Vérification d'implémentations masquées: du software au hardware

Collaborators: G. Barthe, S. Belaid, F. Dupressoir, S. Faust, P.A. Fouque, K. Papagiannopoulos, P. Schwabe, F.X. Standaert, K. Stoffelen, P.Y. Strub, R. Zucchini

Benjamin Grégoire

1. Side Channel Attacks

2. Masking

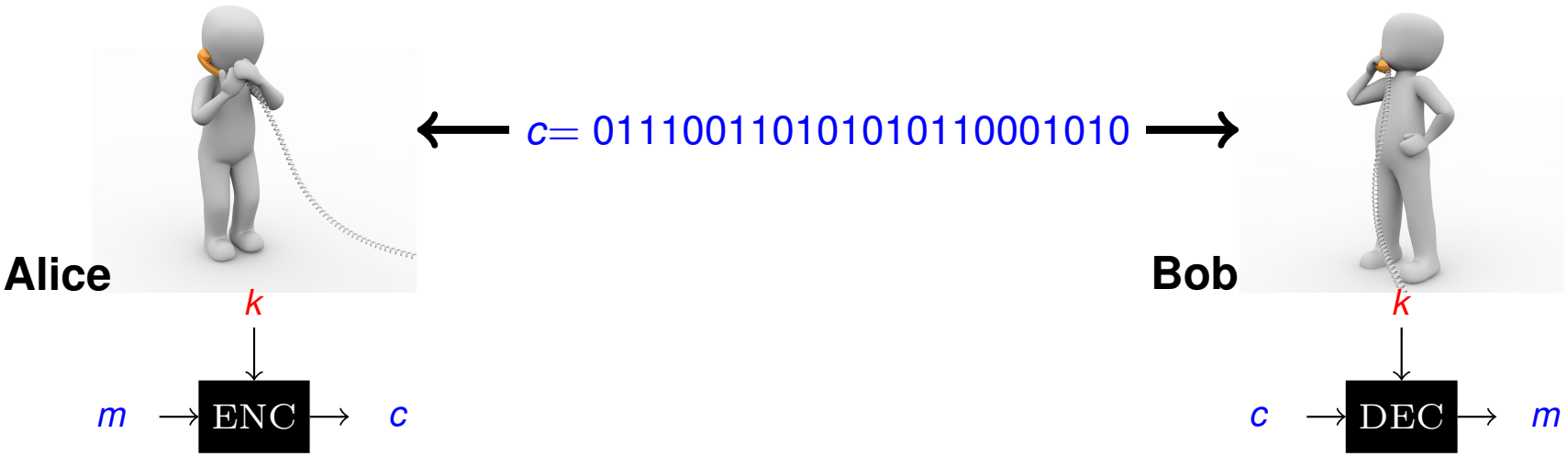
3. Formal Tools

1

Side Channel Attacks

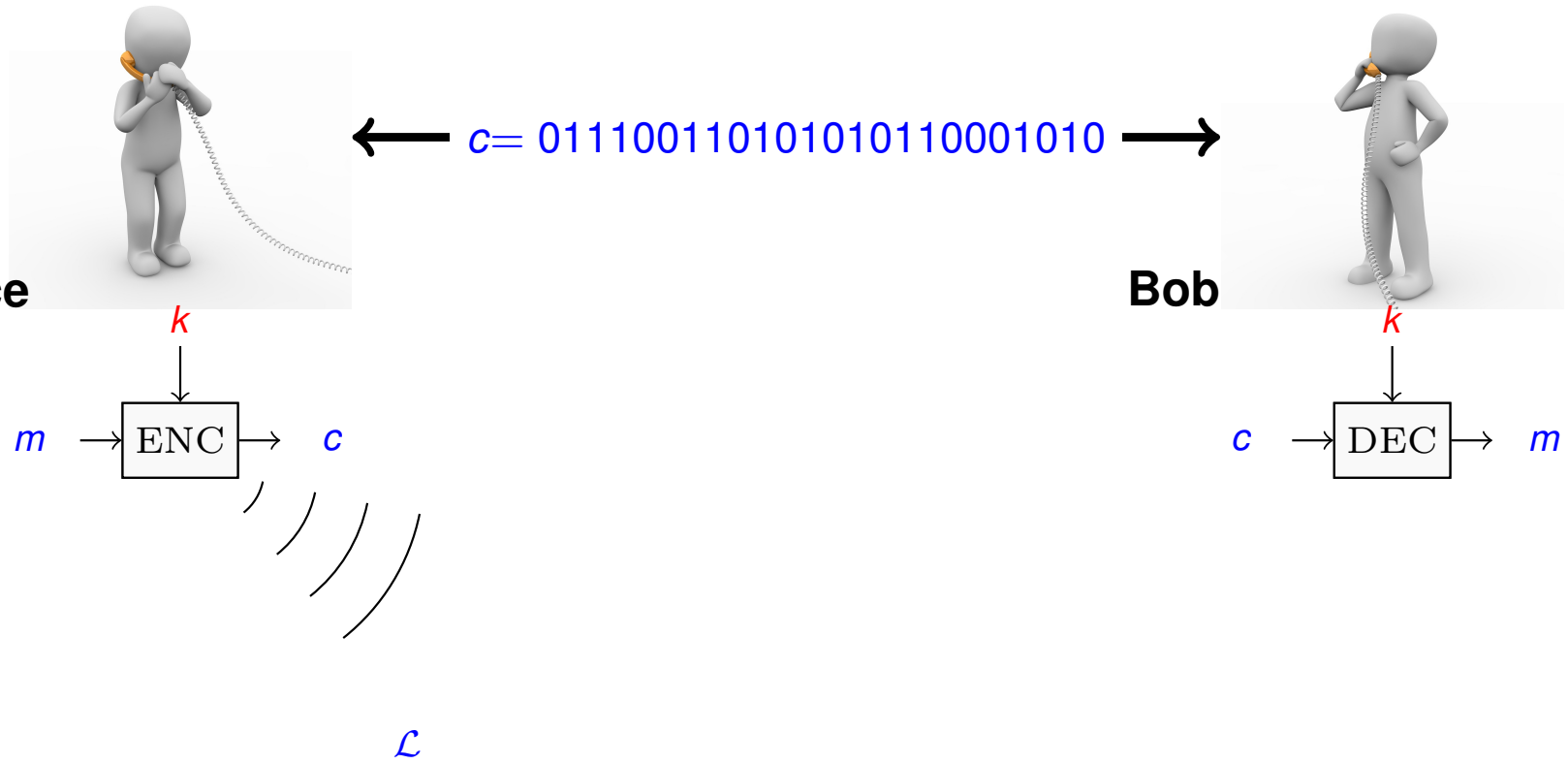
Cryptanalysis

