Symbolic Approach for Side-Channel Resistance Analysis of Masked Assembly Codes
Workshop PROOFS

Inès Ben El Ouahma  Quentin Meunier  Karine Heydemann  Emmanuelle Encrenaz

Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France

September 29th, 2017, Taipei, Taiwan
1. Introduction / Motivation

2. Symbolic Method

3. Experiments

4. Conclusion
Side-Channel Attacks

- **Execution time**
- **EM emission**
- **Power Consumption**

Side channels → Measurements → Statistical analysis for key recovery
The Masking Countermeasure

Aim: observation of $d$ intermediate computations cannot reveal the secret $x \implies d$-th order masking

- Splits a secret $x$ in $d+1$ shares using random uniform variables called masks
- Operation-dependent, i.e boolean masking: $x \oplus m$
- At software level, usually added in the source code (easy to identify secret variables)

Problems

- Need to ensure that a masked program is leakage free in practice
- Compilation flow and optimizations (reordering, removal...) may affect masking effectiveness
Masked Programs Security: Existing Formal Verifications

- **[Bayrak, CHES13]** SAT verification of sensitivity: an operation on a secret must involve a random variable which is not a *don’t care* variable (i.e., it affects the result)
  - ✓ Low level: LLVM programs
  - × Security property not sufficient

- **[Eldib, TACAS14]** SMT verification of perfect masking, i.e., statistical independency of intermediate computations from secrets
  - ✓ Strong security property
  - × C level & Bit-blasted programs (could be applied to low level)
  - × Lack of scalability (combinatorial blow-up of the enumeration)

- **[Barthe, Eurocrypt15]** \( t \)-non-interference: joint probability distribution of any \( t \) intermediate expressions is independent from secrets
  - ✓ Strong security property
  - ✓ Good scalability
  - × Cannot conclude for some cases
Our Goal

To verify side channel resistance:

- Of 1st order masked programs
- At assembly level
- In the value-based model: instruction result leaks
- Considering that: leakage-free instruction $\iff$ result is statistically independent from secrets
- With a symbolic approach that infers the distribution type of instruction expressions
Plan

1. Introduction / Motivation
2. Symbolic Method
3. Experiments
4. Conclusion
Verification Scheme

\[
\begin{align*}
&\# \ r0 \leftarrow k; \ r1 \leftarrow m1; \ r2 \leftarrow m2; \ r3 \leftarrow m3 \\
1 &\text{ eor } r4, \ r0, \ r1 \ # k \oplus m1 \\
2 &\text{ eor } r5, \ r0, \ r2 \ # k \oplus m2 \\
3 &\text{ and } r5, \ r5, \ r3 \ # (k \oplus m2) \& m3 \\
4 &\text{ and } r5, \ r5, \ r4 \ # (k \oplus m1) \& ((k \oplus m2) \& m3)
\end{align*}
\]

Is the root distribution statistically independent from k?

- Inputs tagged with a distribution type
- Bottom-up combination of distribution types using defined inference rules

Data dependency graph of the last instruction
Symbolic Approach

4 distribution types for variables and expressions:

- Random Uniform Distribution (**RUD**)
- Unknown Distribution (**UKD**)
- Constant (**CST**)
- (Statistically) Independent from Secrets Distribution (**ISD**): not necessarily uniform but identical for all values of the secrets.

**k**: secret  
**m₁, m₂**: masks  
**e** = (k ⊕ m₁) & m₂  
**e’** = (k ⊕ m₁) & m₁

<table>
<thead>
<tr>
<th>k</th>
<th>m₁</th>
<th>m₂</th>
<th>e</th>
<th>e’</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
P(e=0) = \frac{3}{4} \quad P(e=1) = \frac{1}{4}
\]

\[
P(e’=0) = \frac{1}{2} \quad P(e’=1) = \frac{1}{2}
\]
Independence Notions

Which distribution types assert that an expression is statistically independent from secrets?

