Fault injection attacks on cryptographic devices and countermeasures

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

  $$\frac{\hat{M}}{M} = S^{2^kd} \mod N$$

- 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 \( M_q = S^{d_q} \mod q \)

\[
M = \text{CRT}(M_p, M_q) = (a \cdot M_p + b \cdot M_q) \mod N \quad \text{where} \quad a \equiv 1 \mod p; a \equiv 0 \mod q \quad \text{and} \quad b \equiv 0 \mod p; a \equiv 1 \mod q
\]

- 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
    \[
    \hat{M} \equiv M \mod q \quad \hat{M} \not\equiv M \mod p
    \]
  - 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 \( \mu P \)
  - 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>

<table>
<thead>
<tr>
<th>Rounds from 3rd to 7th</th>
<th>End of Round 7</th>
<th>End of Round 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>........</td>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>........</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>........</td>
<td>0 0 1 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>........</td>
<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
The Error Propagation Matrix

- The $16 \times 16$ $Z$ 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.

$$Z = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Countermeasures - Examples

- Must first detect injected fault, then prevent attacker from observing erroneous output.
  - Block the output (e.g., generate all zeroes output), or
  - Produce a random output misleading the attacker, or/and
  - Erase the secret key after a certain number of attacks.

- **1. Active protection** - use sensors to detect variations in voltage, frequency, light etc.
- **2. Duplicate encryption** (decryption) process (hardware or time redundancy) and compare results - injected faults transient and will manifest differently.
  - Spatial duplication - redundant encryption unit or use decryption unit & compare to original plaintext
  - Temporal duplication - reuse hardware or re-execute software
  - Above techniques may incur high hardware and/or time penalty.
3. Error Detecting Codes (EDCs)

- Error-detection codes may require less overhead but possibly have a lower coverage
  - Code generator, prediction circuits and comparator(s)

- Common Codes (separable):
  1. Parity codes
  2. Residue codes
  3. Error correcting codes (e.g., Hamming code)

- Specialized codes
  - AES - linear code for the linear part and inverse calculation for the non-linear part (S-Box)
  - A nonlinear code allowing a fault coverage vs. hardware overhead tradeoff

- Performance and area overheads
  - Performance 30% - 100%
  - Area 15% - 170%

Error Detecting Codes (EDCs)

- First generate check bits
- For each operation within encryption predict check bits
- Periodically compare predicted check bits to generated ones
- Predicting check bits for each operation - most complex step
  - Should be compared to duplication
**Example: Parity prediction for AES**

- Byte-level parity is natural - a total of 16 parity bits
- ShiftRows: rotating the parity bits
- AddRoundKey: add parity bits of state to those of key
- SubBytes: Expand Sbox to 256×9 - add output parity bit; to propagate incoming errors (rather than having to check) expand to 512×9 - put incorrect parity bit for inputs with incorrect parity
- MixColumns: Expressions below where \( s_{i,j}^{(7)} \) is msb of state byte \( i,j \)

\[
\begin{align*}
p_{0,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s_{0,j}^{(7)} \oplus s_{1,j}^{(7)} \\
p_{1,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{2,j} \oplus s_{1,j}^{(7)} \oplus s_{2,j}^{(7)} \\
p_{2,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{2,j} \oplus s_{2,j}^{(7)} \oplus s_{3,j}^{(7)} \\
p_{3,j} &= p_{1,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s_{3,j}^{(7)} \oplus s_{0,j}^{(7)}
\end{align*}
\]

**AES – Scheduling of Checks**

- Comparing predicted to generated parity bits
  - After each operation
  - After each round
  - At end of encryption - smallest hardware & time overheads

\( \triangleright \) should not mask error indication (Error propagation matrix)

