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

Workshop PROOFS

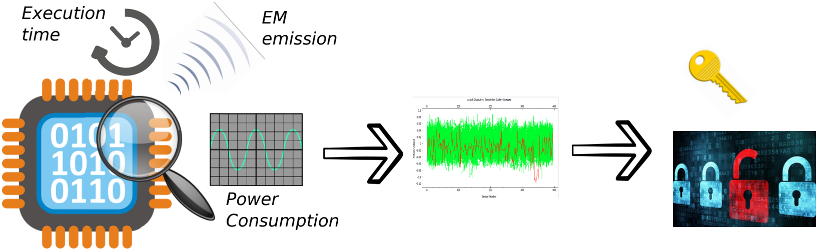
Inès Ben El Ouahma Quentin Meunier Karine Heydemann
Emmanuelle Encrenaz

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



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 $\Longleftrightarrow$ result is statistically independent from secrets
- With a symbolic approach that infers the distribution type of instruction expressions

Plan

1 Introduction / Motivation

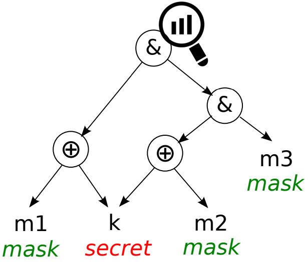
2 Symbolic Method

3 Experiments

4 Conclusion

Verification Scheme

```
    # r0 ← k; r1 ← m1; r2 ← m2; r3 ← m3
    eor r4, r0, r1   # k ⊕ m1
    eor r5, r0, r2   # k ⊕ m2
    and r5, r5, r3   # (k ⊕ m2) & m3
    and r5, r5, r4   # (k ⊕ m1) & ((k ⊕ m2) & m3)
```



Data dependency graph of the last instruction

Is the root distribution statistically independent from k?

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

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
m_1, m_2: masks
$e = (k \oplus m_1) \,\&\, m_2$
$e' = (k \oplus m_1) \,\&\, m_1$

k	m_1	m_2	e	e'
0	0	0	0	0
	0	1	0	0
	1	0	0	1
	1	1	1	1
1	0	0	0	0
	0	1	1	0
	1	0	0	0
	1	1	0	0

For $k=0$: $P(e=0)=\frac{3}{4}$, $P(e=1)=\frac{1}{4}$; $P(e'=0)=\frac{1}{2}$, $P(e'=1)=\frac{1}{2}$

For $k=1$: $P(e=0)=\frac{3}{4}$, $P(e=1)=\frac{1}{4}$; $P(e'=0)=1$, $P(e'=1)=0$

Independence Notions

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

Dependence between expression e and variable v:

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

$\Longrightarrow$ Need to keep track of structural dependencies: $(\mathtt{k} \oplus \mathtt{m})\ \&\ \mathtt{m}$

Safe types:

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

Unsafe type:

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

Dominant Masks

Aim: to find a mask that randomizes the whole expression

Dom Rule

- expression $\mathbf{e} = \mathbf{e'} \oplus \mathbf{m}$ or $\mathbf{e} = \mathbf{e'} + \mathbf{m} \bmod 2^n$
- $\mathbf{m} \sim \mathbf{RUD}\{\mathbf{m}\}$
- $\mathbf{m} \notin \text{dep}(\mathbf{e'})$

$\Longrightarrow$ $\mathbf{e} \sim \mathbf{RUD}$ and $\mathbf{m}$ is a dominant mask of $\mathbf{e}$.

2 sets of dominant masks:

- $\mathtt{dom}_{\oplus}\mathtt{(e)}$ the set of xor dominant masks of e
- $\mathtt{dom}_{+}\mathtt{(e)}$ the set of additive dominant masks of e

Examples:

- $\mathtt{dom}_{\oplus}\mathtt{((k + m1) \oplus (k \oplus m1 \oplus m2)) = m2}$
- $\mathtt{dom}_{+}\mathtt{((k + m1) \oplus 0) = dom}_{+}\mathtt{(k + m1) = m1}$

Other Inference Rules

By distribution types:

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

Disjoint rule for binary operators

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

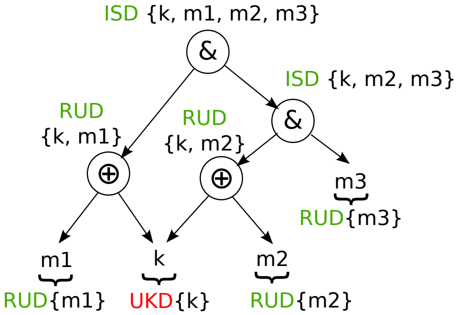
$\Longrightarrow$ $(\mathtt{u\ op\ v}){\sim}\mathtt{ISD}\{\mathtt{dep0} \cup \mathtt{dep1}\}$ for every binary operation op

▷ More details in the paper

Running Example

Type inference for the last instruction *i4*:

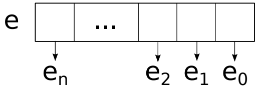
$(\mathtt{k} \oplus \mathtt{m}_1)\ \&\ ((\mathtt{k} \oplus \mathtt{m}_2)\ \&\ \mathtt{m}_3)$