Dependence between expression $e$ and variable $v$:

- **structural** $\Rightarrow v$ appears in $e$
- **statistical** $\Rightarrow$ the distribution of the result of $e$ depends on $v$

$
\Rightarrow$ Need to keep track of structural dependencies: $(k \oplus m) \& m$

Safe types:

- $e \sim \text{RUD}$
- $e \sim \text{ISD}$
- $e \sim \text{UKD}$ with no structural dependency on any secret

Unsafe type:

- $e \sim \text{UKD}\{\text{dep}\}$ with structural dependency on some secret variable: $\text{dep} \cap S \neq \emptyset$
Dominant Masks

Aim: to find a mask that randomizes the whole expression

**Dom Rule**
- expression $e = e' \oplus m$ or $e = e' + m \mod 2^n$
- $m \sim \text{RUD}\{m\}$
- $m \notin \text{dep}(e')$

$\implies e \sim \text{RUD}$ and $m$ is a dominant mask of $e$.

2 sets of dominant masks:
- $\text{dom}_\oplus(e)$ the set of xor dominant masks of $e$
- $\text{dom}_+ (e)$ the set of additive dominant masks of $e$

Examples:
- $\text{dom}_\oplus((k + m1) \oplus (k \oplus m1 \oplus m2)) = m2$
- $\text{dom}_+( (k + m1) \oplus 0) = \text{dom}_+(k + m1) = m1$
Other Inference Rules

By distribution types:

- Set of rules for $\oplus$, $+$ mod $2^n$
- Set of rules for AND and OR

Disjoint rule for binary operators

- $u \sim ISD\{dep0\}$ and $v \sim ISD\{dep1\}$
- No masks in common: $dep0 \cap dep1 \cap M = \emptyset$

$\implies (u \ op \ v) \sim ISD\{dep0 \cup dep1\}$ for every binary operation $op$

▷ More details in the paper
Running Example

Type inference for the last instruction $i4$:

$$(k \oplus m_1) \& ((k \oplus m_2) \& m_3)$$

$\triangleright i4$ is statistically independent from $k$
Bit Level Analysis

When no conclusion is possible at word level:

⇒ split the expression into several expressions at bit level

▷ case 1:

\[ e_n \ldots e_2 e_1 e_0 \]
\[ m_n m_2 m_1 m_0 \]

\( e_i \sim \text{RUD and different dominant mask for each } e_i \)

▷ case 2:

\[ e_n \ldots e_{i+1} e_i e_{i-1} \ldots e_0 \]
\[ \underline{\text{CST}} \quad \underline{\text{ISD}} \quad \underline{\text{CST}} \]

Concatenation of an ISD bit with CST bits

▷ case 3:

\[ e_n \ldots e_i \ldots e_i \ldots e_0 \]
\[ \underline{\text{CST}} \quad \underline{\text{ISD}} \quad \underline{\text{CST}} \quad \underline{\text{ISD}} \quad \underline{\text{CST}} \]

Deduplicated ISD bit and concatenation with CST bits

Example from mix columns in AES:

\[ e = (LSR(mt1 \oplus mp \oplus sbox5, 7) \oplus LSR(mt2 \oplus mp \oplus sbox10, 7)) + (((LSR(mt1 \oplus mp \oplus sbox5, 7) \oplus LSR(mt2 \oplus mp \oplus sbox10, 7)) \ll 1) \]
\[ b_7 = mt17 \oplus mp7 \oplus sbox57 \oplus mt27 \oplus mp7 \oplus sbox107 \]

\[ e \rightarrow 0000 \ 00b_7b_7 \rightarrow \text{ISD} \]
Plan

1. Introduction / Motivation
2. Symbolic Method
3. Experiments
4. Conclusion
Comparison with Two Methods

- **Our method**: distribution type inference implemented in Python

- **C-enumerative**: generates a C program that computes the expression distribution by enumerating on all variable values
  - returns: RUD, ISD or vulnerable

- **SMT-enumerative**: extends Eldib et al.’s approach for $n$-bit variables (generates a SMT problem that searches for two values of a secret for which the expression distribution is different)
  - returns: ISD or vulnerable
## Benchmarks