Transformation level

Round level

Encryption level
Error Coverage – Parity bits

100 % coverage of single faults

Error Detection Overheads

<table>
<thead>
<tr>
<th>Design I (S-Box)</th>
<th>Area ($\mu m^2$)</th>
<th>Latency (ns)</th>
<th>AT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>233,095</td>
<td>8.88</td>
<td>2,069,883</td>
</tr>
<tr>
<td>w/Error Detection</td>
<td>271,245</td>
<td>11.06</td>
<td>2,999,969</td>
</tr>
<tr>
<td>Overhead</td>
<td>+16.36%</td>
<td>+24.55%</td>
<td>+44.93%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design II (S-Box)</th>
<th>Area ($\mu m^2$)</th>
<th>Latency (ns)</th>
<th>AT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>226,099</td>
<td>9.32</td>
<td>2,107,242</td>
</tr>
<tr>
<td>w/Error Detection</td>
<td>259,631</td>
<td>12.27</td>
<td>3,185,672</td>
</tr>
<tr>
<td>Overhead</td>
<td>+14.83%</td>
<td>+31.62%</td>
<td>+51.18%</td>
</tr>
</tbody>
</table>

- Latency overhead is mainly due to the code comparator
  - Can be reduced by moving comparator out of the critical path
- Common design improvements can be followed
  - E.g., pipelining to hide latency
Reducing the Performance Overhead

- Apply complete temporal redundancy to AES but
- Drastically reduce the performance penalty
- Double-Data-Rate (DDR) technique
- Perform the two encryptions rounds (that would be compared) during the same clock cycle
  - Use rising and falling clock edge
- Lower maximum clock frequency
  - No penalty if embedded in a slow system
- Detection relies on the two computations not affected by the same fault which can be a multi-cycle one
  - Authors claim: small percentage (~6%) of undetected faults; goes up to 39% for 6-cycle faults

Maistri and Leveugle, 2008

EDCs for other Block Ciphers

- Other ciphers use different basic operations, e.g.,
  - Bit-oriented operations (DES)
  - Modular arithmetic with unusual modulus (IDEA)
- Determine the “best” EDC for a given cipher, for example:

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Suggested Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>Parity</td>
</tr>
<tr>
<td>IDEA</td>
<td>Residue, but expensive</td>
</tr>
<tr>
<td>MARS</td>
<td>Residue, but expensive</td>
</tr>
<tr>
<td>RC5</td>
<td>Parity or residue</td>
</tr>
<tr>
<td>RC6</td>
<td>Residue</td>
</tr>
<tr>
<td>Rijndael (AES)</td>
<td>Parity, per byte</td>
</tr>
<tr>
<td>Twofish</td>
<td>Parity, per byte</td>
</tr>
</tbody>
</table>
Detection Coverage - RC5 (Residue)

Protecting RSA

- Randomized multiplicative masking - use random integers $r_1$, $r_2$
  
  $M_p^* = S^{d_p} \mod (p \cdot r_1)$  
  $M_q^* = S^{d_q} \mod (q \cdot r_2)$

  $M_1 = S^{d_p \mod \phi(r_1)} \mod r_1$  
  $M_2 = S^{d_q \mod \phi(r_2)} \mod r_2$

- If $M_1 = M_p^* \mod r_1$ and $M_2 = M_q^* \mod r_2$
  
  output $M = CRT(M_p^*, M_q^*)$  
  else Error detected

Shamir, 1999

- A fault injected during the CRT combination not detected

- Another option: Use Residue codes
  
  - Fits modular arithmetic
  
  - The result residue check bits of any operation can be easily obtained from the check bits of the operands
Residue code for RSA: Overheads

<table>
<thead>
<tr>
<th>Key Length (bits)</th>
<th>Global Overheads</th>
<th>Area Memory Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>768</td>
<td>+17.8%</td>
<td>+3.7%</td>
</tr>
<tr>
<td>1024</td>
<td>+17.8%</td>
<td>+2.9%</td>
</tr>
<tr>
<td>1536</td>
<td>+17.8%</td>
<td>+2.0%</td>
</tr>
<tr>
<td>2048</td>
<td>+17.8%</td>
<td>+1.5%</td>
</tr>
</tbody>
</table>

