Differential Fault Analysis on Grøstl–256

Wieland Fischer [1]   Christian A. Reuter [2]

[1]Infineon Technologies AG
Am Campeon 1–12, 85579 Neubiberg, Germany
wieland.fischer@infineon.com

[2]Justus-Liebig-Universität Gießen
35390 Gießen, Germany
christian.a.reuter@web.de

Fault Diagnosis and Tolerance in Cryptography 2012
09. September 2012

Contents

Differential Fault Analysis

- Differential Fault Analysis (DFA): Inducing faults in a cryptographic algorithm with a secret and using the erroneous output as side-channel.
- Assumption: The attacker having control over the hardware device and is able to run the process multiple of times.
- If he is able to, he realizes a certain **Fault model**.
- By using correct and faulty outputs he retrieves (partial or full) information about the secret.
- Fault attacks were done on RSA, DES, AES and many other symmetric and asymmetric crypto algorithms.
- We focus here on fault attacks on hash functions in context of HMAC.

HMAC: Definition

HMAC (Hash-based MAC) is a variant of Message Authentication Code (MAC) based on a cryptographic hash function.

Hash-based Message Authentication Code (HMAC): Definition

$$\mathrm{HMAC}_k(m) = h((k \oplus opad) \mid\mid h((k \oplus ipad) \mid\mid m))$$

where $opad :=$ 0x5C...5C and $ipad :=$ 0x36...36 are padding constants and $\mid\mid$ denotes the concatenation.

HMAC: Basic Attack Idea

- Choose hash function h.
- Unknown: Secret key k and maybe the input message m.
- Then we have
$$\mathrm{HMAC}_k(m) = h(m')$$
with $m' := (k \oplus opad) \,||\, h((k \oplus ipad) \,||\, m)$ being the input for the outer computation.
- The attacker gets the "message" m' if he is able to break h.
- Length and constants are known so one can cut off the last part to retrieve the secret key k.

Differential Fault Analysis: SHA

Secure Hash Algorithm (SHA):

- One-way functions with certain cryptographic properties.
- SHA–1, SHA–2 Family

Attacks:

- DFA on SHACAL–1 reveals the key (FDTC 2009).
- With this result the input value of SHA–1 could be determinated (FDTC 2011).

SHA–3 Contest

- 2012: the next standard SHA–3 will be announced.
- Final round: Five finalists, one of them is Grøstl.
- Grøstl imitates the main stuctures of AES.

Differential Fault Analysis: AES

- Advanced Encryption Standard (AES) is based on **states**. A state is a 4×4 matrix with byte-entries that represent elements/polynomials of $\mathbb{F}_{256} =: \mathbb{K}$.
- There are four round functions:
  - `AddRoundKey`: Adds the round key to the current state
  - `SubBytes`: Replaces all values in the current state by values from a fixed S-Box
  - `ShiftRows`: Shifts cyclic the rows of the current state
  - `MixColumns`: Multiplies the current state with a fixed matrix
- The last one of 10 **rounds** omits `MixColumns`.
- There are many popular DFAs on AES.
- Solely one fault is enough, to completely break the AES (WISTP 2011).

Grøstl–256: Definition

- Size of states: 8×8-bytes
- Let $\mathfrak{S} := \mathbb{K}^{8\times 8}$ be the set of 8×8-byte states.
- Internal block size and output length: $l := 512, n := 256$
- Compression function: $f(h, m) := P(h \oplus m) \oplus Q(m) \oplus h$
- P, Q are permutation functions and consist of 10 rounds R_i each
- One round: $R_i := \mathrm{MB} \circ \mathrm{SB} \circ \mathrm{Sub} \circ \mathrm{AC}$
    - AC: AddRoundConstant
    - Sub: S-Box layer (uses same S-Box as AES)
    - SB: ShiftBytes
    - MB: MixBytes
- Output transformation: $\Omega_n(x) := \mathrm{trunc}_n(P(x) \oplus x)$

Grøstl–256: Definition

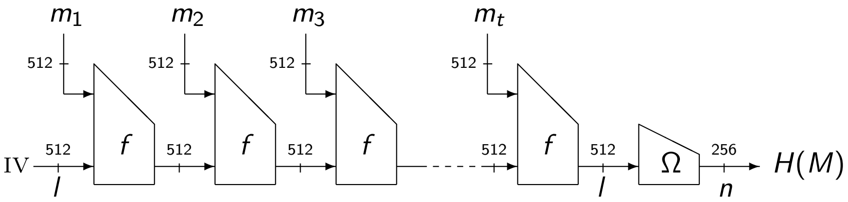

Figure 1: The Grøstl hash function.

Grøstl–256: Definition

Let $\mathfrak{S} := \mathbb{K}^{8\times 8}$ be the set of 8×8-byte states.