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



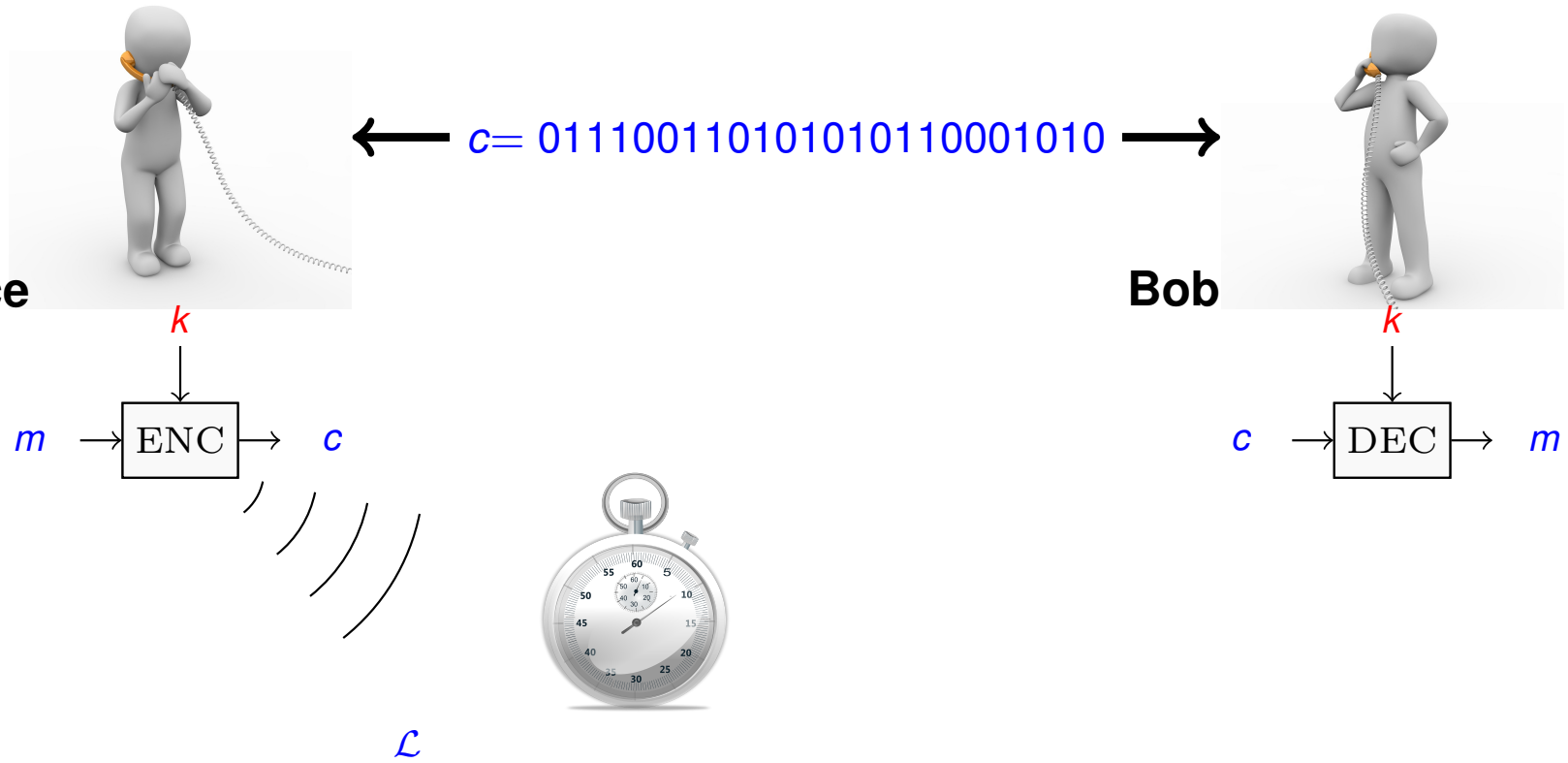
Cryptanalysis

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



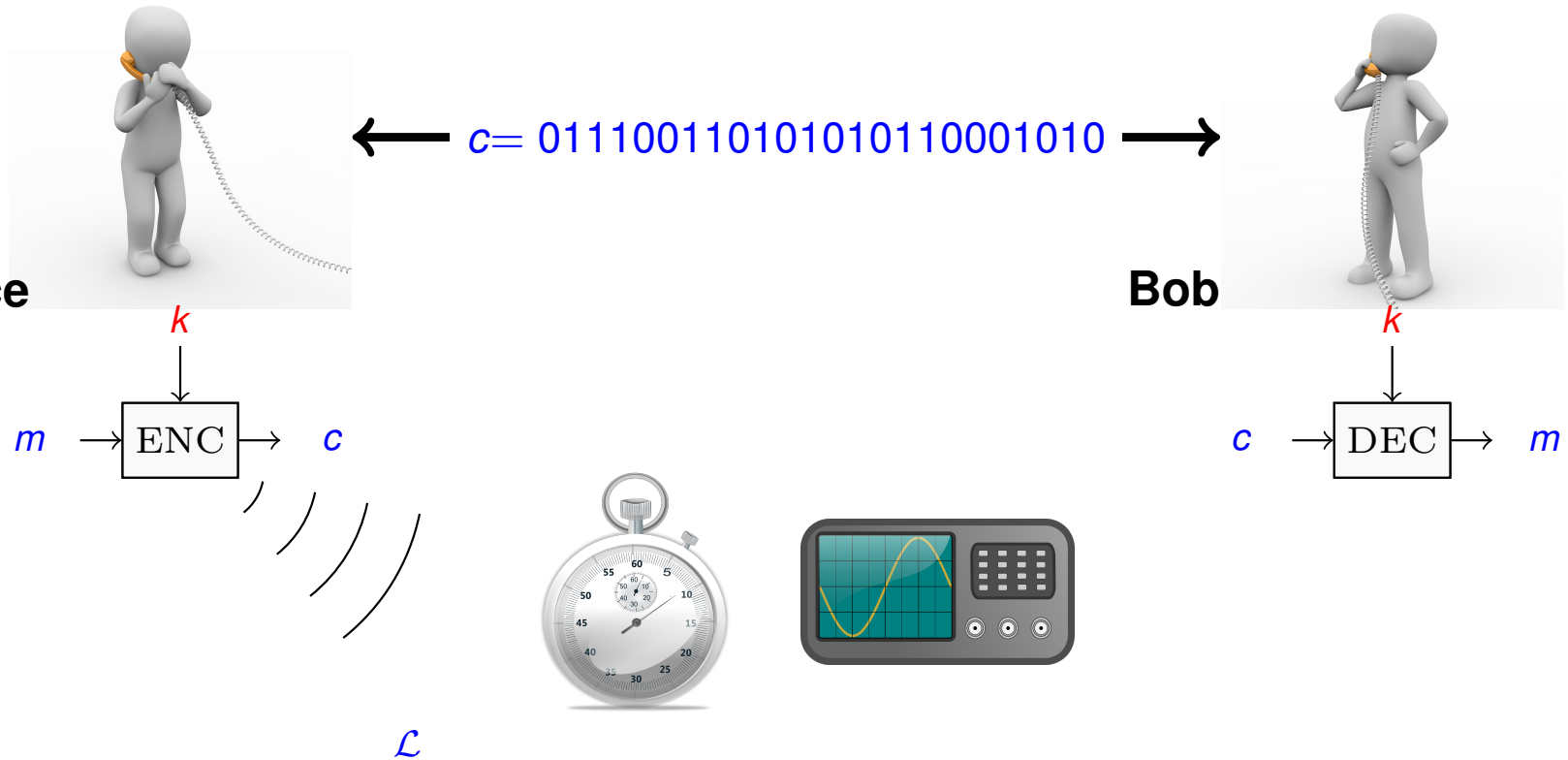
Cryptanalysis

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



Cryptanalysis

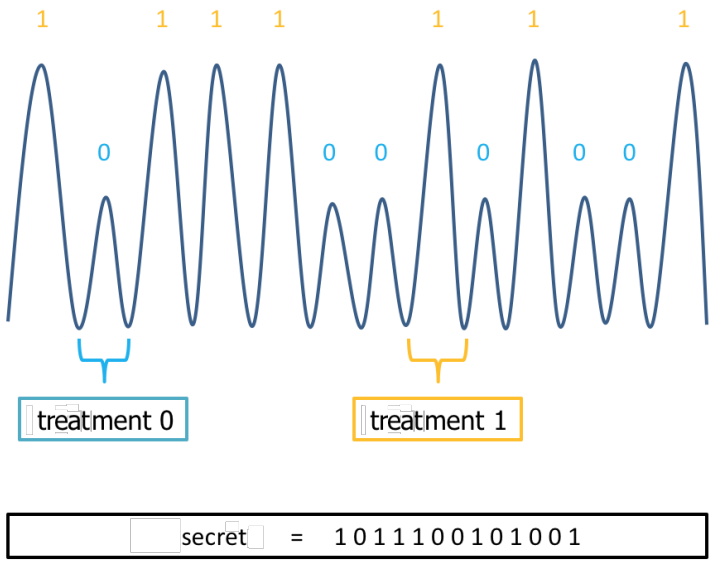
- Black-box cryptanalysis
- Side-Channel Analysis: $\mathcal{A} \leftarrow (m, c, \mathcal{L})$



