Formal Verification of Masked Implementations

Sonia Belaïd   Benjamin Grégoire

CHES 2018 - Tutorial

September 9th 2018

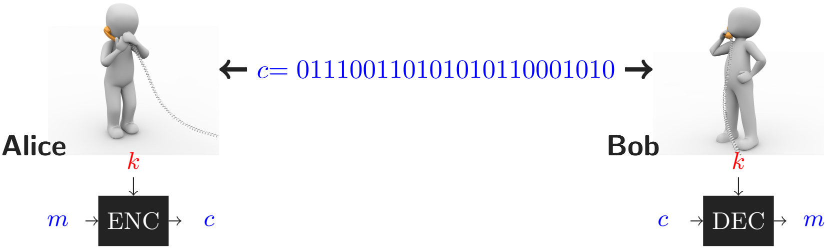
1. Side-Channel Attacks and Masking

2. Formal Tools for Verification at Fixed Order

3. Formal Tools for Verification of Generic Implementations

Cryptanalysis

- Black-box cryptanalysis
- Side-channel analysis



Cryptanalysis

- Black-box cryptanalysis: $\mathcal{A} \leftarrow (m, c)$
- Side-Channel Analysis

Alice

$\leftarrow$ c= 01110011010101011000101 0 $\rightarrow$

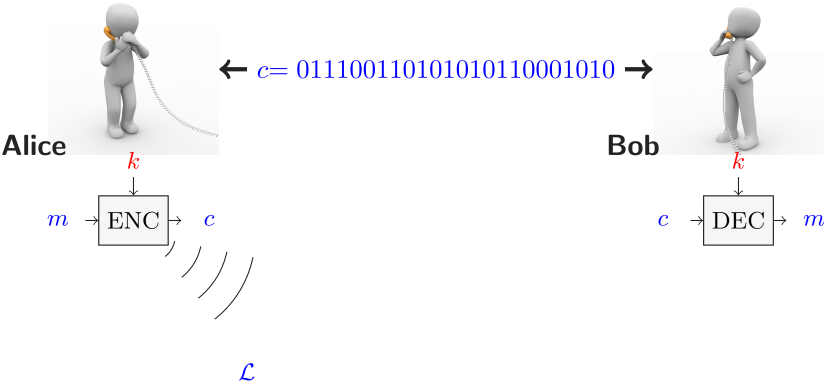
Bob

$$m \rightarrow \boxed{\text{ENC}} \rightarrow c \quad (k \downarrow \text{ENC})$$

$$c \rightarrow \boxed{\text{DEC}} \rightarrow m \quad (k \downarrow \text{DEC})$$

Cryptanalysis

- Black-box cryptanalysis
- Side-Channel Analysis: $\mathcal{A} \leftarrow (m, c, \mathcal{L})$



Cryptanalysis

- ➜ Black-box cryptanalysis
- ➜ Side-Channel Analysis: $\mathcal{A} \leftarrow (m, c, \mathcal{L})$

$\leftarrow c = 011100110101010110001010 \rightarrow$

Alice

$$m \rightarrow \boxed{\text{ENC}} \rightarrow c \quad (\text{key } k)$$

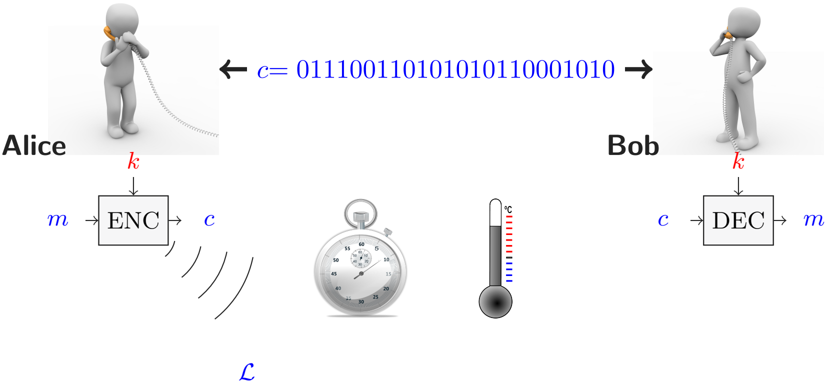
$\mathcal{L}$

Bob

$$c \rightarrow \boxed{\text{DEC}} \rightarrow m \quad (\text{key } k)$$

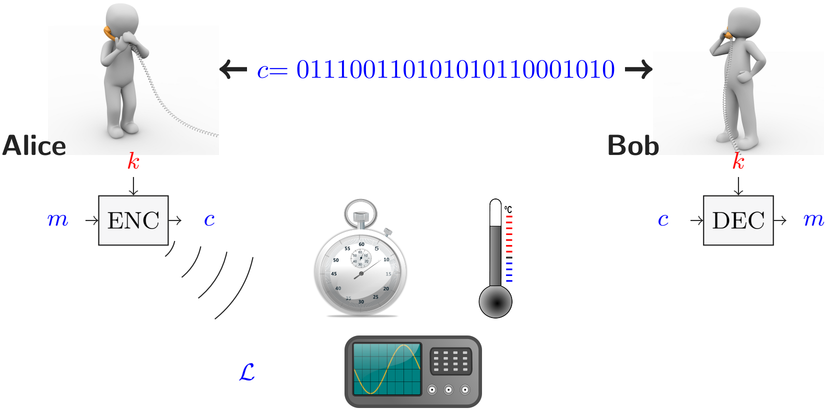
Cryptanalysis

- ➜ Black-box cryptanalysis
- ➜ Side-Channel Analysis: $\mathcal{A} \leftarrow (m, c, \mathcal{L})$



Cryptanalysis

- Black-box cryptanalysis
- Side-Channel Analysis: $\mathcal{A} \leftarrow (m, c, \mathcal{L})$



Cryptanalysis

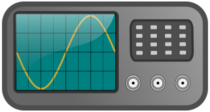
- ➜ Black-box cryptanalysis
- ➜ Side-Channel Analysis: $\mathcal{A} \leftarrow (m, c, \mathcal{L})$

Alice

$\leftarrow$ c= 0111001101010110001010 $\rightarrow$

Bob

$$m \rightarrow \boxed{\mathrm{ENC}_{k}} \rightarrow c \qquad\qquad c \rightarrow \boxed{\mathrm{DEC}_{k}} \rightarrow m$$

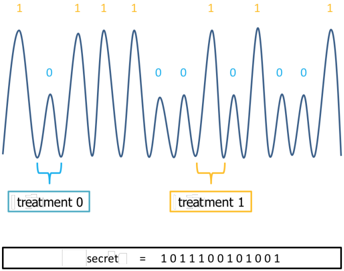


$\mathcal{L}$

Example of SPA

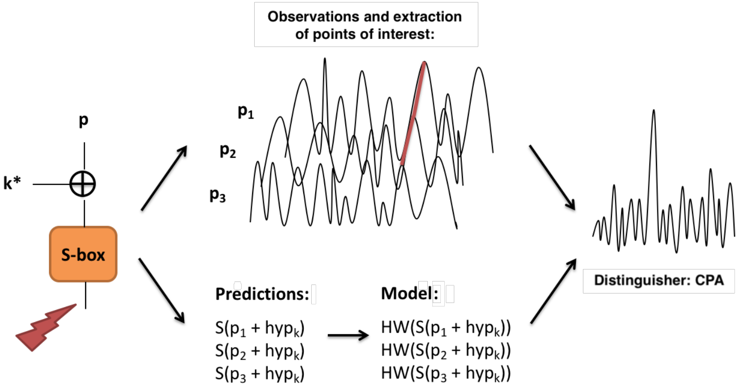
Algorithm 1 Example

```
for i = 1 to n do
    if key[i] = 0 then
        do treatment 0
    else
        do treatment 1
    end if
end for
```



