Verified Proofs of Higher-Order Masking

Gilles Barthe[1] Sonia Belaïd[2] François Dupressoir[1]
Pierre-Alain Fouque[3] Benjamin Grégoire[4] Pierre-Yves Strub[1]

[1]IMDEA Software Institute,

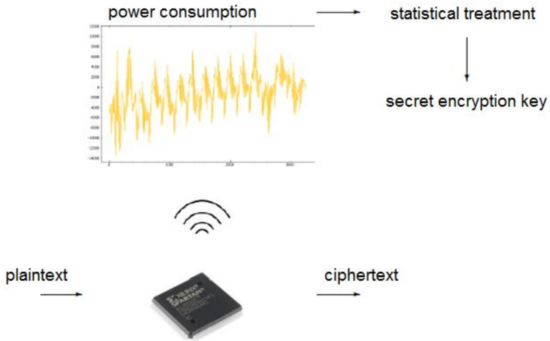
[2]École normale supérieure and Thales Communications & Security,

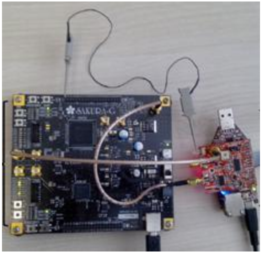
[3]Université de Rennes 1 and Institut Universitaire de France,

[4]Inria

Outline

Side-Channel Attacks

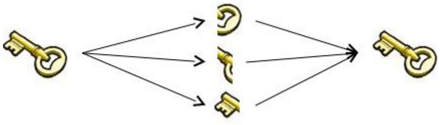




- observation of device leaks (power consumption) during the execution of a cryptographic algorithm
- analysis of this consumption to recover secrets

Masking

- countermeasure which aims to render partial power consumption traces independent from the secrets by randomizing them
- each sensitive value x is replaced in the computations by $t+1$ random variables x_0, ..., x_t such that $x = x_0 \star ... \star x_t$



- generally, we consider that an adversary that observes at most t program variables should not be able to recover x
- t is called *masking order* or *security order*

Security of Masked Programs: Leakage Model

- [IshaiSahaiWagner,Crypto'03] t-threshold probing model
  - convenient to make security proofs
  - ✘ not very relevant in practice

security in the t-threshold probing model

Security of Masked Programs: Leakage Model

- [IshaiSahaiWagner,Crypto'03] *t*-threshold probing model
  - convenient to make security proofs
  - ✘ not very relevant in practice
- [ProuffRivain,Eurocrypt'13] noisy leakage model
  - relevant in practice
  - ✘ not convenient to make security proofs

security in the t-threshold probing model	security in the noisy leakage model

Security of Masked Programs: Leakage Model

- [IshaiSahaiWagner,Crypto'03] *t*-threshold probing model
  - convenient to make security proofs
  - ✘ not very relevant in practice
- [ProuffRivain,Eurocrypt'13] noisy leakage model
  - relevant in practice
  - ✘ not convenient to make security proofs
- [DucDziembowskiFaust,Eurocrypt'14] reduction between *t*-threshold probing model to noisy leakage model
  - relevant in practice
  - convenient to make security proofs

security in the *t*-threshold probing model $\longrightarrow$ security in the noisy leakage model

Security in the t-threshold probing model

Security proof: to prove the security of a program in the t-threshold probing model, it is *enough* to show that any set of t observations can be simulated independently from the secret. (*here, observation* $=$ *intermediate variable*)

Security in the t-threshold probing model

Security proof: to prove the security of a program in the t-threshold probing model, it is *enough* to show that any set of t observations can be simulated independently from the secret. (*here, observation* $=$ *intermediate variable*)

Current Issues in the 'cryptographic' security proofs:

- absence of security proof,
- mistakes in security proofs,
- performances issues (too many refreshings, too many shares, ...)

Security in the t-threshold probing model

Security proof: to prove the security of a program in the t-threshold probing model, it is *enough* to show that any set of t observations can be simulated independently from the secret. (*here, observation* = *intermediate variable*)

Current Issues in the 'cryptographic' security proofs:

- absence of security proof,
- mistakes in security proofs,
- performances issues (too many refreshings, too many shares, ...)

Current Issues in the 'formal' security proofs:

- ➜ either the approach is not complete, *i.e.*, insecure programs typed as secure
- ➜ or they rely on counting the solutions which is exponential in the program size

Outline

Our Contribution

New algorithms to automatically and efficiently verify security of masked programs:

Our Contribution

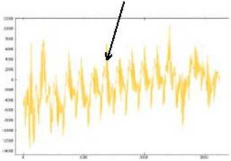
New algorithms to automatically and efficiently verify security of masked programs:

- Security in the t-threshold probing model with no false positive

Our Contribution

New algorithms to automatically and efficiently verify security of masked programs:

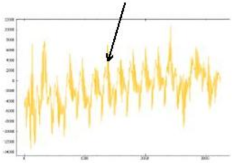
- Security in the t-threshold probing model with no false positive
- Parametric in the leakage model
    - value-based
    - transition-based
    - ...



