Two Operands of Multipliers in Side-Channel Attack

Takeshi Sugawara, Daisuke Suzuki, and Minoru Saeki

Mitsubishi Electric Corp.
Overview

• The recent single-shot internal-collision attack is studied focusing on operand order of multipliers
  – The operand order divides success/failure of the attack
  – The causes of the difference are analyzed
    • Integer multipliers
    • Long-integers multiplication algorithm
  – Designing operand order can be a cost-effective way to suppress side-channel leakage
Background
A classical SCA on RSA*

- Distinguish square and multiply to decode the secret key $d_i$

```
for(i=0; i<N; ++i){
    y = y \times y;
    if($d_i==1$){
        y = y \times M;
    }
}return y;
```

```
for(i=0; i<N; ++i){
    y[1] = y[1] \times y[1];
    y[d_i] = y[1] \times M;
}return y[1];
```

- Example
  - In typical compiler-generated code, “if” branch is faster than “for” branch, and thus they are distinguishable

- Data-dependent branch should be avoided

Internal collision attack by Witteman et al.∗

• Attacking the multiply-always method
• Idea
  – Consecutive multiply & squaring has exponent-dependent collision
  – The collision is found by looking at the correlation matrix

• The matrix requires multiple RSA calls with the same exponent
  – DPA countermeasures are effective

Internal collision attack by Hanley et al.*

- Internal collision is detected using a single trace only

```
y \times y \quad y \times M \quad y \times y \quad y \times M \quad y \times y \quad y \times M
```

- **Time**

Consecutive multiply & squaring are compared at time domain

- **Pro:** defeats DPA countermeasures
- **Con:** available side-channel info. is limited
  - Feasibility heavily depends on SNR
  - Known results
    - 99% success in software
    - Unsuccessful for FPGA impl. with multiply-always method
      - Low SNR in FPGA

Our first attempt

• Improving the attack using the local-EM measurement
  – Very good SNR
  – Even transistor-level leak is measurable*

• Preliminary experiments
  – Local-EM traces are measured from multiple designs and analyzed with the method by Hanley et al.
  – Success rate greatly differ between the designs
  • We noticed that the operand order determines the difference

Operands of multipliers
Two levels of multiplication

- 1st level: word-size integer multiplier
- 2nd level: long-integer multiplication
- 2x2 possible operand orders
Integer multipliers

- Two operands are mixed at the PPG stage
  - The simplest PPG is symmetric

Simple 3-bit PPG (symmetric)

Input:
\{a_2, a_1, a_0\} and \{b_2, b_1, b_0\}

Output:
\{a_2b_2, a_2b_1, a_2b_0, \\
a_1b_2, a_1b_1, a_1b_0, \\
a_0b_1, a_0b_1, a_0b_0\}

Sophisticated PPG: Booth recoding

- Commonly used in commercial logic synthesizers
- Multiplier is recoded with \{-2, -1, 0, 1, 2\} instead of \{0, 1\}
  - The number of partial products is reduced
  - Similar idea to the non-adjacent form (NAF)
- The circuit is asymmetric between operands

A circuit for generating one partial product with radix-4 Booth recoding
Simulating power consumption

- A 32-bit multiplier with the radix-4 Booth recoding is made
- Post-synthesis logic simulation is conducted while counting the number signal-changing events (i.e., toggles)
- Two types of test vectors
  - $a_i \times \text{const}$
  - $\text{const} \times a_i$
- The multiplier uses more power than the multiplicand
Long integer multiplication (LIM)

- Varieties
  - Operand scanning
  - Integration with modular reduction (e.g., Montgomery multiplication)

\[
\begin{align*}
 a_2 & \times b_2 \\
 a_1 & \times b_1 \\
 a_0 & \times b_0
\end{align*}
\]

Partial-product generation and carry propagation
Common circuit architecture for LIM

- Use of multiply-and-accumulate (MAC) unit
Common circuit architecture for LIM

- Use of multiply-and-accumulate (MAC) unit
Operand scanning in LIM

- Reading from memory is a common performance bottleneck
  - An operand is cached if possible
  - regB is updated less frequently thus smaller leakage is expected
Experiment
Experiment

- 1024-bit Montgomery multiplier
  - CIOS operand scanning
  - MAC-based architecture with a 64-bit integer multiplier
  - Two operands to the 64-bit integer multiplier are swappable by an external switch
- The circuit is loaded to FPGA and its local-EM traces are measured
Measured cases

- Test vectors to the Montgomery multiplication is designed to emulate the internal-collision attack of the multiply-always method
- All the possible 2x2 cases are examined
**Result**

- The attack results are shown as histograms of correlation coefficients
  - Black bars: correlation with collision, white bars: correlation without collision

<table>
<thead>
<tr>
<th>Integer Mult.</th>
<th>LIM: Collision at Inner Loop</th>
<th>LIM: Collision at Outer Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Collision at Multiplicand</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image1.png" alt="Histogram" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image2.png" alt="Histogram" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Collision at Multiplier</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image3.png" alt="Histogram" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image4.png" alt="Histogram" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Countermeasure

- The operand order divided success/failure of the attack
  - The best: 99.5% success
  - The worst: 52.7% success

- Designing operand order can be a cost-effective way to suppress side-channel leakage
  -Operand order can be changed for almost no cost
    - The good configuration can be predicted with simulation
    - Reconfiguring operand order after fabrication is also possible at low cost
    - Easy integration with other countermeasures
Conclusion

• Conclusion
  – Single-shot internal-collision is effective to hardware implementations
    • especially when combined with local EM measurement
  – Small difference like operand order divides success/failure
    • Understanding of low-layer circuits is useful for efficient countermeasures

• Further study
  – How much success rate of the exponent recovery is acceptable?
    • Say, for 80-bit security
  – Randomized operand scanning
  – Collision between input and output?
    • We have never seen exploitable (pure) input-to-output collision so far
    • Sometimes, an input-to-output collision becomes input-to-input collision
      – E.g., an output is immediately read back for the final subtraction