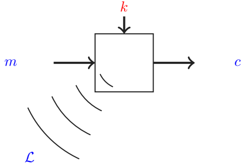
SPA: one single trace to recover the secret key

Example of DPA



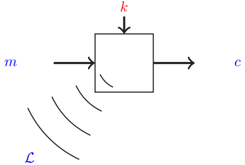
DPA: several traces to recover the secret key

How to thwart SCA?



Issue: leakage $\mathcal{L}$ is key-dependent

How to thwart SCA?



Issue: leakage $\mathcal{L}$ is key-dependent

Idea of masking: make leakage $\mathcal{L}$ random

sensitive value: $v = f(m, k)$

$$v_0 \leftarrow v \oplus \left(\bigoplus_{1 \leqslant i \leqslant t} v_i \right) \quad v_1 \leftarrow \$ \quad \cdots \quad v_t \leftarrow \$$$

➜ any t-uple of v_i is independent from v

Masked Implementations

- Linear functions: apply the function to each share

$$v \oplus w \to (v_0 \oplus w_0, v_1 \oplus w_1, \ldots, v_t \oplus w_t)$$

Masked Implementations

- Linear functions: apply the function to each share

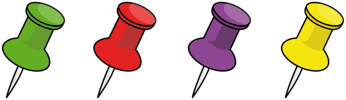
$$v \oplus w \to (v_0 \oplus w_0, v_1 \oplus w_1, \ldots, v_t \oplus w_t)$$

- Non-linear functions: much more complex

$$\begin{aligned}
\forall\, 0 \le i < j \le t-1, \qquad & r_{i,j} \leftarrow \$ \\
\forall\, 0 \le i < j \le t-1, \qquad & r_{j,i} \leftarrow (r_{i,j} \oplus v_i w_j) \oplus v_j w_i \\
\forall\, 0 \le i \le d-1, \qquad & c_i \leftarrow v_i w_i \oplus \sum_{j \ne i} r_{i,j} \\
vw \quad \to \quad & (c_0, c_1, \ldots, c_t)
\end{aligned}$$

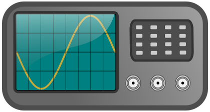
Leakage Models

- **Probing model** by Ishai, Sahai, and Wagner (Crypto 2003)
  - a circuit is t-probing secure iff any set composed of the exact values of at most t intermediate variables is independent from the secret



Leakage Models

- **Probing model** by Ishai, Sahai, and Wagner (Crypto 2003)
  - a circuit is t-probing secure iff any set composed of the exact values of at most t intermediate variables is independent from the secret
- **Noisy leakage model** by Chari, Jutla, Rao, and Rohatgi (Crypto 1999) then Rivain and Prouff (EC 2013)
  - a circuit is secure in the noisy leakage model iff the adversary cannot recover information on the secret from the noisy values of all the intermediate variables



Leakage Models

- **Probing model** by Ishai, Sahai, and Wagner (Crypto 2003)
  - a circuit is t-probing secure iff any set composed of the exact values of at most t intermediate variables is independent from the secret
- **Noisy leakage model** by Chari, Jutla, Rao, and Rohatgi (Crypto 1999) then Rivain and Prouff (EC 2013)
  - a circuit is secure in the noisy leakage model iff the adversary cannot recover information on the secret from the noisy values of all the intermediate variables
- **Reduction** by Duc, Dziembowski, and Faust (EC 2014)
  - t-probing security $\Rightarrow$ security in the noisy leakage model for some level of noise

How to Verify Probing Security?

- variables: secret, shares, constant
- masking order $t = 3$

function Ex-t3(x_0, x_1, x_2, x_3, c):

(* $x_0, x_1, x_2 = \$$ *)
(* $x_3 = x + x_0 + x_1 + x_2$ *)

$r_0 \leftarrow \$$
$r_1 \leftarrow \$$
$y_0 \leftarrow x_0 + r_0$
$y_1 \leftarrow x_3 + r_1$
$t_1 \leftarrow x_1 + r_0$
$t_2 \leftarrow (x_1 + r_0) + x_2$
$y_2 \leftarrow (x_1 + r_0 + x_2) + r_1$
$y_3 \leftarrow c + r_1$

return(y_0, y_1, y_2, y_3)

How to Verify Probing Security?

- variables: secret, shares, constant
- masking order $t = 3$

independent from the secret?

function Ex-t3(x_0, x_1, x_2, x_3, c):

(* $x_0, x_1, x_2 = \$$ *)
(* $x_3 = x + x_0 + x_1 + x_2$ *)

$$
\begin{aligned}
r_0 &\leftarrow \$ \\
r_1 &\leftarrow \$ \\
y_0 &\leftarrow x_0 + r_0 \\
y_1 &\leftarrow x_3 + r_1 \\
t_1 &\leftarrow x_1 + r_0 \\
t_2 &\leftarrow (x_1 + r_0) + x_2 \\
y_2 &\leftarrow (x_1 + r_0 + x_2) + r_1 \\
y_3 &\leftarrow c + r_1
\end{aligned}
$$

return(y_0, y_1, y_2, y_3)

How to Verify Probing Security?

- variables: secret, shares, constant
- masking order $t = 3$

independent from the secret?

function Ex-t3(x_0, x_1, x_2, x_3, c):

(* $x_0, x_1, x_2 = \$$ *)
(* $x_3 = x + x_0 + x_1 + x_2$ *)

$$
\begin{aligned}
r_0 &\leftarrow \$ \\
r_1 &\leftarrow \$ \\
y_0 &\leftarrow x_0 + r_0 \\
y_1 &\leftarrow x_3 + r_1 \\
t_1 &\leftarrow x_1 + r_0 \\
t_2 &\leftarrow (x_1 + r_0) + x_2 \\
y_2 &\leftarrow (x_1 + r_0 + x_2) + r_1 \\
y_3 &\leftarrow c + r_1
\end{aligned}
$$

return(y_0, y_1, y_2, y_3)

Non-Interference (NI)

- t-NI $\Rightarrow$ t-probing secure
- a circuit is t-NI iff any set of t intermediate variables can be perfectly simulated with at most t shares of each input

can be simulated with x_0 and x_1

function Ex-t3(x_0, x_1, x_2, x_3, c):