Example of Simple Power Analysis (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
```



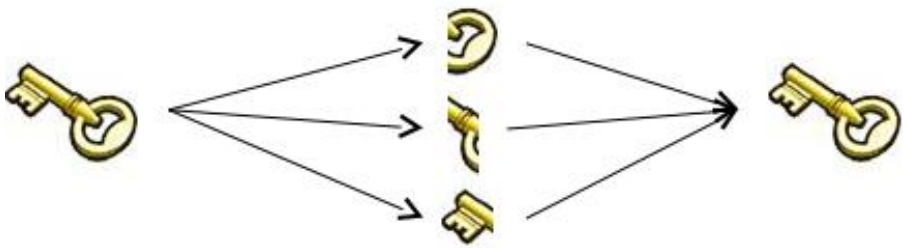
- SPA: one single trace to recover the secret key
- Differential Power Analysis (DPA): several traces to recover the secret key

2

Masking

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 $x = x_0 \oplus \ldots \oplus x_t$



- 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*

Masked Implementations (first order examples)

- a secret s is initially shared into two shares

$$s_0, s_1 = s \oplus s_0$$

  where s_0 is a random value
- Linear functions: apply the function to each share

$$v \oplus w \rightarrow (v_0 \oplus w_0, v_1 \oplus w_1)$$

- Non-linear functions: much more complex $v \cdot w = c$

$$\begin{aligned} v \cdot w &= (v_0 \oplus v_1) \cdot (w_0 \oplus w_1) \\ &= (v_0 \cdot w_0 \oplus v_0 \cdot w_1) \oplus (v_1 \cdot w_0 \oplus v_1 \cdot w_1) \\ &= c_0 \oplus c_1 \end{aligned}$$

Leakage Models

Noisy leakage model by Chari, Jutla, Rao, and Rohatgi (Crypto 1999) then Rivain and Prouff (EuroCrypt 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

Realistic

