Fault injection attacks on cryptographic devices and countermeasures – Part 1

Israel Koren

Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA

Outline

- Introduction - Side Channel Attacks
  - Passive and Active (Fault injection) attacks
  - Use RSA and AES as examples
- Countermeasures, e.g.,
  - Randomization
  - Duplication
  - Error detecting codes
- Interactions among different side channel attacks
  - Power analysis and fault injection
- Conclusions
Side-Channel Attacks

- Use information obtained from physical implementation rather than crypto-analysis of the cipher

- Also known as Passive attacks
  - Timing - encryption time may depend on key bits
  - Power - power profile may depend on bits of the key
  - Electro-magnetic radiation emanating from device

- Power analysis techniques have become the most “popular”
  - SPA - Simple power analysis
  - DPA - Differential power analysis
  - CPA - Correlation power analysis
  - HODPA - Higher order differential power analysis

- Sufficient to narrow the range of values to be attempted exhaustively

Active Attacks

- Deliberately injecting faults and observing the erroneous outputs
  - Proved to be a powerful technique allowing to retrieve the secret key with a very small number of experiments
  - Attacks exist against almost all known ciphers, e.g., AES, DES, RC4, RSA and ECC
Fault Injection Attacks

- Fault injection techniques
  - Vary the supply voltage - generate a spike
  - Vary the clock frequency - generate a glitch
  - Overheat the device
  - Expose to intense light - camera flash or precise laser beam

- In most cases - inexpensive equipment

Source: D. Naccache, 2004

**1st Fault Attack on RSA - Bellcore**

- Only decryption of the ciphertext $S$ is subject to attacks ($N=pq, d,e=$private/public key)

- Assume:
  \[ S^d \mod N = M^{de} \mod N = M \]

- 1. Attacker can flip a single bit in key $d$
- 2. $S$ and corresponding decrypted $M$ known to attacker

- Decryption device generates $\hat{M}$ satisfying

\[
\frac{\hat{M}}{M} = \frac{S^{2^j d_i}}{S^{2^j d_i}} \mod N
\]

- If $\hat{M} / M = S^2 \mod N$ then $d_i = 0$
- If $\hat{M} / M = 1 / S^2 \mod N$ then $d_i = 1$

- Similarly - flip a bit in $S$ or flip two or more bits

Boneh, DeMillo and Lipton (Bellcore), 1996
Simpler attack of CRT implementations of RSA

- Replace $d$ by $d_p = d \mod (p-1)$ and $d_q = d \mod (q-1)$
- Calculate $M_p = S^{d_p} \mod p$ and $M_q = S^{d_q} \mod q$

$$M = CRT(M_p,M_q) = (a \cdot M_p + b \cdot M_q) \mod N$$

where $a \equiv 1 \mod p; a \equiv 0 \mod q$ and $b \equiv 0 \mod p; a \equiv 1 \mod q$

Easier to attack using fault injection
- Inject a fault in the computation of either $M_p$ or $M_q$
- Resulting in, for example, $\hat{M}_p$
- The faulty decrypted message $\hat{M}$ satisfies $\hat{M} \equiv M \mod q$ $\hat{M} \not\equiv M \mod p$
- Thus, $q = \gcd(\hat{M} - M \mod N, N)$

Low Voltage Attack on RSA

- Non invasive attack
- Experiment - reduce the source voltage to an ARM µP
  - One out the 3 supply inputs - only LOADs were affected
  - For a certain voltage range single faults are more likely
- Data loads (data corruption) or Instruction fetch (Instr. "swap", e.g., BNE instead of BEQ)
- Data corruption - CRT implementation attack
- Instr. Swap - single bit flip in $d$
- Less than 5 min/attack

Barenghi et al, FDTC 2009
Fault Attacks against AES

- Many faults attacks have been proposed with some implemented in practice, e.g., SW implementation:
  - Cause branch instruction to fail using clock glitch
  - Execute only one or two rounds simplifying key extraction
- The shortest attack (against HW implementation)

- ~1046 possible values for the last round were needed
- If 2 faults were injected in previous step, only ≤16 possible values were left to examine
- Key observation: A byte fault propagates during MixCol to 4 bytes

Piret and Quisquater, 2003

Error propagation in AES (with byte parity)

- A single transient fault at byte #0

<table>
<thead>
<tr>
<th>Beginning of round 1</th>
<th>End of round 1</th>
<th>End of round 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 0 0 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
</tr>
</tbody>
</table>

Rounds from 3rd to 7th

<table>
<thead>
<tr>
<th>End of Round 7</th>
<th>End of Round 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>........</td>
<td>........</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>........</td>
<td>........</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>........</td>
<td>........</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

If two faults hit the same byte, but are separated by a distance of 8 rounds - will not be detected by a simple byte parity check
Z - The Error Propagation Matrix

- The 16×16 Z matrix describes the error propagation in a round
  - For \( n \) rounds \( E = Z^n E \)
- Error can be detected as long as \( E \) is not completely zeroed
  - \( Z \) is orthogonal, thus it never completely cancels an error

\[
Z = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

Piret Attack – AES-128

**Piret attack - Settings**

- The most famous attack to the AES retrieves the key of a 128 bit AES injecting single byte faults during a run of the algorithm, it does not matter if the computed primitive is the encryption or decryption one
- The attack relies on the fact that knowing a single round key from the AES key schedule is enough to reconstruct the full 128 bit key supplied by the user
- The attacker is required to inject a single byte fault in the AES state between the penultimate and last MixColumns operations
- We will now examine the effect of inserting a single byte fault in the correct position for the attack, in terms of the diffusion of the fault over the result
Piret Attack (2)

Fault injection point

Piret Attack (3)

No diffusion due to SubBytes and AddRoundKey
Piret Attack (4)

Shift effect due to Shiftrows

Piret Attack (5)

No diffusion due to SubBytes and AddRoundKey
Piret Attack (6)

Differential Fault Analysis - Technique

- The attacker needs to gather a couple of faulty-correct outputs pairs $F, \tilde{F}$ for the same inputs to the cipher primitive.