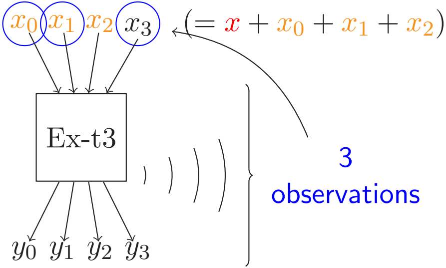
(* $x_0, x_1, x_2 = \$$ *)
(* $x_3 = x + x_0 + x_1 + x_2$ *)

$$
\begin{aligned}
r_0 &\leftarrow \$ \\
r_1 &\leftarrow \$ \\
y_0 &\leftarrow x_0 + r_0 \\
y_1 &\leftarrow x_3 + r_1 \\
t_1 &\leftarrow x_1 + r_0 \\
t_2 &\leftarrow (x_1 + r_0) + x_2 \\
y_2 &\leftarrow (x_1 + r_0 + x_2) + r_1 \\
y_3 &\leftarrow c + r_1
\end{aligned}
$$

return(y_0, y_1, y_2, y_3)

Non-Interference (NI)

- t-NI $\Rightarrow$ t-probing secure
- a circuit is t-NI iff any set of t intermediate variables can be perfectly simulated with at most t shares of each input



State-Of-The-Art

- several tools were built to formally verify security of first-order implementations $t = 1$
- then a sequence of work tackled higher-order implementations $t \leq 5$
  - `maskVerif` from Barthe et al.: first tool to achieve verification at high orders
  - `CheckMasks` from Coron: improvements in terms of efficiency
  - Bloem et al.'s tool: treatment of glitches attacks

State-Of-The-Art

- several tools were built to formally verify security of first-order implementations $t = 1$
- then a sequence of work tackled higher-order implementations $t \leq 5$
  - `maskVerif` from Barthe et al.: first tool to achieve verification at high orders
  - `CheckMasks` from Coron: improvements in terms of efficiency
  - Bloem et al.'s tool: treatment of glitches attacks

maskVerif

- input:
  - pseudo-code of a masked implementation
  - order t
- output:
  - formal proof of t-probing security (or NI, SNI)
  - potential flaws

Gilles Barthe and Sonia Belaïd and François Dupressoir and Pierre-Alain Fouque and Benjamin Grégoire and Pierre-Yves Strub *Verified Proofs of Higher-Order Masking*, EUROCRYPT 2015, Proceedings, Part I, 457–485.

Checking probabilistic independence

Problem: Check if a program expression e is probabilistic independent from a secret s
Example: $e = (s \oplus r_1) \cdot (r_1 \oplus r_2)$

First solution:

- for each value of s computes the associate distribution of e
- if all the resulting distribution are equals then e is independent of s

$$s = 0 \left\{ \begin{array}{ccc} r_1 & r_2 & e \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array} \right. \qquad s = 1 \left\{ \begin{array}{ccc} r_1 & r_2 & e \\ 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{array} \right.$$

Checking probabilistic independence

Problem: Check if a program expression e is probabilistic independent from a secret s
Example: $e = (s \oplus r_1) \cdot (r_1 \oplus r_2)$

First solution:

- for each value of s computes the associate distribution of e
- if all the resulting distribution are equals then e is independent of s

- Complete
- Exponential in the number of secret and random values

Checking probabilistic independence

Second solution, using simple rules:

- Rule 1: If e does not use s then it is independent

Checking probabilistic independence

Second solution, using simple rules:

- Rule 1: If e does not use s then it is independent
- Rule 2: If e can be written as $C[f \oplus r]$ and r does not occur in C and f then it is sufficient to test the independence of $C[r]$

The distribution of $f \oplus r$ is equal to the distribution of r

Checking probabilistic independence

Second solution, using simple rules:

- Rule 1: If e does not use s then it is independent
- Rule 2: If e can be written as $C[f \oplus r]$ and r does not occur in C and f then it is sufficient to test the independence of $C[r]$
- Rule 3: If Rules 1 and 2 do not apply then use the first solution (when possible)

Checking probabilistic independence

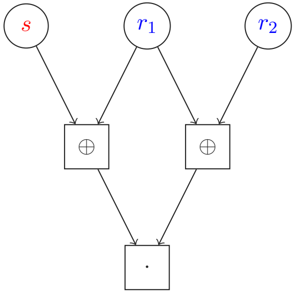
Second solution, using simple rules:

- Rule 1: If e does not use s then it is independent
- Rule 2: If e can be written as $C[f \oplus r]$ and r does not occur in C and f then it is sufficient to test the independence of $C[r]$
- Rule 3: If Rules 1 and 2 do not apply then use the first solution (when possible)

Problem: finding occurence of Rule 2 is relatively costly

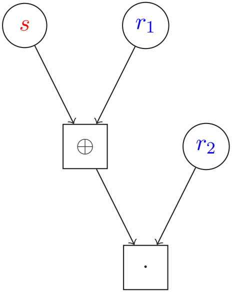
Independence: dag representation

$$(s \oplus r_1) \cdot (r_1 \oplus r_2)$$



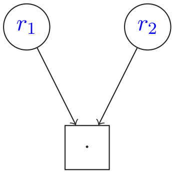
Independence: dag representation

$$(s \oplus r_1) \cdot r_2$$



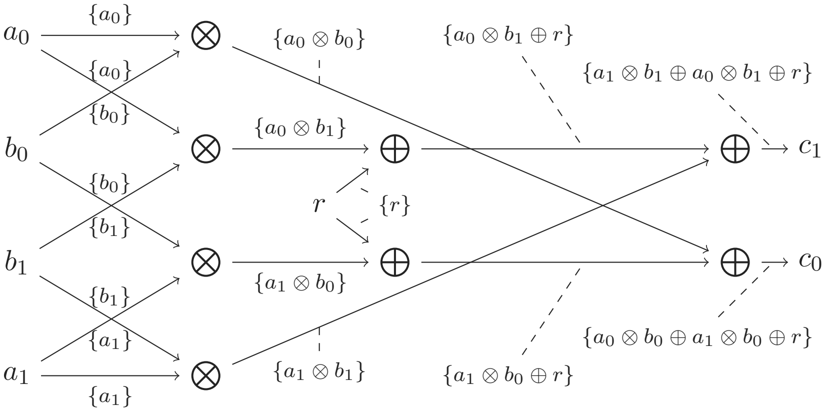
Independence: dag representation

$r_1 \cdot r_2$



Independent from the secret

First order Dom AND : NI



Extension to All Possible Sets

- Verification of first order masking is just a linear iteration of the previous algorithm (one call for each program point)
  100 checks for a program of 100 lines

Extension to All Possible Sets

- Verification of first order masking is just a linear iteration of the previous algorithm (one call for each program point)
  100 checks for a program of 100 lines
- For second order masking:
  forall pair of program point, the corresponding pair of expressions is independent from the secrets
  $4,950$ checks for a program of 100 lines

Extension to All Possible Sets

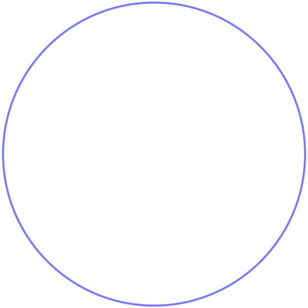
- Verification of first order masking is just a linear iteration of the previous algorithm (one call for each program point)
  100 checks for a program of 100 lines
- For second order masking:
  forall pair of program point, the corresponding pair of expressions is independent from the secrets
  $4,950$ checks for a program of 100 lines
- For t-order masking:
  forall t-tuple of program point, the corresponding t-tuple of expressions is independent from the secrets
  $\binom{N}{t}$ where N is the number number program points
  $3,921,225$ for a program of 100 lines and 4 observations

Extension to All Possible Sets

Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.