▷ *i4* is statistically independent from `k`

Bit Level Analysis

When no conclusion is possible at word level:
$\Longrightarrow$ split the expression into several expressions at bit level



▷ case 1:

$$\begin{array}{ccccc} e_n & \cdots & e_2 & e_1 & e_0 \\ \downarrow & & \downarrow & \downarrow & \downarrow \\ m_n & & m_2 & m_1 & m_0 \end{array}$$

e_i~RUD and different dominant mask for each e_i

▷ case 2:

$$\underbrace{e_n \cdots e_{i+1}}_{\text{CST}}\ \underbrace{e_i}_{\text{ISD}}\ \underbrace{e_{i-1} \cdots e_0}_{\text{CST}}$$

Concatenation of an ISD bit with CST bits

▷ case 3:

$$\underbrace{e_n \cdots}_{\text{CST}}\ \underbrace{e_i}_{\text{ISD}}\ \underbrace{\cdots}_{\text{CST}}\ \underbrace{e_i}_{\text{ISD}}\ \underbrace{\cdots e_0}_{\text{CST}}$$

Deduplicated ISD bit and concatenation with CST bits

Example from mix columns in AES:

$\texttt{e = ((LSR(mt1} \oplus \texttt{mp} \oplus \texttt{sbox5, 7)} \oplus \texttt{LSR(mt2} \oplus \texttt{mp} \oplus \texttt{sbox10, 7)) +}$
$\quad \texttt{(((LSR(mt1} \oplus \texttt{mp} \oplus \texttt{sbox5, 7)} \oplus \texttt{LSR(mt2} \oplus \texttt{mp} \oplus \texttt{sbox10, 7))} \ll \texttt{1)}$

$\texttt{b}_7 = \texttt{mt1}_7 \oplus \cancel{\texttt{mp}_7} \oplus \texttt{sbox5}_7 \oplus \texttt{mt2}_7 \oplus \cancel{\texttt{mp}_7} \oplus \texttt{sbox10}_7$

$\texttt{e} \Longrightarrow \texttt{0000 00b}_7\texttt{b}_7 \Longrightarrow \texttt{ISD}$

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

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

Experimental Comparison

Program	Ref (enumeration)			Symbolic			
	# RUD	# ISD	# Vuln	# RUD	# ISD	# UKD	# CST
P6	6	2	0	6	2	0	0
Masked Chi	2	2	4	2	2	4	0
Goubin Conversion	7	1	0	5	0	3	0
Coron Conversion	19	11	7	14	10	13	0
Masked AES 1st round	-	-	-	302	0	0	120
Simon TI 1st round	-	-	-	7	4	3	1

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

Verification Time

Program	Symbolic time	Enum C time	SMT time
P6	<1s	<1s	<1s
Masked Chi	<1s	<1s	<1s
Goubin Conversion	<1s	<1s	35mn
Coron Conversion	2s	1s	5,6h
Masked AES 1st round	22s	-	-
Simon TI 1st round	8.5s	-	-

- C-enumeration $\Longrightarrow$ fast but only for small programs
- SMT-enumeration $\Longrightarrow$ can be long even for small programs
- Symbolic method $\Longrightarrow$ better scalability

Bit Level vs. Word Level Analysis

Program	#$\mathtt{UKD}_w$	#$\mathtt{UKD}_b$	#total inst
P6	0	0	8
Masked Chi	4	4	8
Goubin Conversion	3	3	8
Coron Conversion	21	13	37
Masked AES 1st round	80	0	422
Simon 1st round	7	4	15

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

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

[Bayrak,CHES13] Ali Galip Bayrak, Francesco Regazzoni, David Novo, Paolo Ienne. Sleuth: Automated Verification of Software Power Analysis Countermeasures. *CHES 2013: 293-310*

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

[Barthe,Eurocrypt15] Gilles Barthe, Sonia Belad, Franois Dupressoir, Pierre-Alain Fouque, Benjamin Grgoire, Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. *EUROCRYPT (1) 2015: 457-485*

[Goubin01] Louis Goubin. A sound method for switching between boolean and arithmetic masking. In Cryptographic Hardware and Embedded SystemsCHES 2001, pages 315. Springer, 2001.

[Coron15] Jean-S ebastien Coron, Johann Grosch adl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to boolean masking with logarithmic complexity. In International Workshop on Fast Software Encryption, pages 130149. Springer, 2015.

[Herbst06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An aes smart card implementation resistant to power analysis attacks. In ACNS, volume 3989, pages 239252. Springer, 2006.

[Shahverdi17] Aria Shahverdi, Mostafa Taha, and Thomas Eisenbarth. Lightweight side channel resistance. Threshold implementations of simon. IEEE Transactions on Computers, 66(4):661671, 2017.

Thank you for your attention!

Backup Slide 1

Algorithm 1 Distribution inference algorithm

```
function INFER(E)
    if e is a leaf then
        if e ∈ S then return UKD{e}
        else if e ∈ M then return RUD{e}
        else return CST
    else
        le{ld} = infer(e.left_child)
        re{rd} = infer(e.right_child)
        if ∃ rule for (le{ld} e.op re{rd}) that returns RUD{dep} then
            return RUD{dep}
        else if ∃ rule for (le{ld} e.op re{rd}) that returns ISD{dep} then
            return ISD{dep}
        else return UKD{dep}
```