Reduction by Duc, Dziembowski, and Faust (EuroCrypt 2014)

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 (initially perfectly shared)

Verification

back to non-linear functions

Shared multiplication (and): $v \cdot w$

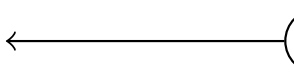
$$
\begin{aligned}
t_0 &\leftarrow v_0 \cdot w_0 \\
t_1 &\leftarrow v_0 \cdot w_1 \\
t_2 &\leftarrow v_1 \cdot w_1 \\
t_3 &\leftarrow v_1 \cdot w_0 \\
c_0 &\leftarrow t_0 \oplus t_1 \\
c_1 &\leftarrow t_2 \oplus t_3
\end{aligned}
$$

back to non-linear functions

Shared multiplication (and): $v \cdot w$

$$t_0 \leftarrow v_0 \cdot w_0$$
$$t_1 \leftarrow v_0 \cdot w_1$$
$$t_2 \leftarrow v_1 \cdot w_1$$
$$t_3 \leftarrow v_1 \cdot w_0$$
$$c_0 \leftarrow t_0 \oplus t_1$$
$$c_1 \leftarrow t_2 \oplus t_3$$

independent from the secret?



back to non-linear functions

Shared multiplication (and): $v \cdot w$

$$t_0 \leftarrow v_0 \cdot w_0$$
$$t_1 \leftarrow v_0 \cdot w_1$$
$$t_2 \leftarrow v_1 \cdot w_1$$
$$t_3 \leftarrow v_1 \cdot w_0$$

independent from the secret? ⟵ $c_0 \leftarrow t_0 \oplus t_1$

$$c_1 \leftarrow t_2 \oplus t_3$$