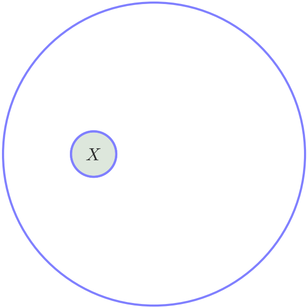
Extension to All Possible Sets

Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



Extension to All Possible Sets

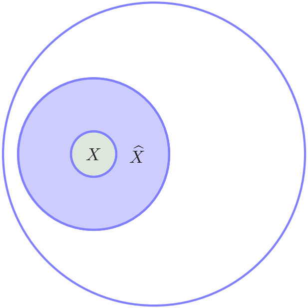
Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



1. select $X = (t$ variables$)$ and prove its independence

Extension to All Possible Sets

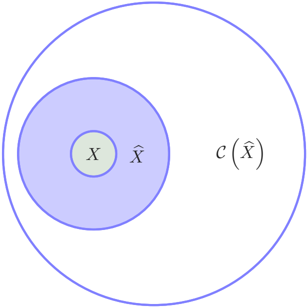
Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



1. select $X = (t$ variables$)$ and prove its independence
2. extend X to $\widehat{X}$ with more observations but still independence

Extension to All Possible Sets

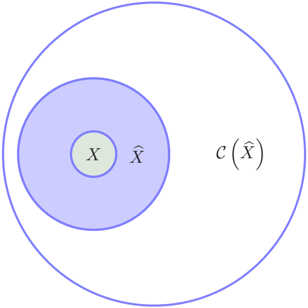
Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



1. select $X = (t$ variables$)$ and prove its independence
2. extend X to $\widehat{X}$ with more observations but still independence
3. recursively descend in set $\mathcal{C}(\widehat{X})$

Extension to All Possible Sets

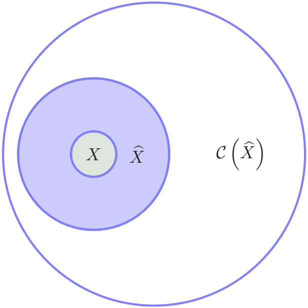
Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



1. select $X = (t$ variables$)$ and prove its independence
2. extend X to $\widehat{X}$ with more observations but still independence
3. recursively descend in set $\mathcal{C}(\widehat{X})$
4. merge $\widehat{X}$ and $\mathcal{C}(\widehat{X})$ once they are processed separately.

Extension to All Possible Sets

Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



1. select $X = (t$ variables) and prove its independence
2. extend X to $\widehat{X}$ with more observations but still independence
3. recursively descend in set $\mathcal{C}(\widehat{X})$
4. merge $\widehat{X}$ and $\mathcal{C}(\widehat{X})$ once they are processed separately.

Finding $\widehat{X}$ can be done very efficiently using a dag representation

Benchmark

It is working for relatively small programs:

Algorithm	Order	Tuples	Secure	Verification time
Refresh	9	2.10^{10}	✓	2s
Refresh	17	2.10^{20}	✓	3d
Refresh	18	4.10^{21}	✓	1 month

But there is a problem with large programs:

- Full AES implementation at order 1
- only 4 rounds of AES at order 2

Demo

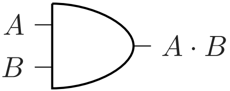
https://sites.google.com/view/maskverif/home

Demo `maskVerif`

Extending the model: glitches

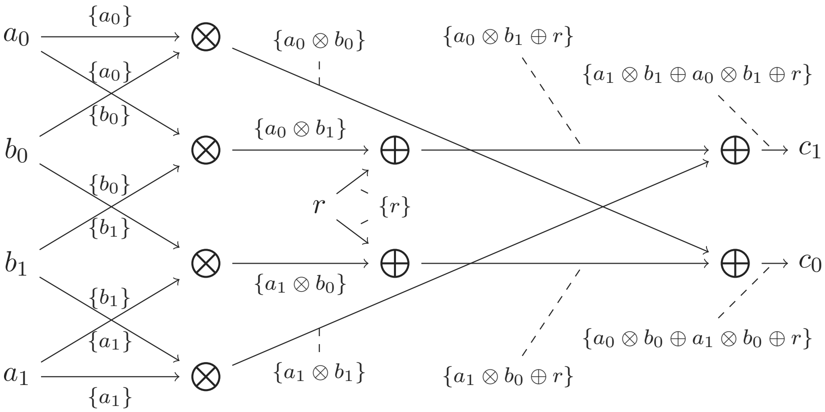
For hardware implementation a more realistic model need to take into account glitches

