TOWARDS SECURE CRYPTOGRAPHIC SOFTWARE IMPLEMENTATION AGAINST SIDE-CHANNEL POWER ANALYSIS ATTACKS

Northeastern University Energy-Efficient and Secure Systems Lab
Outline

- Background and Preliminaries
- Dependence extraction and shuffling space exploration
- Implementation and side-channel analysis results
Outline

• Background and Preliminaries
  • Side-channel analysis and countermeasures
  • Keccak and its side-channel attacks

• Dependence extraction and shuffling space exploration

• Implementation and side-channel analysis results
Side-Channel Analysis
Countermeasures

- Countermeasures
  - Masking (secret sharing)
  - Shuffling (permutation)
  - Power balance circuits (for ASICs and FPGAs)

- Shuffling
  - Lower overhead than masking
  - Require no thorough understanding of the target algorithm
Shuffling

- Randomize the order of execution every time such that the leakages spread to multiple points

```
Algorithm 1 Shuffling execution

Input: $C = \{C_0, C_1, \ldots, C_{n-1}\}$, and a random order array, Order

1: for $i = 0 \rightarrow n - 1$ do
2:   switch Order[$i$] do
3:     case 0
4:       Execute $C_0$
5:     case 1
6:       Execute $C_1$
7:     \ldots
8:     case $n - 1$
9:       Execute $C_{n-1}$
10: end for
```
Side-channel attacks on Keccak

\( \theta_1 \) compress five bits in one column to one bit; 
\( \theta_2 \) combines two \( \theta_1 \) bits and one input bit.

Each \( \theta_1 \) involves one key bit and this has been used for attacks.

Outline

- Background and Preliminaries
- Dependence extraction and shuffling space exploration
- Implementation and side-channel analysis results
Definition of Dependence

- Data dependence
  - Read after write (RAW)
  - Write after read (WAR)
  - Write after write (WAW)

- Shuffling should preserve inherent data dependence
Extraction of Dependence

Algorithm 1 Dependence extraction

Input: The source code of the crypto algorithm
Output: Lower triangular dependence matrix $M$ for the algorithm

1: $N \leftarrow \text{number of statements } |A|$
2: Generate an $N \times N$ matrix $M$, initialized at zero
3: for $i = 1 \rightarrow N - 1$ do
4: for $j = 0 \rightarrow i - 1$ do
5: if $V[j][0] \in V[i]$ then \( A[i] \Rightarrow A[j], \text{ WAW and RAW} \)
6: \( M[i][j] \leftarrow 1 \)
7: end if
8: if $V[i][0] \in V[j]$ then \( A[i] \Rightarrow A[j], \text{ WAW and WAR} \)
9: \( M[i][j] \leftarrow 1 \)
10: end if
11: end for
12: end for
Extraction of Dependence
Shuffling Space Exploration

- Random insertion of one statement
  - For one statement leaks information,
- Separate the statements into several levels
  - Statements in the same level do not depend on each other
  - Statements in higher level depend on statement in lower level
Shuffling Space Exploration

Algorithm 2 Execution level extraction

\textbf{Input:} The $N \times N$ dependence matrix $M$
\textbf{Output:} Execution level array $L[N]

1: Initialize an zero array $L[N]$
2: \textbf{for} $i = 1 \rightarrow N - 1$ \textbf{do}
3: \hspace{1em} $level = 0$, $newlevel = 0$
4: \hspace{1em} \textbf{for} $j = 0 \rightarrow i - 1$ \textbf{do}
5: \hspace{2em} \textbf{if} $M[i][j] = 1$ \textbf{then}
6: \hspace{3em} $newlevel = L[j] + 1$
7: \hspace{2em} \textbf{if} $newlevel > level$ \textbf{then}
8: \hspace{3em} \hspace{1em} $level = newlevel$
9: \hspace{2em} \textbf{end if}
10: \hspace{1em} \textbf{end if}
11: \hspace{1em} \textbf{end for}
12: \hspace{1em} $L[i] = level$
13: \textbf{end for}
Outline

• Background and Preliminaries
• Dependence extraction and shuffling space exploration
• Implementation and side-channel analysis results
Correlation Power Analysis Result

(a) Power leakage of the unprotected Keccak

(b) Power leakage of Keccak protected with shuffling
# Resource Overhead

<table>
<thead>
<tr>
<th>Implementations</th>
<th>File size (Byte)</th>
<th>Clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original [1]</td>
<td>31040</td>
<td>1670</td>
</tr>
<tr>
<td>Shuffling</td>
<td>40128</td>
<td>2580</td>
</tr>
</tbody>
</table>

Conclusion

• Conclusion
  • Our method can efficiently explore the space for shuffling
  • Our method has low resource overhead
  • Our method can help to fight against side-channel analysis
Thanks!