Our Contribution

New algorithms to automatically and efficiently verify security of masked programs:

- Security in the *t*-threshold probing model with no false positive
- Parametric in the leakage model
  - value-based
  - transition-based
  - ...



- Complexity
  - ➜ non exponential techniques to prove the independence of one set of observations from the secret
  - ➜ faster methods to test all the possible sets
  - ➜ verification of high orders programs (> 2)

Outline

Verification in two steps

Security proof: to prove the security of a program in the t-threshold probing model, it is *enough* to show that any set of t observations can be simulated independently from the secret.

Verification in two steps

> Security proof: to prove the security of a program in the t-threshold probing model, it is *enough* to show that any set of t observations can be simulated independently from the secret.

Verification in two steps:

1. Prove that a set of intermediate variables is jointly independent from the secret (non-interferent)

Verification in two steps

Security proof: to prove the security of a program in the t-threshold probing model, it is *enough* to show that any set of t observations can be simulated independently from the secret.

Verification in two steps:

1. Prove that a set of intermediate variables is jointly independent from the secret (non-interferent)
2. Prove that every set of t intermediate variables is independent from the secret

1. Verifying Sets' Non-Interference

Proving probabilistic non-interference of a set of intermediate variables $\mathcal{I}$[1]:

(Rule 1) all the deterministic variables in $\mathcal{I}$ are public $\Rightarrow \mathcal{I} \perp \mathcal{S}$

(Rule 2) $\mathcal{I}$ and $\mathcal{I}'$ are provably equivalent and $\mathcal{I}' \perp \mathcal{S} \Rightarrow \mathcal{I} \perp \mathcal{S}$

(Rule 3) $\exists\, (\mathcal{I}', v, r \in \mathcal{R})$ such that

$$\left.\begin{array}{l} \text{- } v \text{ is invertible in } r, \\ \text{- } r \text{ appears only in } v, \\ \text{- } \mathcal{I}' = \mathcal{I}\,\{\text{where } r \text{ replaces } v\} \quad \perp \mathcal{S} \end{array}\right\} \Rightarrow \mathcal{I} \perp \mathcal{S}$$

[1] $\mathcal{I} \perp \mathcal{S} \equiv$ the joint distribution of $\mathcal{I}$ is independent from the secrets $\mathcal{S}$

1. Verifying Sets' Non-Interference

Proving probabilistic non-interference of a set of intermediate variables $\mathcal{I}$[1]:

(Rule 1) all the deterministic variables in $\mathcal{I}$ are public $\Rightarrow \mathcal{I} \perp \mathcal{S}$

(Rule 2) $\mathcal{I}$ and $\mathcal{I}'$ are provably equivalent and $\mathcal{I}' \perp \mathcal{S} \Rightarrow \mathcal{I} \perp \mathcal{S}$

(Rule 3) $\exists\,(\mathcal{I}', v, r \in \mathcal{R})$ such that

$$\left.\begin{array}{l} \text{- } v \text{ is invertible in } r, \\ \text{- } r \text{ appears only in } v, \\ \text{- } \mathcal{I}' = \mathcal{I}\,\{\text{where } r \text{ replaces } v\} \quad \perp \mathcal{S} \end{array}\right\} \Rightarrow \mathcal{I} \perp \mathcal{S}$$

Example: $\mathcal{I} = \{a \oplus b,\ r \oplus c,\ a \oplus c\} \quad \Rightarrow \quad \mathcal{I}' = \{a \oplus b,\ r,\ a \oplus c\}$

---

[1] $\mathcal{I} \perp \mathcal{S} \equiv$ the joint distribution of $\mathcal{I}$ is independent from the secrets $\mathcal{S}$

1. Verifying Sets' Non-Interference

Proving probabilistic non-interference of a set of intermediate variables $\mathcal{I}$[1]:

(Rule 1) all the deterministic variables in $\mathcal{I}$ are public $\Rightarrow \mathcal{I} \perp \mathcal{S}$

(Rule 2) $\mathcal{I}$ and $\mathcal{I}'$ are provably equivalent and $\mathcal{I}' \perp \mathcal{S} \Rightarrow \mathcal{I} \perp \mathcal{S}$

(Rule 3) $\exists\, (\mathcal{I}', v, r \in \mathcal{R})$ such that

- v is invertible in r,
- r appears only in v,
- $\mathcal{I}' = \mathcal{I}\,\{\text{where } r \text{ replaces } v\} \quad \perp \mathcal{S}$