Example: AND gate $A \bigotimes B$

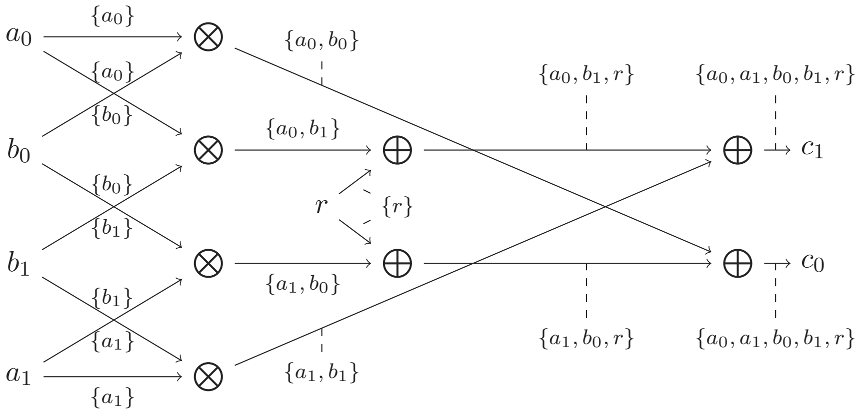


Possible leaks : $A \cdot B$, A, B

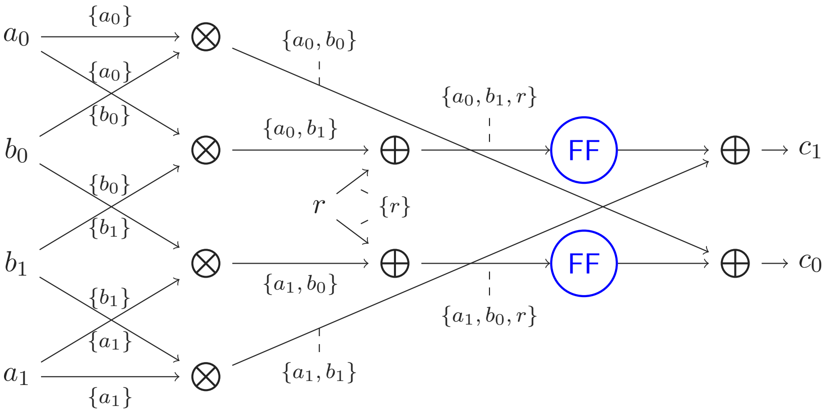
First order DOM AND : NI with glitches



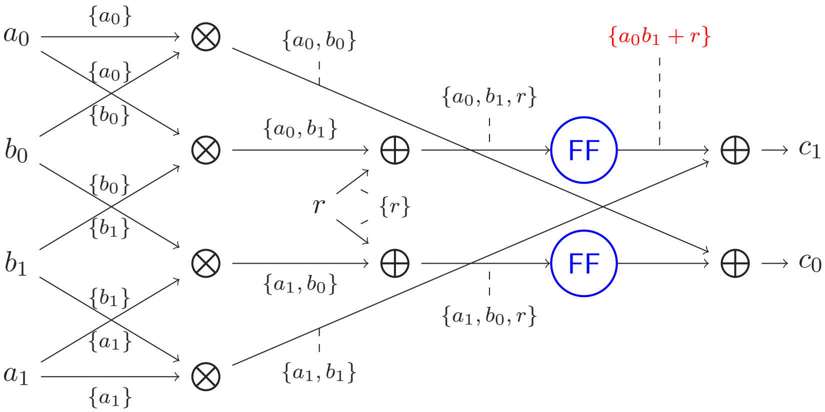
First order DOM AND : NI with glitches



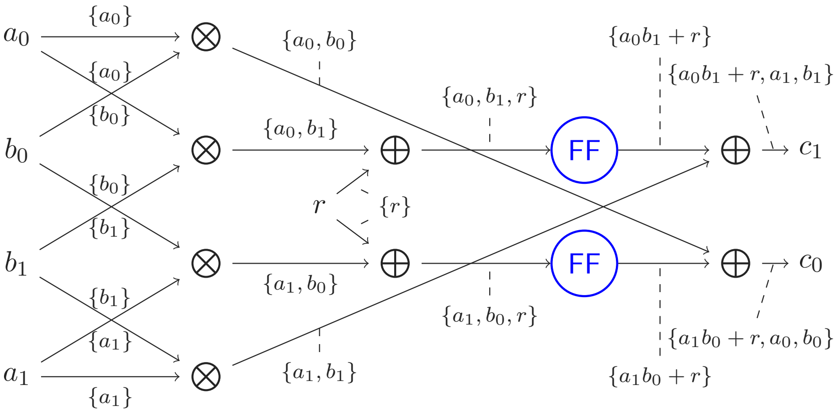
First order DOM AND : NI with glitches



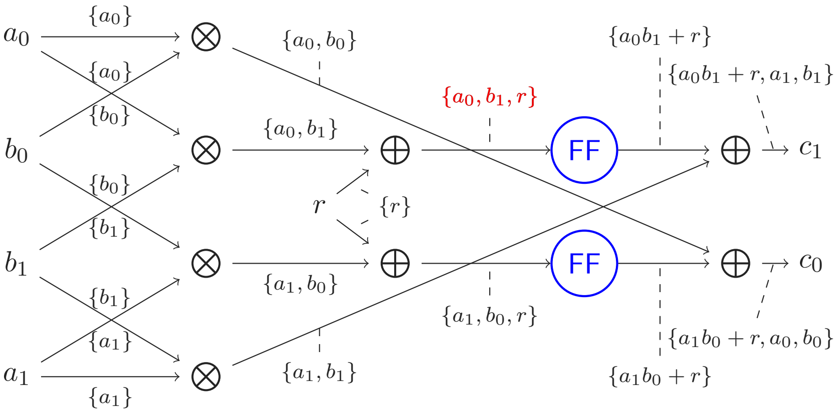
First order DOM AND : NI with glitches



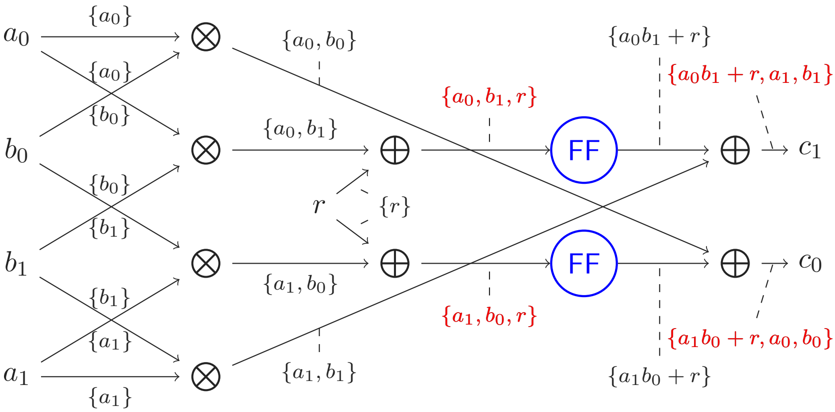
First order DOM AND : NI with glitches



First order DOM AND : NI with glitches



First order DOM AND : NI with glitches



Hardware implementation

We have extended `maskVerif` to

- take Verilog implementation as input
- take extra information on input shares (random, shares secret, public input)
- Check the security with or without glitches

Demo Hardware

https://sites.google.com/view/maskverif/home

yosys + `maskVerif`

Examples (provided by Bloem et al)

Algo	# obs		probing	
	wG	woG	wG	woG
first-order verification				
Trichina AND	2	13	0.01s ✗	0.01s ✗
ISW AND	1	13	0.01s ✗	0.01s
DOM AND	4	13	0.01s	0.01s
DOM Keccak S-box	20	76	0.01s	0.01s
DOM AES S-box	96	571	2.3s	0.4s
second-order verification				
DOM Keccak S-box	60	165	0.02s	0.02s
third-order verification				
DOM Keccak S-box	100	290	0.28s	0.25s
fourth-order verification				
DOM Keccak S-box	150	450	11s	14s
fifth-order verification				
DOM Keccak S-box	210	618	9m44s	18m39s

1. Side-Channel Attacks and Masking

2. Formal Tools for Verification at Fixed Order

3. Formal Tools for Verification of Generic Implementations

Probing Model

Require: Encoding $[x]$
Ensure: Fresh encoding $[x]$

for $i = 1$ to t **do**
  $r \leftarrow \$$
  $x_0 \leftarrow x_0 + r$
  $x_i \leftarrow x_i + r$
end for
return $[x]$

Probing Model

Require: Encoding $[x]$
Ensure: Fresh encoding $[x]$

```
for i = 1 to t do
    r ← $
    x_0 ← x_0 + r
    x_i ← x_i + r
end for
return [x]
```

Simulation-based proof:

- show that any set of t variables can be simulated with at most t input shares x_i
- any set of t shares x_i is independent from x

Probing Model

Require: Encoding $[x]$
Ensure: Fresh encoding $[x]$

```
for i = 1 to t do
    r ← $
    x_0 ← x_0 + r
    x_i ← x_i + r
end for
return [x]
```

Simulation-based proof:

- show that any set of t variables can be simulated with at most t input shares x_i
- any set of t shares x_i is independent from x
- exactly relies on the notion of *non interference* (NI)

And then?

