W_{10} \oplus k_{10}) = S^{-1}(w_{13} \oplus k_{13}) \oplus S^{-1}(W_{13} \oplus k_{13})
\]
\[
S^{-1}(w_7 \oplus k_7) \oplus S^{-1}(W_7 \oplus k_7) = 3(S^{-1}(w_{13} \oplus k_{13}) \oplus S^{-1}(W_{13} \oplus k_{13}))
\]

• Obtain the values for \(k\)
  • MITM-approach is very efficient.
  • Round 8 key is \(\text{MC}(k)\).
  • Revert the Key-Scheduling algorithm to obtain the encryption key.
Work load

- Step 1: \(\rightarrow \sim 2^{31}\) WB encryptions, 0 operations
- Step 2: \(\rightarrow \sim 2^{20}\) WB encryption, \(2^{18}\) operations
- Step 3: \(\rightarrow \sim 2^{10}\) WB encryptions, \(2^{20}\) operations
  - Step 3/4: \(\rightarrow 0\) WB encryptions, 12 operations
  \(\omega 2^{19}\) operations
- Step 4: \(\rightarrow 4\omega\) WB encryptions, \(\omega' 2^{13}\) operations
- Step 5

\(< 2^{32}\) WB encryptions
\(< 2^{22}\) operations
Summary

• We perform the attack stepwise:
  • Construct last round up to some function
  • Remove the randomness and retrieve non-encoded state
  • Extract round-8 key

• Open Problems/Future work:
  • Work on assumptions
  • Consider stronger external encodings
    • Study what external encodings are safe
  • Reduce complexities
Thank you!

Any Questions?

I DON'T UNDERSTAND