$= v_0 \cdot w_0 \oplus v_0 \cdot w_1 = v_0 \cdot (w_0 \oplus w_1) = v_0 \cdot w$

Need more randomness

back to non-linear functions

Shared multiplication (and): $v \cdot w$

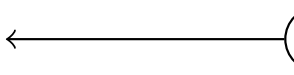
$$
\begin{aligned}
t_0 &\leftarrow v_0 \cdot w_0 \\
t_1 &\leftarrow v_0 \cdot w_1 \\
t_2 &\leftarrow v_1 \cdot w_1 \\
t_3 &\leftarrow v_1 \cdot w_0 \\
r &\leftarrow \$ \\
t_4 &\leftarrow t_1 \oplus r \\
c_0 &\leftarrow t_0 \oplus t_4 \\
t_5 &\leftarrow t_3 \oplus r \\
c_1 &\leftarrow t_2 \oplus t_5
\end{aligned}
$$

back to non-linear functions

Shared multiplication (and): $v \cdot w$

$$
\begin{aligned}
t_0 &\leftarrow v_0 \cdot w_0 \\
t_1 &\leftarrow v_0 \cdot w_1 \\
t_2 &\leftarrow v_1 \cdot w_1 \\
t_3 &\leftarrow v_1 \cdot w_0 \\
r &\leftarrow \$ \\
t_4 &\leftarrow t_1 \oplus r \\
c_0 &\leftarrow t_0 \oplus t_4 \qquad = v_0 \cdot w_0 \oplus (v_0 \cdot w_1 \oplus r) \approx v_0 \cdot w_0 \oplus r \approx r \\
t_5 &\leftarrow t_3 \oplus r \\
c_1 &\leftarrow t_2 \oplus t_5
\end{aligned}
$$

independent from the secret



Difficulty

- First order: How to test independence from the secret
- High order : How to deal with all possible sets (binomial coefficient complexity).