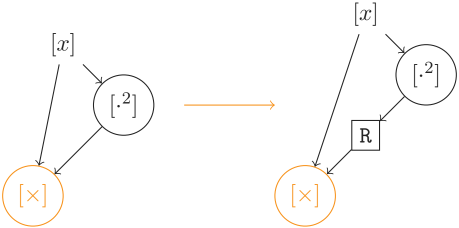
once done for small gadgets, how to extend it?

Until Recently

- composition probing secure for $2t+1$ shares
- no solution for $t+1$ shares

First Proposal

- Rivain and Prouff (CHES 2010): add refresh gadgets (NI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

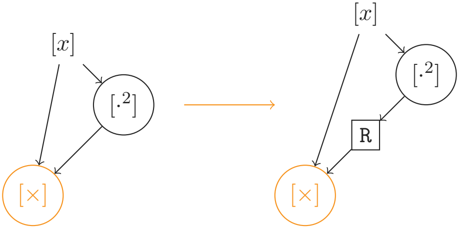


Require: Encoding $[x]$
Ensure: Fresh encoding $[x]$

    for $i = 1$ to t **do**
        $r \leftarrow \$$
        $x_0 \leftarrow x_0 + r$
        $x_i \leftarrow x_i + r$
    end for
    return $[x]$

First Proposal

- Rivain and Prouff (CHES 2010): add refresh gadgets (NI)
- Example: AES S-box on $\mathrm{GF}(2^8)$



Require: Encoding $[x]$
Ensure: Fresh encoding $[x]$

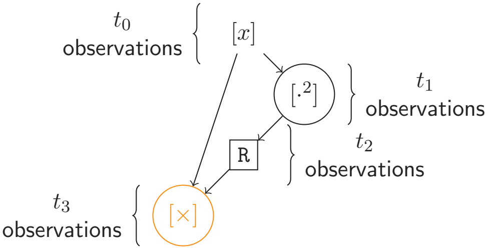
for $i = 1$ to t **do**
  $r \leftarrow \$$
  $x_0 \leftarrow x_0 + r$
  $x_i \leftarrow x_i + r$
end for
return $[x]$

$\Rightarrow$ Flaw from $t = 2$ (FSE 2013: Coron, Prouff, Rivain, and Roche)

Why This Flaw?

- Rivain and Prouff (CHES 2010): add refresh gadgets (NI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

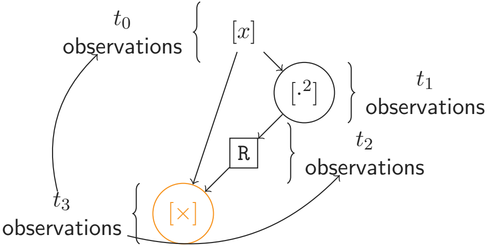
Constraint:
$t_0 + t_1 + t_2 + t_3 \leqslant t$



Why This Flaw?

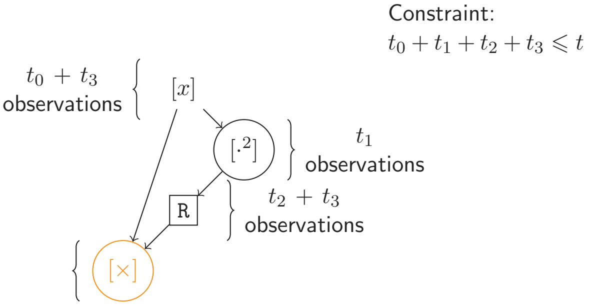
- Rivain and Prouff (CHES 2010): add refresh gadgets (NI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

Constraint:
$t_0 + t_1 + t_2 + t_3 \leqslant t$



Why This Flaw?

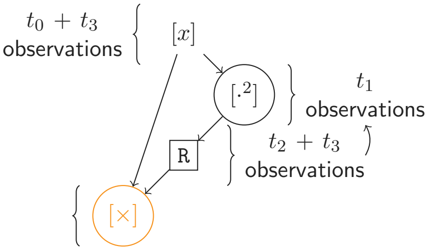
- Rivain and Prouff (CHES 2010): add refresh gadgets (NI)
- Example: AES S-box on $\mathrm{GF}(2^8)$



Why This Flaw?

- Rivain and Prouff (CHES 2010): add refresh gadgets (NI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

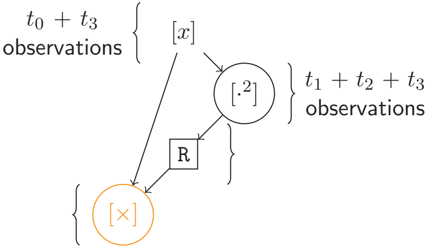
Constraint:
$t_0 + t_1 + t_2 + t_3 \leqslant t$



Why This Flaw?

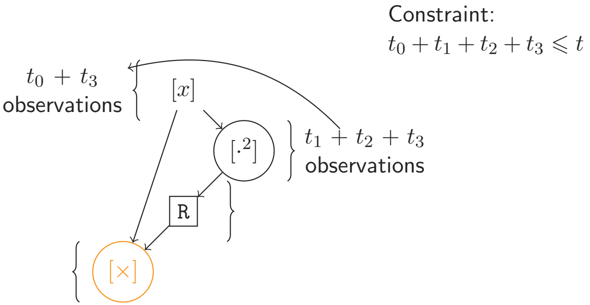
- Rivain and Prouff (CHES 2010): add refresh gadgets (NI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

Constraint:
$t_0 + t_1 + t_2 + t_3 \leqslant t$



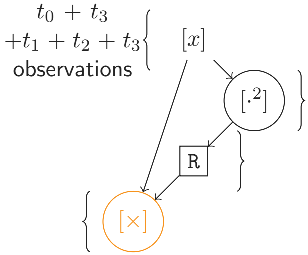
Why This Flaw?

- Rivain and Prouff (CHES 2010): add refresh gadgets (NI)
- Example: AES S-box on $\mathrm{GF}(2^8)$



Why This Flaw?