$\Big\} \Rightarrow \mathcal{I} \perp \mathcal{S}$

Example: $\mathcal{I} = \{a \oplus b,\ r \oplus c,\ a \oplus c\} \quad \Rightarrow \quad \mathcal{I}' = \{a \oplus b,\ r,\ a \oplus c\}$

➜ every set proven non-interferent *is* non-interferent

➜ *no false negative* in our experiments

➜ *not exponential* in the size of the expressions

➜ resulting proofs can be *easily checked*

---

[1] $\mathcal{I} \perp \mathcal{S} \equiv$ the joint distribution of $\mathcal{I}$ is independent from the secrets $\mathcal{S}$

2. Extension to All Possible Sets

Complexity/Issue: for n intermediate variables $\Rightarrow \binom{n}{t}$ proofs of independence (e.g., $\approx 2^{27}$ for 4 rounds of a 2^{nd}-order AES)

2. Extension to All Possible Sets

Complexity/Issue: for n intermediate variables $\Rightarrow \binom{n}{t}$ proofs of independence (e.g., $\approx 2^{27}$ for 4 rounds of a 2^{nd}-order AES)

New Idea: proving independence for larger sets of more than t elements (extending the set and checking again is very fast !)

2. Extension to All Possible Sets

Complexity/Issue: for n intermediate variables $\Rightarrow \binom{n}{t}$ proofs of independence (e.g., $\approx 2^{27}$ for 4 rounds of a 2^{nd}-order AES)

New Idea: proving independence for larger sets of more than t elements (extending the set and checking again is very fast !)

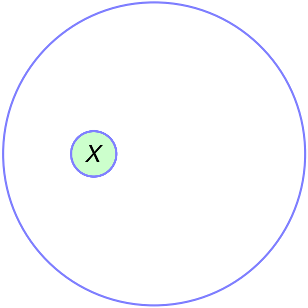
- Alg. 1 - Workpair-based splitting: split in 2 then merge

2. Extension to All Possible Sets

Complexity/Issue: for n intermediate variables $\Rightarrow \binom{n}{t}$ proofs of independence (e.g., $\approx 2^{27}$ for 4 rounds of a 2^{nd}-order AES)

New Idea: proving independence for larger sets of more than t elements (extending the set and checking again is very fast !)

Alg. 1 - Workpair-based splitting: split in 2 then merge



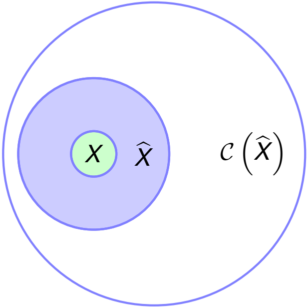
1. select X with t observations and prove its non-interference

2. Extension to All Possible Sets

Complexity/Issue: for n intermediate variables $\Rightarrow \binom{n}{t}$ proofs of independence (e.g., $\approx 2^{27}$ for 4 rounds of a 2nd-order AES)

New Idea: proving independence for larger sets of more than t elements (extending the set and checking again is very fast !)

Alg. 1 - Workpair-based splitting: split in 2 then merge



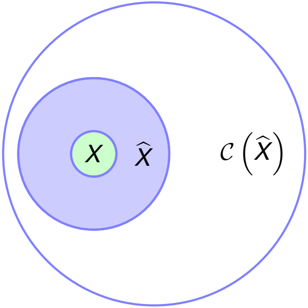
1. select X with t observations and prove its non-interference
2. extend X to $\widehat{X}$ with many more observations but still non-interferent

2. Extension to All Possible Sets

Complexity/Issue: for n intermediate variables $\Rightarrow \binom{n}{t}$ proofs of independence (e.g., $\approx 2^{27}$ for 4 rounds of a 2nd-order AES)

New Idea: proving independence for larger sets of more than t elements (extending the set and checking again is very fast !)

Alg. 1 - Workpair-based splitting: split in 2 then merge



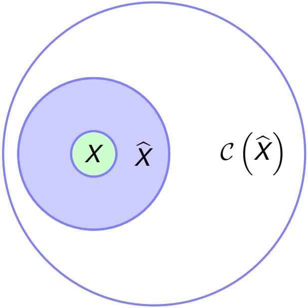
1. select X with t observations and prove its non-interference
2. extend X to $\widehat{X}$ with many more observations but still non-interferent
3. recursively descend in set $\mathcal{C}\left(\widehat{X}\right)$

2. Extension to All Possible Sets

Complexity/Issue: for n intermediate variables $\Rightarrow \binom{n}{t}$ proofs of independence (e.g., $\approx 2^{27}$ for 4 rounds of a 2nd-order AES)

New Idea: proving independence for larger sets of more than t elements (extending the set and checking again is very fast !)