Compression Function

$$f\colon \mathfrak{S} \times \mathfrak{S} \longrightarrow \mathfrak{S},\ (h, m) = P(h \oplus m) \oplus Q(m) \oplus h$$

$$P, Q\colon \mathfrak{S} \longrightarrow \mathfrak{S},\ P = R_{P,9} \circ \ldots \circ R_{P,0},\ Q = R_{Q,9} \circ \ldots \circ R_{Q,0}$$

$$R_i\colon \mathfrak{S} \longrightarrow \mathfrak{S},\ R_i = \mathrm{MB} \circ \mathrm{SB} \circ \mathrm{Sub} \circ \mathrm{AC}_i$$

Output Transformation

$$\Omega_n\colon \mathfrak{S} \xrightarrow{P \oplus \mathrm{id}_{\mathfrak{S}}} \mathfrak{S} \xrightarrow{\mathrm{trunc}_n} \mathbb{K}^{8\times 4},\ x \longmapsto \mathrm{trunc}_n(P(x) \oplus x)$$

$$\mathrm{trunc}_n\colon \mathfrak{S} \longrightarrow \mathbb{K}^{8\times 4}, (s_{ij}) \longmapsto (s_{04}, s_{14}, \ldots, s_{74}, \ldots, s_{67}, s_{77})$$

Grøstl–256: Definition

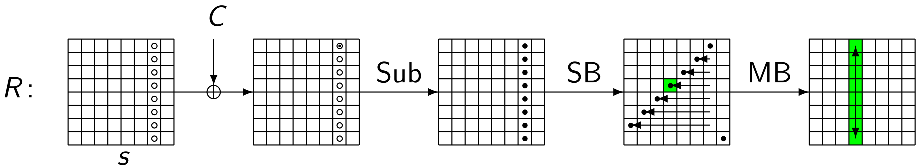

Figure 2: One round R of the Grøstl round function P.

Here s denotes the input state, C the constant added with AC (**AddRoundConstant**), Sub the **S-Box layer**, SB the **ShiftBytes** map and MB the map **MixBytes**.

Important Differences: DFA on AES and Grøstl

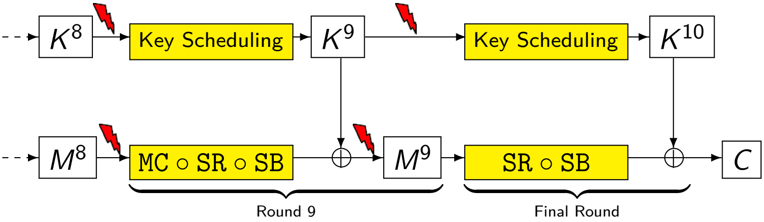

Figure 3: Some of the fault positions used in DFA on AES.

Known DFA on AES are not directly applicable to Grøstl.

- Only half of the informal information is being output.
- The output transformation Ω_n is a one-way function, so the output of the compression function is unknown.
- There is no key schedule, only plain constants.

Very Basic DFA on AES by Dusart (4. AES-C. 2003)

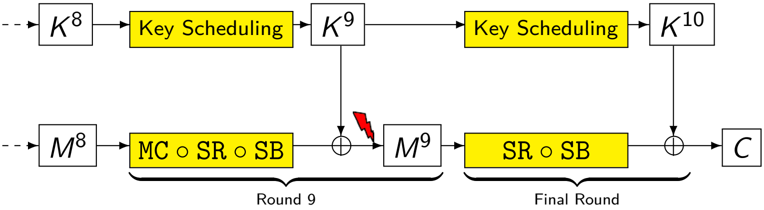

Figure 4: Position of induced fault in a very basic DFA on AES.

- A single one-bit fault is induced in only one byte.
- The correct output C and the faulty output D are known.
- Inversing `ShiftRows` in one byte of C and D.
- $C \oplus D$ is 0 in every entry except for the one in entry j.

$$\delta_j = \texttt{SubByte}(M_j^9) \oplus \texttt{SubByte}(M_j^9 \oplus \epsilon_j)$$

- Guess ϵ_j and obtain one byte of M^9.

Fault Model

- One-Bit Fault Model: Only one entry in a state is changed in exactly one bit.
- There are eight possibilities for a one-bit fault in a byte.
- The knowledge of the position of the fault is *not* essential for a successful attack.

The S-Box Difference

- The S-Box allows retrieving “hidden” information.
- Given the difference of correct and faulty S-Box values one can compute the original value x.

$$\boxed{\delta_i = S(x) + S(x + \epsilon_i), \quad \text{for} \quad i = 1, \dots, 4, \quad \delta_i, \epsilon_i, x \in \mathbb{K}}$$

- A maximum of four different faults ϵ_i are needed to compute one byte.