The residue modulus is $2^{16} - 1$

The aliasing probability (i.e., error not detected) is $\approx 2^{-16}$

Fault Detection not always sufficient

- **Example - RSA**
- **Can be protected against fault injection using residue check or encrypting $M$ & comparing to $S$**
- **Is vulnerable to power analysis - more power consumed if $d_i = 1$**
- ♦ **Modified to use same power + fault detection**
- ♦ **Still vulnerable to fault injection**
- ♦ **Inject fault in calculation of $b$ - if correct $M$, one bit of $d$ is deduced**
- ♦ **Faulty result not needed!**
- ♦ **"Shut down" if several faults detected!**

```
DecryptionAlgorithmException.1\((S, N, (d_{n-1}, d_{n-2}, \ldots, d_0))\)
begin
    a = S
    for i from n - 2 to 0 do
        a = a^2 mod N
        if d_i = 1 then a = S \cdot a \mod N
    end
    M = a
end
```

```
DecryptionAlgorithmException.2\((S, N, (d_{n-1}, d_{n-2}, \ldots, d_0))\)
begin
    a = S
    for i from n - 2 to 0 do
        a = a^2 mod N
        b = S \cdot a \mod N
        if d_i = 1 then a = b else a = a
    end
    if (no error has been detected) then M = a
end
```

Yen and Joye, 2000

23

24
Montgomery-step Algorithm

- Intermediate values of a & b are used and an injected fault will be detected
- Provides another way to detect faults: a & b must be of form (M,SM)
- Checking this relation detects most faults except - bits of d or number of loop iterations - these must be checked separately (e.g., EDC)

Decryption Algorithm 3(S, N, (d_{n-1}, d_{n-2}, \ldots, d_0))
begin
    a = 1
    b = S
    for i from n - 1 to 0 do
        if d_i = 0 then
            a = a^2 mod N
            b = a \cdot b mod N
        end
        if d_i = 1 then
            a = a \cdot b mod N
            b = b^2 mod N
        end
    end
    if (no error has been detected) then M = a
end

Joye and Yen, 2002

Safe-error Resistant Algorithm

- Avoid decision tests
- Check errors in d and loop counter
- Error detected if
  \[ a_i \neq S \cdot a_0 \mod N \]

Algorithm 4(S, N, d_{n-1}, d_{n-2}, \ldots, d_0)
begin
    a_0 = S
    a_1 = S^2 \mod N
    for i from n-2 to 1 do
        a_{d_i} = a_{d_i} \cdot a_i \mod N
        a_{d_i} = a_{d_i}^2 \mod N
        a_i = a_i \cdot a_0 \mod N
        a_0 = a_0^2 \mod N
        if (loop counter and d not modified) then
            return (a_0, a_1)
        end
    end
end

Giraud, 2005
**AES - Successful attack even if faults detected**

- Provide all-zero input to AES encryption
- An initial round key is added (XOR): state=key
- Before SubBytes inject a stuck-at-0 fault into bit j
  - If result is correct then bit j of key is 0
- Even duplicating the encryption will not help - it does not matter whether the fault was detected or not
  - Unless the number of allowed faults is limited
- Attack is complicated - exact timing and precise location of fault and fault type
  - If strict timing and location are not practical - repeating the experiment many times will allow extracting the secret key
- Attack can be done if a byte (or several bytes) are reset to 0
  - If key byte j is reset to 0, perform 256 encryptions with byte j of message assuming values 0 to 255 - the one that matches the faulty ciphertext reveals byte j of key (a.k.a Collision Fault attack)

*Blomer and Seifert, 2003*

**Combining Passive and Active Attacks**

- Many current cryptographic devices include separate countermeasures against power attacks and fault injection attacks
- Two new questions/challenges
  - Can a countermeasure against one type of attacks make the other one simpler to execute?
  - What happens if the attacker uses a combination of passive and active attacks?
Can the presence of error checking circuitry make a power attack simpler?