Alg. 1 - Workpair-based splitting: split in 2 then merge



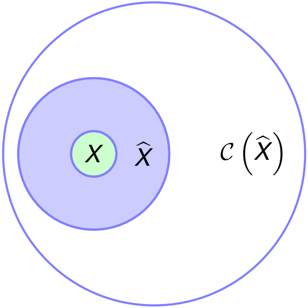
1. select X with t observations and prove its non-interference
2. extend X to $\widehat{X}$ with many more observations but still non-interferent
3. recursively descend in set $\mathcal{C}\left(\widehat{X}\right)$
4. merge $\widehat{X}$ and $\mathcal{C}\left(\widehat{X}\right)$ once they are processed separately.

2. Extension to All Possible Sets

Complexity/Issue: for n intermediate variables $\Rightarrow \binom{n}{t}$ proofs of independence (e.g., $\approx 2^{27}$ for 4 rounds of a 2^{nd}-order AES)

New Idea: proving independence for larger sets of more than t elements (extending the set and checking again is very fast !)

Alg. 1 - Workpair-based splitting: split in 2 then merge
Alg. 2 - Worklist-based splitting: split in more than 2



1. select X with t observations and prove its non-interference
2. extend X to $\widehat{X}$ with many more observations but still non-interferent
3. recursively descend in set $\mathcal{C}\left(\widehat{X}\right)$
4. merge $\widehat{X}$ and $\mathcal{C}\left(\widehat{X}\right)$ once they are processed separately.

Application to the Sbox [CPRR13, Algorithm 4]

Method	# tuples	Security	Complexity # sets	Complexity time*
First-Order Masking				
naive	63	✓	63	0.001s
workpair	63	✓	17	0.001s
worklist	63	✓	17	0.001s
Second-Order Masking				
naive	12,561	✓	12,561	0.180s
workpair	12,561	✓	851	0.046s
worklist	12,561	✓	619	0.029s
Third-Order Masking				
naive	4,499,950	✓	4,499,950	140.642s
workpair	4,499,950	✓	68,492	9.923s
worklist	4,499,950	✓	33,075	3.894s
Fourth-Order Masking				
naive	2,277,036,685	✓	-	unpractical
workpair	2,277,036,685	✓	8,852,144	2959.770s
worklist	2,277,036,685	✓	3,343,587	879.235s

*run on a headless VM with a dual core (only one core is used in the computation) 64-bit processor clocked at 2GHz

Outline

Benchmarks for the Value-Based Model

Reference	Target	# tuples	Security	Complexity # sets	Complexity time (s)
First-Order Masking					
CHES10	$\odot$	13	✓	7	ε
FSE13	Sbox	63	✓	17	ε
FSE13	full AES	17,206	✓	3,342	128
MAC-SHA3	full Keccak-f	13,466	✓	5,421	405
Second-Order Masking					
RSA06	Sbox	1,188,111	✓	4,104	1.649
CHES10	$\odot$	435	✓	92	0.001
CHES10	Sbox	7,140	1^{st}-order flaws (2)	866	0.045
CHES10	AES KS	23,041,866	✓	771,263	340,745
FSE13	2 rnds AES	25,429,146	✓	511,865	1,295
FSE13	4 rnds AES	109,571,806	✓	2,317,593	40,169
Third-Order Masking					
RSA06	Sbox	2,057,067,320	3^{rd}-order flaws (98, 176)	2,013,070	695
CHES10	$\odot$	24,804	✓	1,410	0.033
FSE13	Sbox(4)	4,499,950	✓	33,075	3.894
FSE13	Sbox(5)	4,499,950	✓	39,613	5.036
Fourth-Order Masking					
CHES10	$\odot$	2,024,785	✓	33,322	1.138
FSE13	Sbox (4)	$2,277,036,685$	✓	3,343,587	879
Fifth-Order Masking					
CHES10	$\odot$	216,071,394	✓	856,147	45

Outline

Conclusion

- Summary
  - new algorithms to automatically verify security of masked programs
  - no false positive, *i.e., a program typed as secure is secure*
  - verification programs at high orders (> 2)

- Further Work
  - ➜ verify larger masked programs at higher orders
  - ➜ exhibit and prove efficient methods to compose
  - ➜ adapt to more practical languages

Conclusion

- Summary
  - new algorithms to automatically verify security of masked programs
  - no false positive, *i.e., a program typed as secure is secure*
  - verification programs at high orders (> 2)
- Further Work
  - ➜ verify larger masked programs at higher orders
  - ➜ exhibit and prove efficient methods to compose
  - ➜ adapt to more practical languages

Thank you for your attention.