Attack in five Steps

Overview

$$x := f(h, m)$$

$$h(m) := \Omega_n(x)$$

Grøstl Attack

1. Step 1: Recovering half of the state $P(x)$ and x in $\Omega_n(x)$
2. Step 2: Recovering the full state x
3. Step 3: Pre-Computation
4. Step 4: Revealing $h \oplus m$
5. Step 5: Revealing m

Step 1: Recovering half of the state $P(x)$ and x in $\Omega_n(x)$

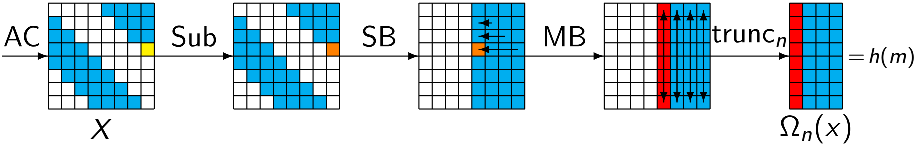

Figure 5: Processing of faults in the last round of P in Ω_n.

- $\Omega_n(x) \oplus \Omega_n'(x) = \text{trunc}_n(P(x) \oplus x \oplus P(x') \oplus x)$
- Faults are induced in X and process through SB, MB and trunc_n.
- Since SB, MB and trunc_n are bijective for the cyan shaded values, we can inverse them.
- Now we have the difference formula with a correct and faulty S-Box output. This reveals the absolute values of the cyan shaded entries of X and therefore half of $P(x)$ or x.

Step 2: Recovering the full state x

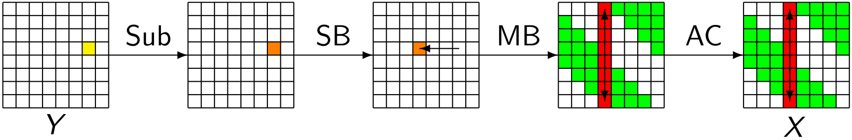

Figure 6: Processing of one fault in the penultimate round of P in Ω_n.

- The green shaded entries of correct X are known from Step 1.
- Yellow Induced Fault; Orange Faulty S-Box value; Red Specific linear distributed faulty values.
- The specific, constant multiplication of MB allows to compute the orange value if two values in one red column are known. We know four: the green ones.
- We again get a S-Box difference which we can solve like before. This provides the complete state Y and therefore x.

Step 3: Pre-Computation

- Now known: x. Still unknown: m and h of $f(h, m)$.
- Because of the one-way function Ω_n and its truncation there is no chance to compute the faulty output x' of f.
- Assume one-bit faults in the state Z of the last round of P or Q before Sub. They provide the differences δ_k after Sub.

$$\boxed{\delta = \text{Sub}(Z) \oplus \text{Sub}(Z \oplus \epsilon))}$$

$$\boxed{x' = x \oplus (\text{MB} \circ \text{SB})(\delta)}$$

- δ can only have $255 \cdot 8 \cdot 8 \approx 2^{14}$ different values, they are stored in a table with entries $\Omega_n(x')$.
- The table provides x' out of $\Omega_n(x')$.
- This is the most time expensive step.

Step 4: Revealing $h \oplus m$

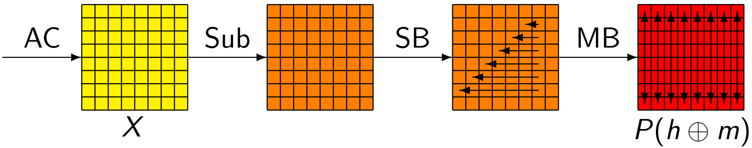

Figure 7: Processing of all faults in the last round of P in f.

- Insert faults in P within the computation of $f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h \quad (= x)$.
- x and x' are known, so once again a S-Box difference can be solved to obtain the values of the state X.
- $Q(m)$ and h cancel out, MB and SB are bijective:
  $\boxed{\delta = (\mathrm{MB} \circ \mathrm{SB})^{-1}(x \oplus x') = \mathrm{Sub}(X) \oplus \mathrm{Sub}(X \oplus \epsilon)}$
- Therefore, knowing X, the value $h \oplus m$ is retrieved by computing back the remaining nine rounds.

Step 5: Revealing m

- This step is very similar to the previous one.
- The faults are induced in Q instead of P.
- With the same methods as before this provides the value m and therefore also h.

- Steps 3–5 have to be done for every message block m.

Improvements to the Attack

- In **Step 2** the whole state was recovered – this is not necessary. It is enough to recover the half with four bytes per column and solve linear equations.
- **Step 1 and Step 2** can be done with a random byte fault model. It needs less faults and a weaker fault model imitating the attack of [Piret, Quisquater: CHES 2003].

Simulation

- The attack is of low complexity: A complete attack of the output transformation Ω_n and one compression step f takes **less than three minutes** on a usual PC.
- Most time is needed for the pre-computation: 2^{14} computations of Ω_n have to be done.
- Number of necessary errors depends on the way they are induced.
  - Induced when needed and with a known position: 2.19 faults per byte, this are 70, 140, 140, 140 faults for Step 1–4 respectively.
  - The improved method for Step 2 needs only 70 faults.
  - The random-byte fault model needs only 16 faults for Step 1 and Step 2, this are in average 296 faults overall.
  - Unknown position: 2.39 faults per byte needed, so we need in average 459 faults overall.

Thank you for your attention!

1eafa538 7de6610c 42a2598c d2996bf8 517d06f2 5a9962fa 0236f23e 27d8725d