- **Key Idea**: Make an hypothesis on the value of four bytes of the last round key $K_r$ and roll back the cipher up to the last MixColumns.

- If the key hypothesis is correct, the difference between a faulty and a correct state before the last MixColumns should exactly be of one byte, while guessing a wrong value of four bytes of the last round key will result in a random difference.

- Since the hypotheses for a four bytes key value are only $2^{32}$ we can check which ones give us the correct difference through rolling back the cipher up to the fault injection point.

- We will now follow the cipher rollback process step by step, starting from the key guess:

Piret Attack (7)

Make an hypothesis on four bytes of $K_r$ (marked in red)
Piret Attack (8)

Invert the AddRoundKey for both $F$ and $\tilde{F}$ using the hp on $K_r$

```
+---+  +---+  +---+  +---+  +---+
|  0 4 8 12 |
|  1 5 9 13 |
|  2 6 10 14 |
|  3 7 11 15 |
+---+  +---+  +---+  +---+  +---+

K^{n+1}
```

```
+---+  +---+  +---+  +---+  +---+
|  0 4 8 12 |
|  1 5 9 13 |
|  2 6 10 14 |
|  3 7 11 15 |
+---+  +---+  +---+  +---+  +---+

K^n
```

Piret Attack (9)

Invert ShiftRows and SubBytes: obtain the values $C$ and $\tilde{C}$

```
+---+  +---+  +---+  +---+  +---+
|  0 4 8 12 |
|  1 5 9 13 |
|  2 6 10 14 |
|  3 7 11 15 |
+---+  +---+  +---+  +---+  +---+

K^{n+1}
```

```
+---+  +---+  +---+  +---+  +---+
|  0 4 8 12 |
|  1 5 9 13 |
|  2 6 10 14 |
|  3 7 11 15 |
+---+  +---+  +---+  +---+  +---+

K^n
```
Piret Attack (10)

Compute the difference $\varepsilon = C \oplus \tilde{C}$

Piret Attack (11)

$C \oplus \tilde{C} = B \oplus K^9 \oplus \tilde{B} \oplus K^9 = B \oplus \tilde{B}$: we know the difference between the states before ADDROUNDKEY, without need to know the key!
Piret Attack (12)

Since the MixColumns is linear w.r.t. \( \oplus \) computing \( \text{InMixColumns}(B \oplus \widetilde{B}) \) yields \( A \oplus \widetilde{A} \). We can check if \( A \oplus \widetilde{A} \)!

Piret Attack (13)

Differential Fault Analysis - Feasibility and complexity

- After checking the \( 2^{32} \) possible hypotheses with a single correct/faulty ciphertext pair, on average 1020 solutions remain: this is due to the fact that there are \( 255 \times 4 \) possible single byte differences \( A \oplus \widetilde{A} \) matching the fault injection model.
- With a single correct/faulty ciphertext pair per column of the state it is thus possible to cut down the keyspace to \( (255 \times 4)^4 \approx 2^{40} \).
- Iterating the procedure with another correct/faulty key pair effectively yields only one correct key candidate for a quarter of the last round key \( K_r \).
- On average, the procedure takes less than 20 seconds on a common desktop, thus enabling a full recovery of the last AES round key in around one minute of computation.
**Piret Attack (14)**

- We need the last + penultimate round keys to break AES-192 and AES-256 due to the nature of the AES key schedule.
- Due to the presence of an extra MixColumns operation, it is not possible to reuse the same technique to attack AES-192/256: the key hypothesis would be 128 bit wide!

**Piret Attack (15)**

Extended attack to AES - Working around Piret’s limitations

- It is possible however to work around the double MixColumns diffusion effect, through exploiting its linearity w.r.t the XOR operation twice.
- The key idea is to make an hypothesis on 32 bits of an intermediate state of the cipher in order to overcome the empathe, and bypassing both the AddRoundKey operations employing the XOR-difference method.
- We will assume in the following slides that the attacker has already peeled off the last round of the cipher retrieving the key via Piret’s method and follow the technique.
# Piret Attack (16)

**Summing up**

- It is possible to recover the round key of AES for any of the rounds, regardless of the fact that the last one is missing the `MixColumns` operation or not.
- This allows an attacker to roll back any number of AES rounds, rendering useless the addition of rounds as a countermeasure.
- One of the issues which allows this attack is the partial diffusion performed by the `MixColumns` operation (allows 32 bit hypotheses to be made).
- The attacks are feasible in a couple of minutes at worst on a common desktop, with a couple of kB of correct and faulty outputs.