3

Formal Tools

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
  - `maskVerif`: G. Barthe, S. Belaid, F. Dupressoir, P.A. Fouque, B. Grégoire, P.Y. Strub
  - `CheckMasks`: J.S. Coron
  - Bloen et al.'s tool
  - C. Wang, P. Schaumont

Checking probabilistic independence from a secret s

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 from a secret s

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

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

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

Checking probabilistic independence from a secret *s*

Second solution, using simple rules:

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

Checking probabilistic independence from a secret s

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 from a secret s

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 from a secret s

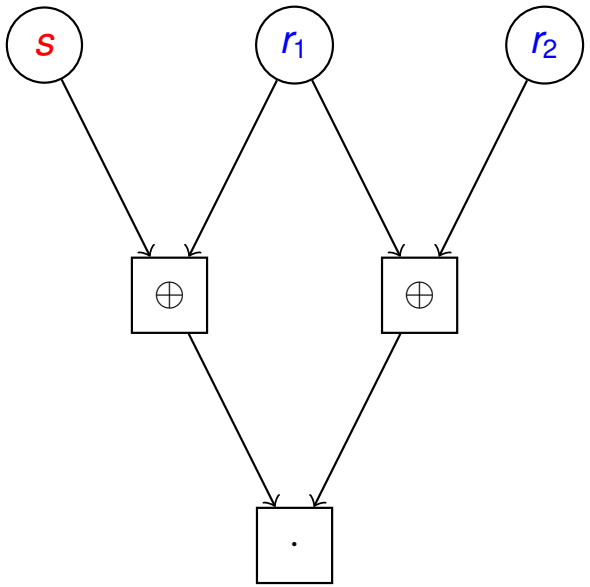
Second solution, using simple rules:

- 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

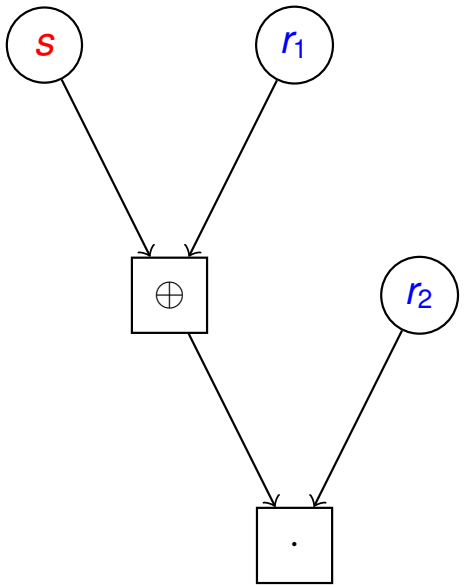
Independence from the secret: dag representation