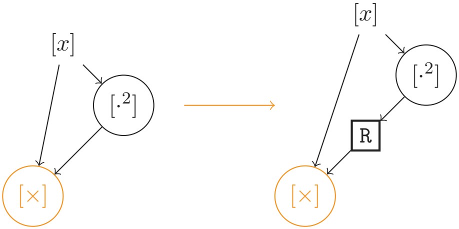
- Rivain and Prouff (CHES 2010): add refresh gadgets (NI)
- Example: AES S-box on $\mathrm{GF}(2^8)$



Constraint:
$t_0 + t_1 + t_2 + t_3 \leqslant t$

Second Proposal

- Barthe, B., Dupressoir, Fouque, Grégoire, Strub, Zucchini (CCS 2016): add stronger refresh gadgets (SNI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

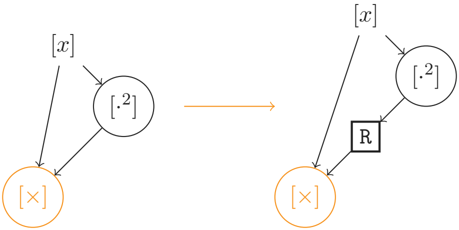


Require: Encoding $[x]$
Ensure: Fresh encoding $[x]$

  for $i = 0$ to t **do**
    for $j = i + 1$ to t **do**
      $r \leftarrow \$$
      $x_i \leftarrow x_i + r$
      $x_j \leftarrow x_j + r$
    end for
  end for
  return $[x]$

Second Proposal

- Barthe, B., Dupressoir, Fouque, Grégoire, Strub, Zucchini (CCS 2016): add stronger refresh gadgets (SNI)
- Example: AES S-box on $\mathrm{GF}(2^8)$



Require: Encoding $[x]$
Ensure: Fresh encoding $[x]$

> **for** $i = 0$ to t **do**
>   **for** $j = i + 1$ to t **do**
> $r \leftarrow \$$
> $x_i \leftarrow x_i + r$
> $x_j \leftarrow x_j + r$
>   **end for**
> **end for**
> **return** $[x]$

$\Rightarrow$ Formal security proof for any order t

Strong Non-Interference (SNI)

- t-SNI $\Rightarrow$ t-NI $\Rightarrow$ t-probing secure
- a circuit is t-SNI iff any set of t intermediate variables, whose t_1 on the internal variables and t_2 and the outputs, can be perfectly simulated with at most t_1 shares of each input

require x_0 and x_1 to be perfectly simulated $\Rightarrow$ not 3-SNI since y_0 is an output variable

function Ex-t3(x_0, x_1, x_2, x_3, c):

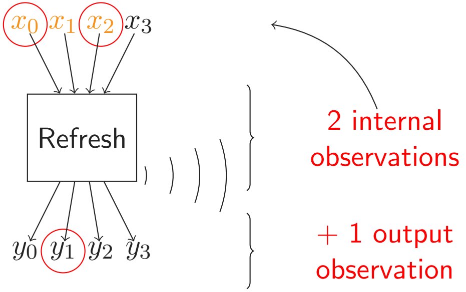
(* $x_0, x_1, x_2 = \$$ *)
(* $x_3 = x + x_0 + x_1 + x_2$ *)

$$
\begin{aligned}
r_0 &\leftarrow \$ \\
r_1 &\leftarrow \$ \\
y_0 &\leftarrow x_0 + r_0 \\
y_1 &\leftarrow x_3 + r_1 \\
t_1 &\leftarrow x_1 + r_0 \\
t_2 &\leftarrow (x_1 + r_0) + x_2 \\
y_2 &\leftarrow (x_1 + r_0 + x_2) + r_1 \\
y_3 &\leftarrow c + r_1
\end{aligned}
$$

return(y_0, y_1, y_2, y_3)

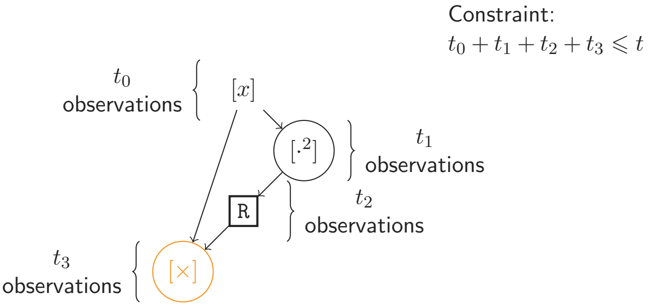
Strong Non-Interference (SNI)

- t-SNI $\Rightarrow$ t-NI $\Rightarrow$ t-probing secure
- a circuit is t-SNI iff any set of t intermediate variables, whose t_1 on the internal variables and t_2 and the outputs, can be perfectly simulated with at most t_1 shares of each input



Why Does It Works?

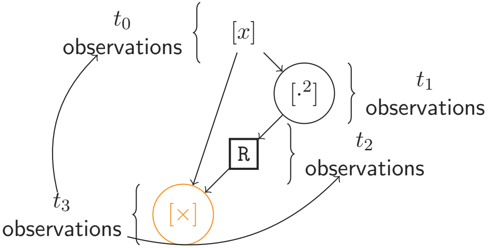
- Barthe, B., Dupressoir, Fouque, Grégoire, Strub, Zucchini (CCS 2016): add stronger refresh gadgets (SNI)
- Example: AES S-box on $\mathrm{GF}(2^8)$



Why Does It Works?

- Barthe, B., Dupressoir, Fouque, Grégoire, Strub, Zucchini (CCS 2016): add stronger refresh gadgets (SNI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

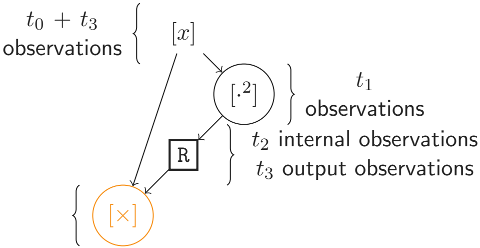
Constraint:
$t_0 + t_1 + t_2 + t_3 \leqslant t$



Why Does It Works?

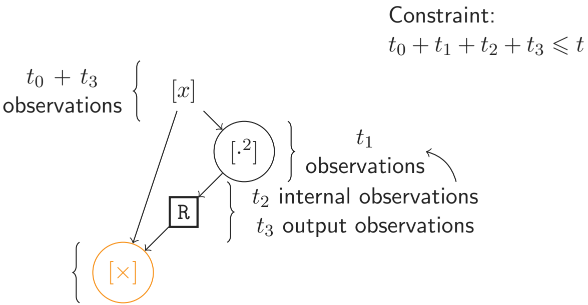
- Barthe, B., Dupressoir, Fouque, Grégoire, Strub, Zucchini (CCS 2016): add stronger refresh gadgets (SNI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

Constraint:
$t_0 + t_1 + t_2 + t_3 \leqslant t$



Why Does It Works?

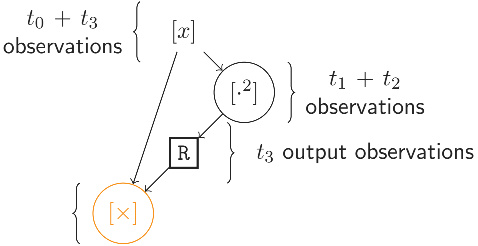
- Barthe, B., Dupressoir, Fouque, Grégoire, Strub, Zucchini (CCS 2016): add stronger refresh gadgets (SNI)
- Example: AES S-box on $\mathrm{GF}(2^8)$



Why Does It Works?

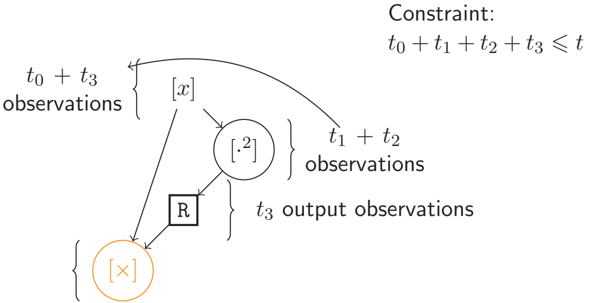
- Barthe, B., Dupressoir, Fouque, Grégoire, Strub, Zucchini (CCS 2016): add stronger refresh gadgets (SNI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

Constraint:
$t_0 + t_1 + t_2 + t_3 \leqslant t$



Why Does It Works?