- Correlation Power Analysis (CPA)
- Based on linear relationship between power and Hamming weight of data processed
- AES implementation with no error check circuit
- Correct key distinguishable after 160 traces

---

**Differential Power Analysis**

1. Collect many ciphertexts and the power traces for the last round
2. Guess a byte of the final round key
3. Calculate the target byte based on the guess
4. Select one bit of the target byte B
5. Divide the power traces into 2 sets: those for B=1 and those for B=0
6. Calculate the averages of the 2 sets and the difference between the averages
7. If the average depends on B there will be a spike in the data indicating correlation
8. If the guess (of the key byte) is correct the power should depend on the value of B
Hypothesis: The 8 key bits are 00001111

Hypothesis: The 8 key bits are 01010101
Correlation Power analysis

* Construct a power model to estimate the power for every value of one byte of the last round key

* Calculate the correlation between the estimated power and the power traces

* The highest correlation indicates the correct key byte

Power Attacks in the presence of error checking

- AES with a parity bit per byte
- Correct key distinguishable after 130 traces
- For residue mod 3 code - correct key distinguishable after 100 traces
Fault Injections that make DPA feasible

- Circuit techniques to protect HW implementations against DPA have been developed
  - Specially designed balanced gates for which the power consumption is equal for all data
- Faults injected in the “balancing” part of the circuit will imbalance it but will not cause a logical error
  - Can not be detected by any redundancy scheme
  - If 4 out of 137 gates were made imbalanced (through fault injection) the protected circuit was as vulnerable to DPA as an unprotected circuit
- A possible countermeasure is adding differential current comparators that would detect the imbalances

Kulikowski, Karpovsky and Taubin, 2006

Protecting RSA against DPA and Faults

- The “fault resistant” algorithm is multiplicatively blinded by a random number r making it DPA resistant as well
  - The increase in execution time vs. Algorithm_4 is about 45%
- A fault is not detected if injected during the computation of
  - $a_2 = a_2^2 \mod N$
- Modified algorithm developed in 2008 by Kim & Quisquater

Algorithm_5 $(S, N, d_{n-1}, \ldots, d_0)$
Select a random number $r$
$a_0 = r$; $a_1 = rS$; $a_2 = r^{-1}$
for $i$ from $n-1$ to 1 do
  $a_{2i} = a_{2i}^2 a_{d_i} \mod N$
  $a_{d_i} = a_{d_i}^2 \mod N$
  $a_2 = a_2^2 \mod N$
$a_1 = a_1 a_0 \mod N$
$a_0 = a_0^2 \mod N$
$a_2 = a_2^2 \mod N$
return $(a_2 a_0, a_2 a_1)$

Fumaroli and Vigilant, 2006
Protecting AES against DPA and Faults

To protect an fault-resistant AES implementation against DPA one can mask all 16 data bytes with a random number $r_1$ and all key bytes with $r_2$

Injecting faults in the first XOR operation using the Collision Fault Attack allowed recovering the key with 112 fault injections (Amiel, Clavier and Tunstal, 2006)

To protect against the above use 16 different random masks instead of one
- Can not use S-Boxes (would need 16 tables for each value of $r$)
- This implementation increased the area by ~40% and the latency by ~250%

AES DPA and DFA – Recent results

Recently (2010) a modified collision fault attack was developed requiring ~1568 faults to be injected

One countermeasure that has been suggested - duplicate the AES rounds that are exposed to the attack
- The duplicated rounds should be performed with two different masks
- Bytes should be processed in a random order
- The first 3 and last 3 rounds duplicated leading to
- Latency overhead of about 400% vs. the previous 250%
Conclusions

- The need to protect cryptographic devices against passive and active side channel attacks is well established
  - A strong cipher is insufficient
  - Hardware and/or software aids must be included in the design to counteract side channel attacks
  - Current techniques incur a high overhead

- Interactions among different side channel attacks must be further investigated
  - Separately protecting against individual side channel attacks is insufficient
  - The currently known techniques to counter both passive and active attacks have a high overhead