<table>
<thead>
<tr>
<th>Program</th>
<th>#ASM inst</th>
<th>Size in bits</th>
<th># masks</th>
<th># secrets</th>
<th>Secure in literature</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Boolean programs for comparison with SMT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6 [Eldib, TACAS14]</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>×</td>
</tr>
<tr>
<td>Masked Chi [Eldib, TACAS14]</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Algorithms for switching between boolean and arithmetic maskings</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goubin Conversion [Goubin01]</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>Coron Conversion [Coron15]</td>
<td>37</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Cryptographic algorithms</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Masked AES 1st round [Herbst06]</td>
<td>422</td>
<td>8</td>
<td>6</td>
<td>16 + 16</td>
<td>✓</td>
</tr>
<tr>
<td>Simon TI 1st round [Shahverdi17]</td>
<td>15</td>
<td>32</td>
<td>5</td>
<td>3 + 2</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Experimental Comparison

<table>
<thead>
<tr>
<th>Program</th>
<th>Ref (enumeration)</th>
<th>Symbolic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># RUD</td>
<td># ISD</td>
</tr>
<tr>
<td>P6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Masked Chi</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Goubin Conversion</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Coron Conversion</td>
<td>19</td>
<td>11</td>
</tr>
<tr>
<td>Masked AES 1st round</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Simon TI 1st round</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Enumeration methods $\implies$ sound, complete but not applicable on AES/Simon
- Symbolic method $\implies$ sound $\{\text{Vuln}\} \subseteq \{\text{UKD}\}$ but not complete
## Verification Time

<table>
<thead>
<tr>
<th>Program</th>
<th>Symbolic time</th>
<th>Enum C time</th>
<th>SMT time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P6</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>Masked Chi</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
</tr>
<tr>
<td>Goubin Conversion</td>
<td>&lt;1s</td>
<td>&lt;1s</td>
<td>35mn</td>
</tr>
<tr>
<td>Coron Conversion</td>
<td>2s</td>
<td>1s</td>
<td>5.6h</td>
</tr>
<tr>
<td>Masked AES 1st round</td>
<td>22s</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Simon TI 1st round</td>
<td>8.5s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- **C-enumeration**: fast but only for small programs
- **SMT-enumeration**: can be long even for small programs
- **Symbolic method**: better scalability
Bit Level vs. Word Level Analysis

<table>
<thead>
<tr>
<th>Program</th>
<th>#UKD_w</th>
<th>#UKD_b</th>
<th>#total inst</th>
</tr>
</thead>
<tbody>
<tr>
<td>P6</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Masked Chi</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Goubin Conversion</td>
<td>3</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Coron Conversion</td>
<td>21</td>
<td>13</td>
<td>37</td>
</tr>
<tr>
<td>Masked AES 1st round</td>
<td>80</td>
<td>0</td>
<td>422</td>
</tr>
<tr>
<td>Simon 1st round</td>
<td>7</td>
<td>4</td>
<td>15</td>
</tr>
</tbody>
</table>

With bit level analysis:

- For Coron Conversion & Simon TI: around 40% of unsafe instructions become safe
- For Masked AES: ALL unsafe instructions become safe
Plan

1. Introduction / Motivation
2. Symbolic Method
3. Experiments
4. Conclusion
We proposed a symbolic method:
- For verifying side channel robustness of 1st order masked programs at assembly level
- Using type inference of expression distributions
- Scalable, sound but not complete

Perspectives for future work:
- Automatic tool that analyses an assembly code
- Refine the set of rules / bit level analysis
- Combine with enumerative approach at bit level (need to consider inter-bit dependencies)
- Extend to other leakage models (e.g. transition-based model) / higher masking orders
References


[Eldib,TACAS14] Hassan Eldib, Chao Wang, Patrick Schaumont. SMT-Based Verification of Software Countermeasures against Side-Channel Attacks. TACAS 2014: 62-77


Thank you for your attention!
Algo 1 Distribution inference algorithm

1. function infer(e)
2. if e is a leaf then
3. if e ∈ S then return UKD{e}
4. else if e ∈ M then return RUD{e}
5. else return CST
6. else
7. le{ld} = infer(e.left_child)
8. re{rd} = infer(e.right_child)
9. if ∃ rule for (le{ld} e.op re{rd}) that returns RUD{dep} then
   return RUD{dep}
10. else if ∃ rule for (le{ld} e.op re{rd}) that returns ISD{dep} then
    return ISD{dep}
11. else return UKD{dep}