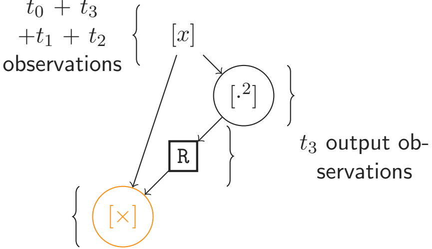
- Barthe, B., Dupressoir, Fouque, Grégoire, Strub, Zucchini (CCS 2016): add stronger refresh gadgets (SNI)
- Example: AES S-box on $\mathrm{GF}(2^8)$



Why Does It Works?

- Barthe, B., Dupressoir, Fouque, Grégoire, Strub, Zucchini (CCS 2016): add stronger refresh gadgets (SNI)
- Example: AES S-box on $\mathrm{GF}(2^8)$

Constraint:
$t_0 + t_1 + t_2 + t_3 \leqslant t$



Tool `maskComp`

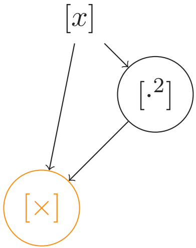
- from t-NI and t-SNI gadgets $\Rightarrow$ build a t-NI circuit by inserting t-SNI refresh gadgets at carefully chosen locations
- formally proven

Implementation in C language with no countermeasure $\longrightarrow$ `maskComp` $\longrightarrow$ t-NI secure implementation in C language

Gilles Barthe and Sonia Belaïd and François Dupressoir and Pierre-Alain Fouque and Benjamin Grégoire and Pierre-Yves Strub *Strong Non-Interference and Type-Directed Higher-Order Masking and Rebecca Zucchini,* ACM CCS 2016, Proceedings, 116–129.

Demo AES S-box without refresh

https://sites.google.com/site/maskingcompiler/home



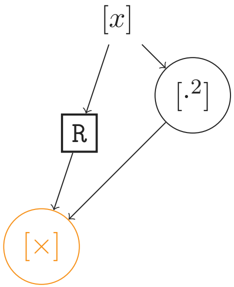

```
bint8_t x3(bint8_t x) {
  bint8_t r, z;
  z = gf256_pow2(x);
  r = gf256_mul(x,z);
  return r;
}
```

```
Start type checking of x3
insert refresh 1 1
x3 : {S_34 } ->
     O_21
     side
     constraints LE:S_34 <= I_35
                 NEEDED:[ {O_21 }]
1 refresh inserted in x3.
1 refresh inserted.
```

> ./maskcomp.native −o myoutput_masked.c x3.c

Demo AES S-box with refresh

https://sites.google.com/site/maskingcompiler/home




```
bint8_t x3(bint8_t x) {
  bint8_t r, w, z;
  z = gf256_pow2(x);
  w = bint8_refresh(x);
  r = gf256_mul(w,z);
  return r;
}
```

```
Start type checking of x3
x3 : {S_29 } ->
     O_21
     side
     constraints LE:S_29 <= I_30
                 NEEDED:[ {O_21 }]
0 refresh inserted.
```

> ./maskcomp.native – o myoutput_masked.c x3.c

Demo full AES

https://sites.google.com/site/maskingcompiler/home

> ./maskcomp.native -o myoutput_masked.c aes.c

Limitations of maskComp

- maskComp adds a refresh gadget to Circuit 1
- but Circuit 1 was already t-probing secure

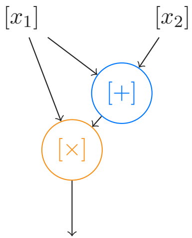


Figure: Circuit 1.

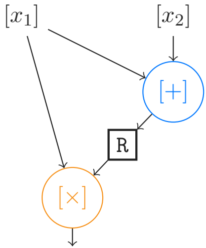
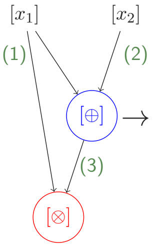


Figure: Circuit 1 after maskComp.

Tool `tightPROVE`

- Joint work with Dahmun Goudarzi and Matthieu Rivain to appear in Asiacrypt 2018
- Apply to tight shared circuits:
  - sharewise additions,
  - ISW-multiplications,
  - ISW-refresh gadgets
- Determine exactly whether a tight shared circuit is probing secure for any order t
  1. Reduction to a simplified problem
  2. Resolution of the simplified problem
  3. Extension to larger circuits

Demo tightPROVE 1



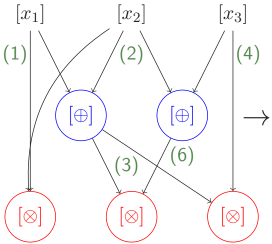

```
in 0
in 1
xor 0 1
and 0 2
out 3
```

$\longrightarrow$

```
-------------------------
list_comb =  [1, 3]
-------------------------
comb =  1
=> NO ATTACK (G2 = G1)
   G: [[(1,3)], []]
   O: [[3], []]
-------------------------
-------------------------
comb =  3
=> NO ATTACK (G2 = G1)
   G: [[(1,3)], []]
   O: [[1], []]
-------------------------
No attack found
```

```
> sage verif.sage example1.circuit
```

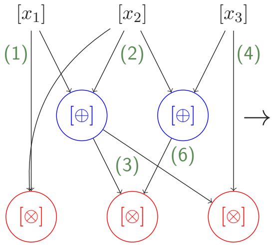
Demo tightPROVE 2




```
in 0
in 1
in 2
xor 0 1
xor 1 2
and 0 1
and 3 4
and 2 3
out 5
out 6
out 7
```

```
> sage verif.sage example2.circuit
```

Demo tightPROVE 2



→

```
in 0
in 1
in 2
xor 0 1
xor 1 2
and 0 1
and 3 4
and 2 3
out 5
out 6
out 7
```

→

```
--------------------------------
list_comb =  [1, 3, 2, 4, 6]
--------------------------------
comb =  1
=> NO ATTACK (G3 = G2)
   G: [[(1,2)], [(3,6),(3,4)],
    []]
   O: [[2], [6, 4], []]
--------------------------------
--------------------------------
comb =  3
=> NO ATTACK (G3 = G2)
   G: [[(3,6),(3,4)], [(1,2)],
    []]
   O: [[6, 4], [2], []]
--------------------------------
--------------------------------
comb =  2
=> ATTACK
   G: [[(1,2)], [(3,6),(3,4)]]
   O: [[1], [6, 4]]
--------------------------------
Attack found: 2 in span [1, 6,
   4]
```

```
> sage verif.sage example2.circuit
```

Conclusion

In a nutshell...

- Formal tools to verify security of masked implementations
- Trade-off between security and performances

To continue...

- Achieve better performances
- Apply such formal verifications to every circuit