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



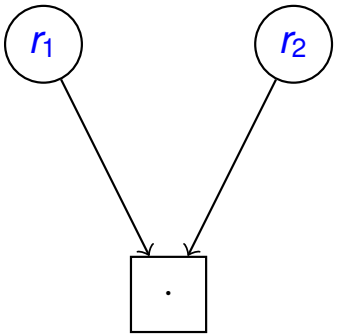
Independence from the secret: dag representation

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



Independence from the secret: dag representation

$$r_1 \cdot r_2$$



Independent from the secret

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

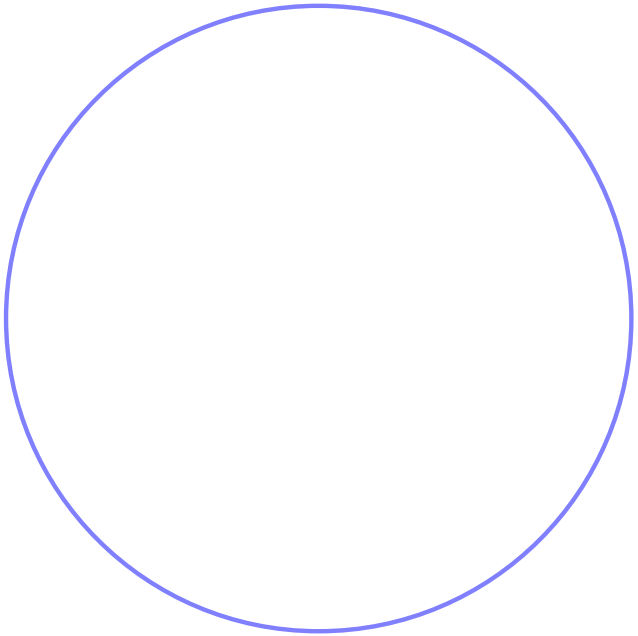
- Verification of first order masking is just a linear iteration of the previous algorithm (one call for each program point)
  100 checks for a program of 100 lines
- For second order masking:
  forall pair of program point, the corresponding pair of expressions is independent from the secrets
  $4,950$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

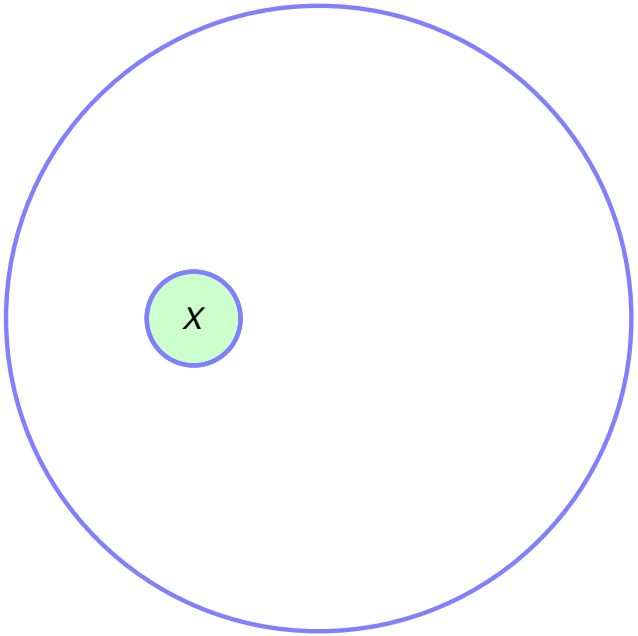
Extension to All Possible Sets

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

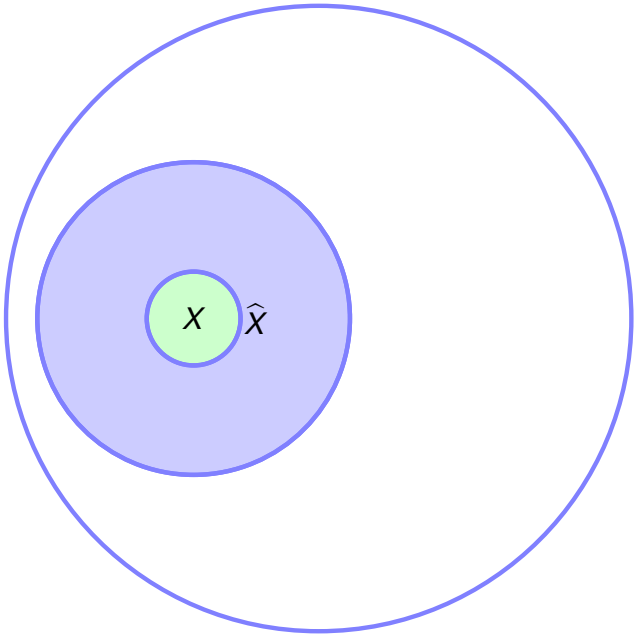
Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



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

Extension to All Possible Sets

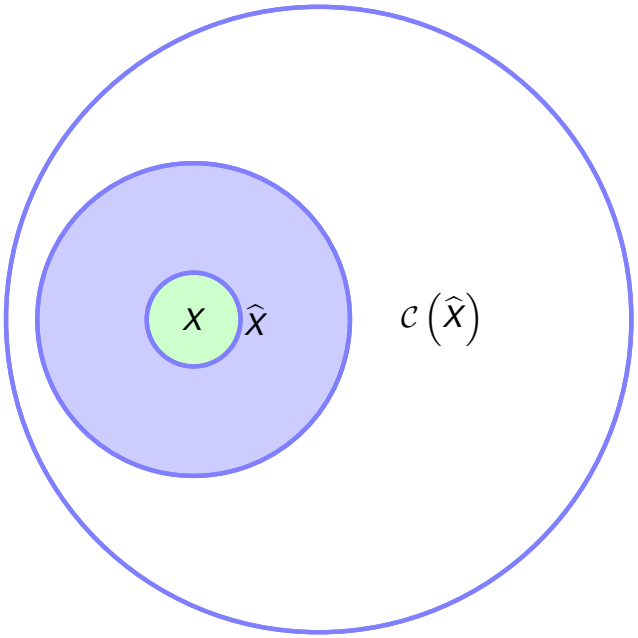
Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



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

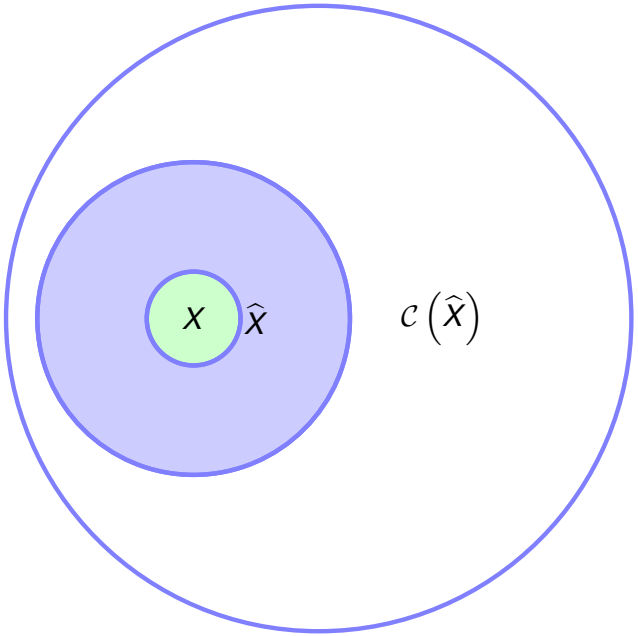
Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



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

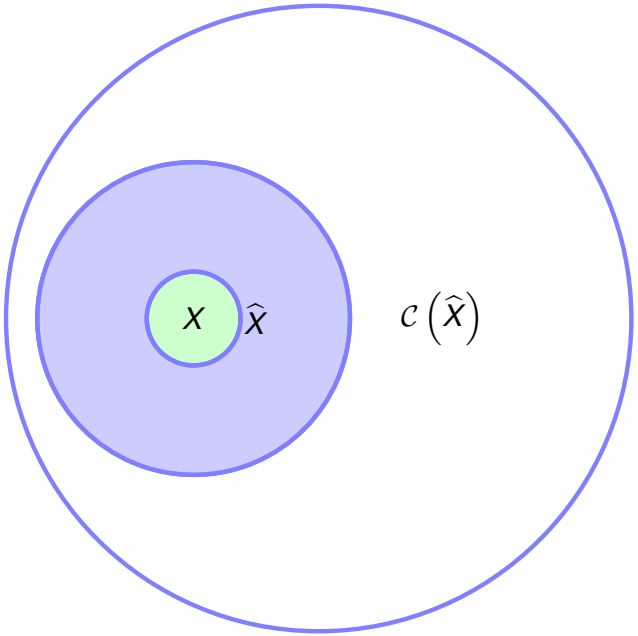
Idea: if $e_1, \ldots, e_p$ is independent from the secrets then all subtuples are independent from the secrets.



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

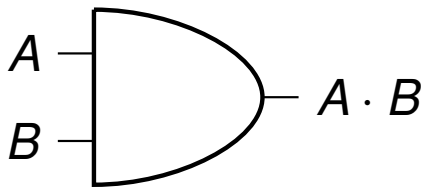
But there is a problem with large programs:

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

Extending the model: glitches

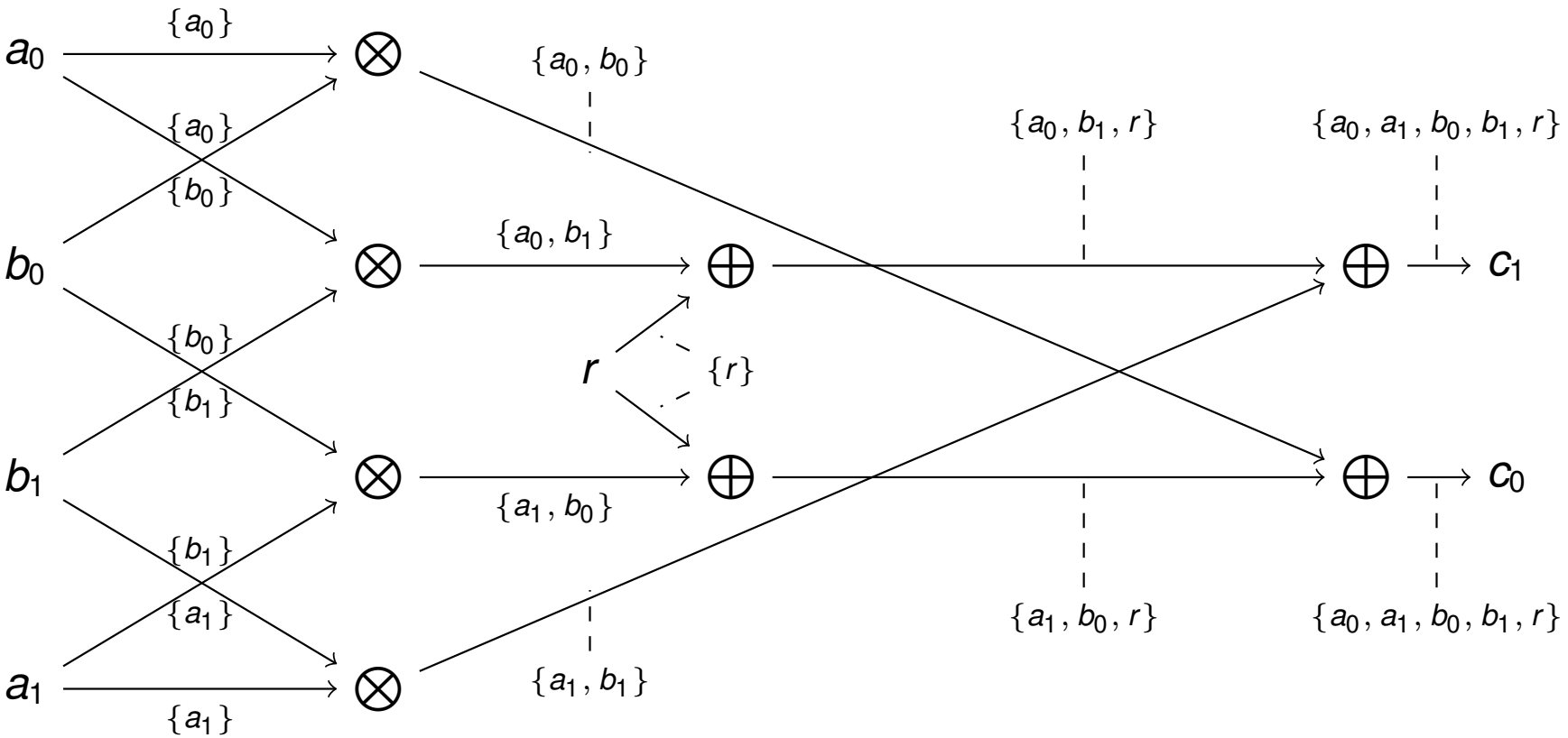
For hardware implementation a more realistic model need to take into account glitches

Example: AND gate $A \otimes B$

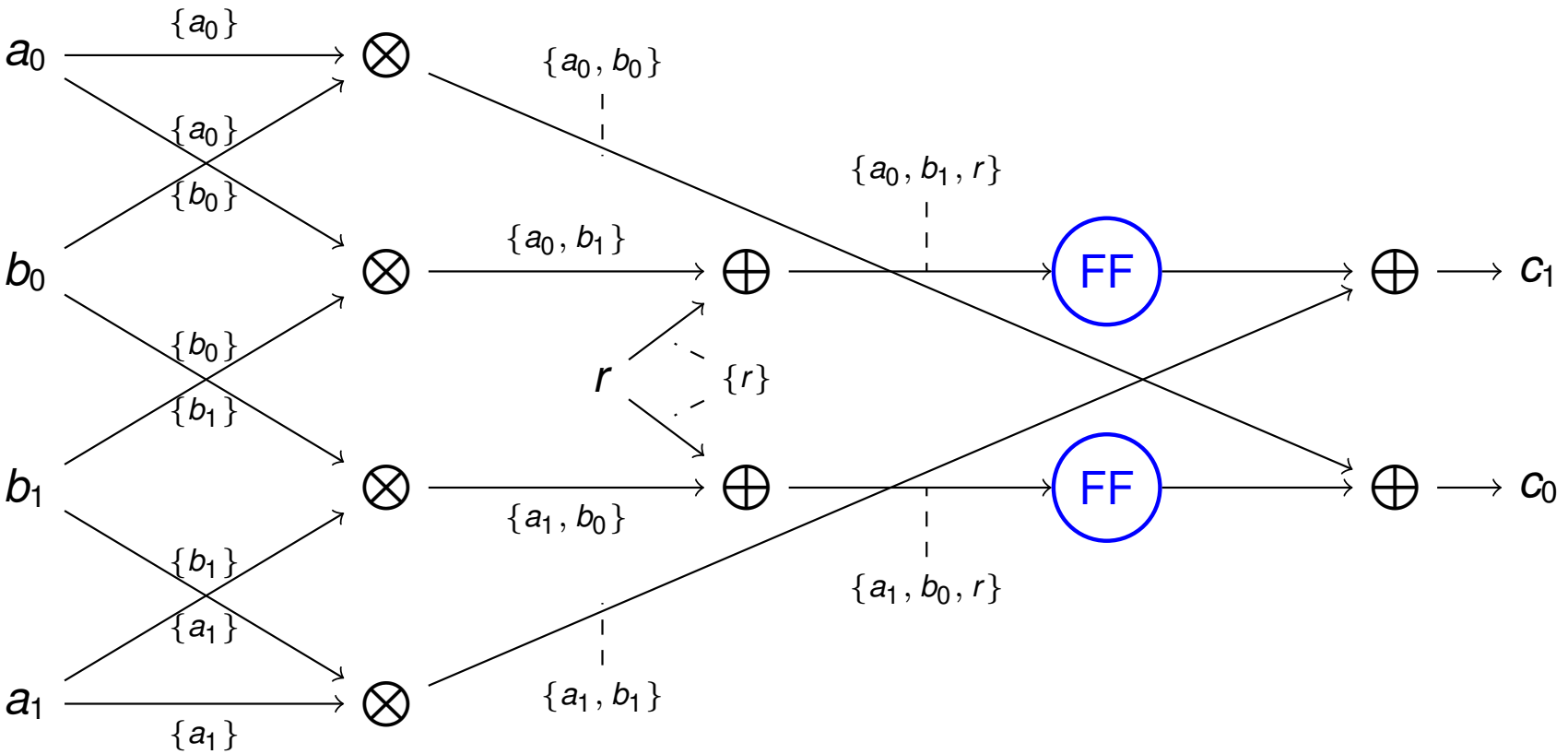


Possible leaks : $A \cdot B$, A, B

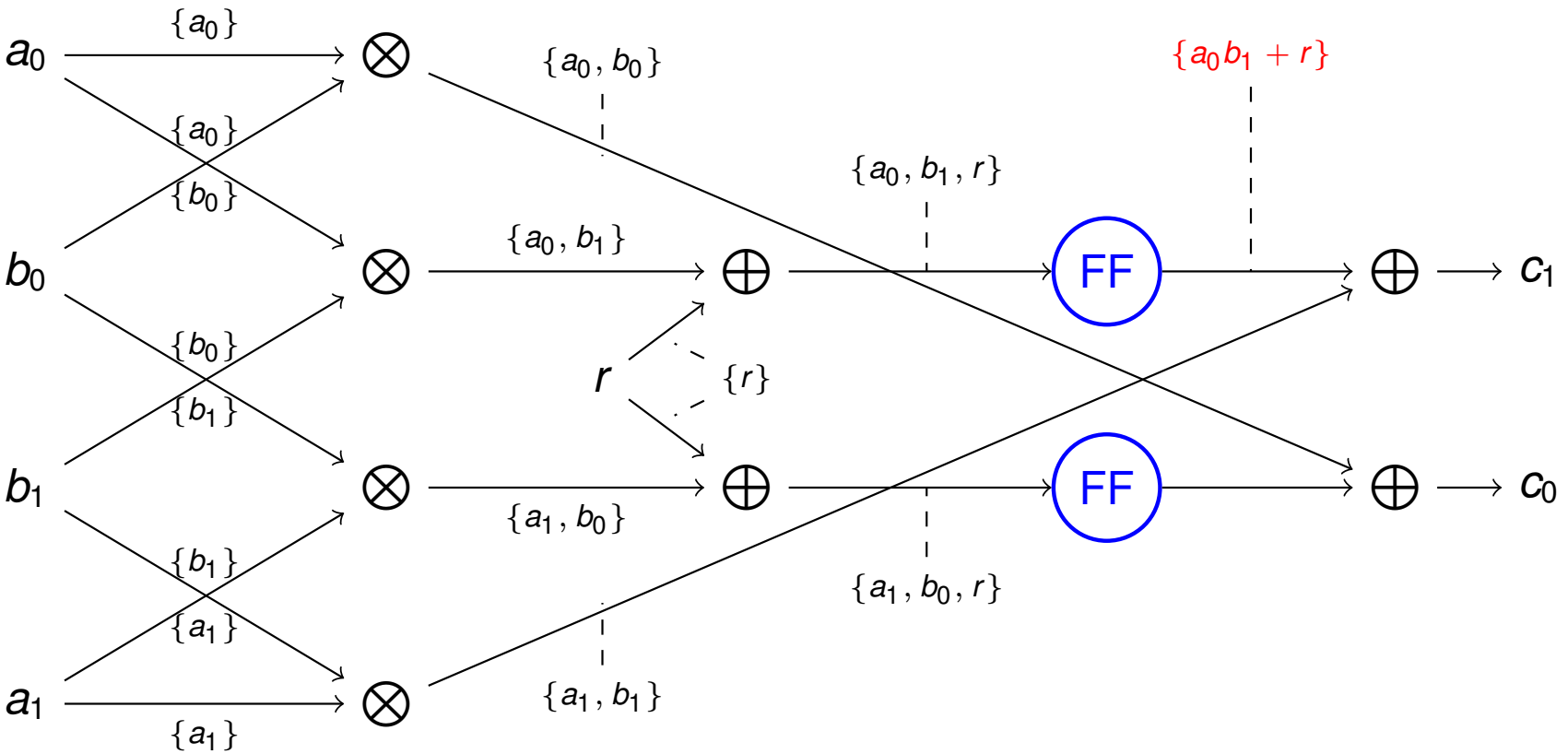
First order masked implementation of an AND gate



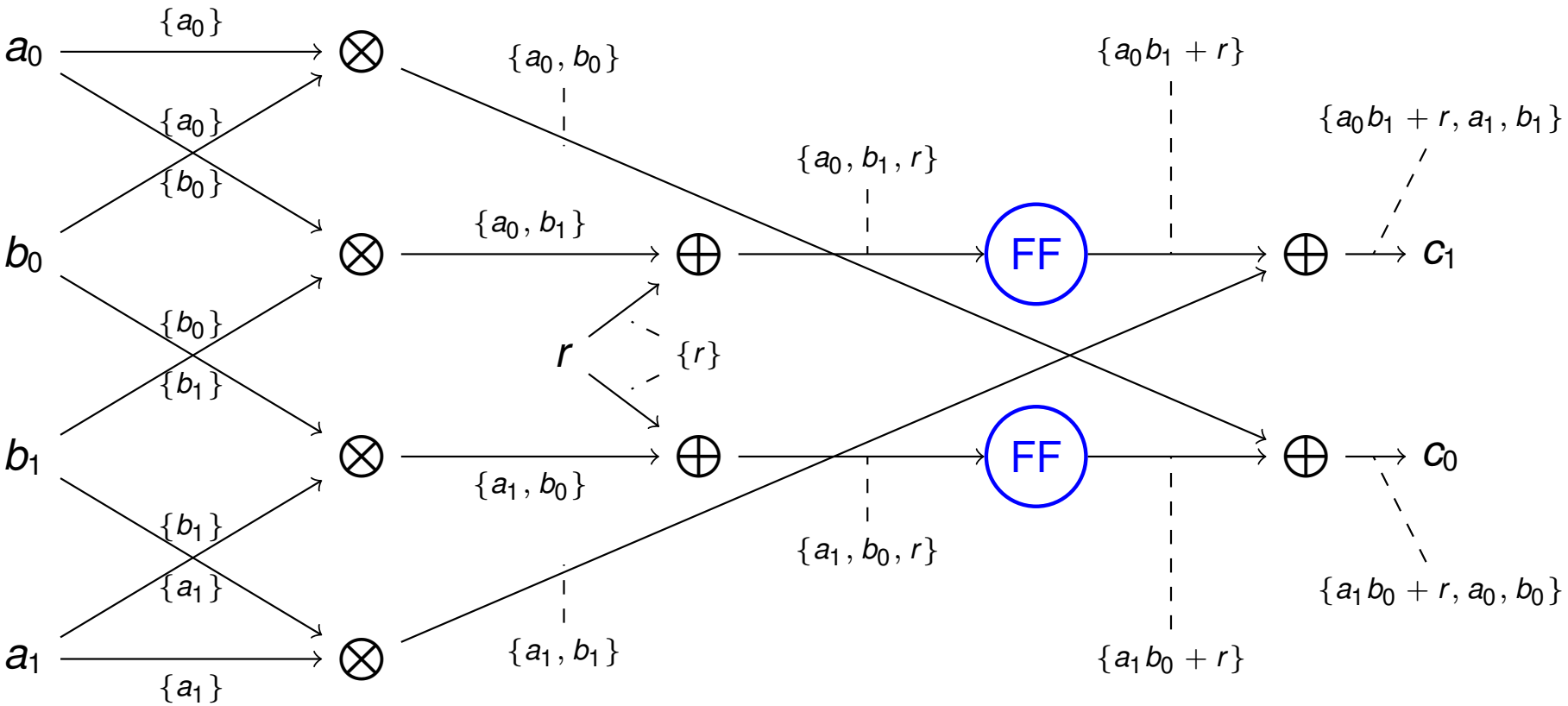
First order masked implementation of an AND gate



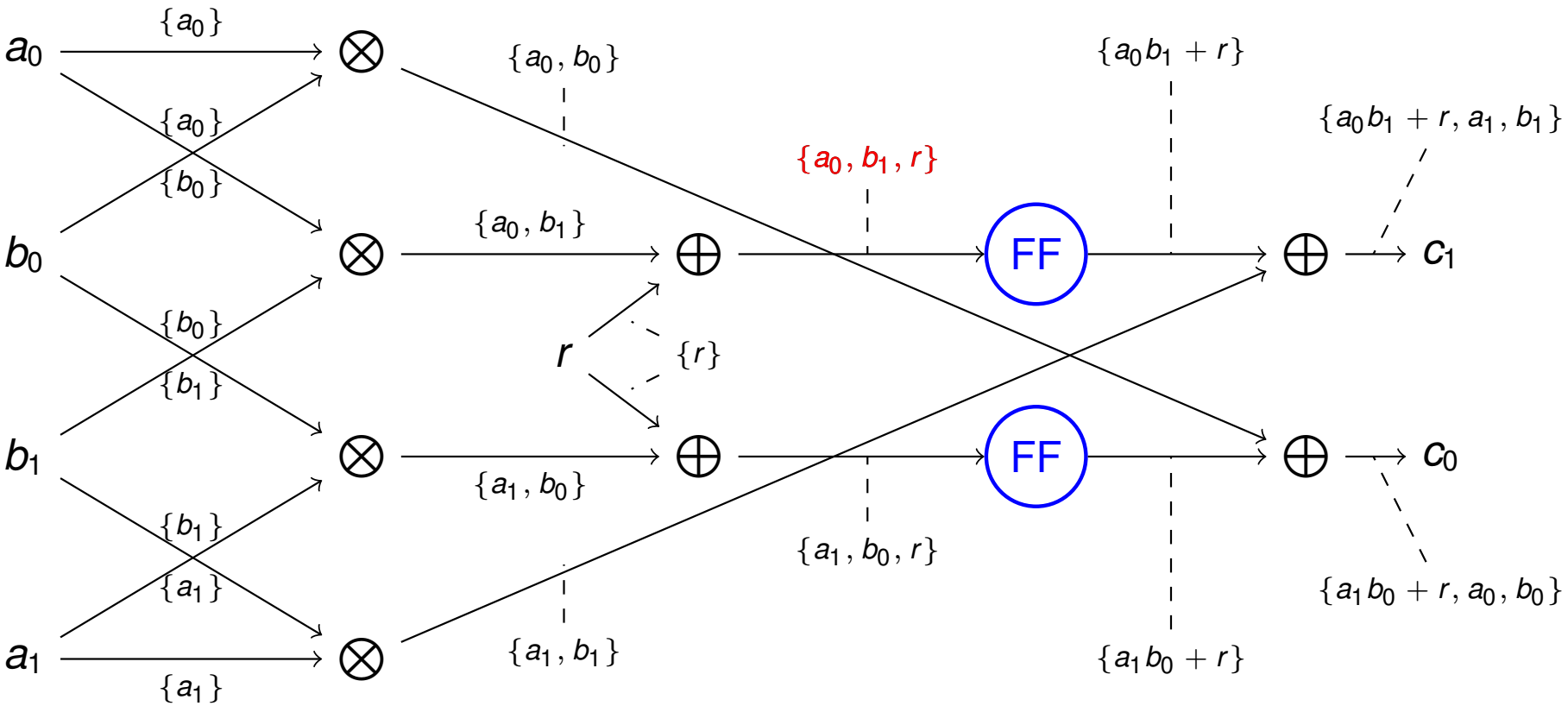
First order masked implementation of an AND gate



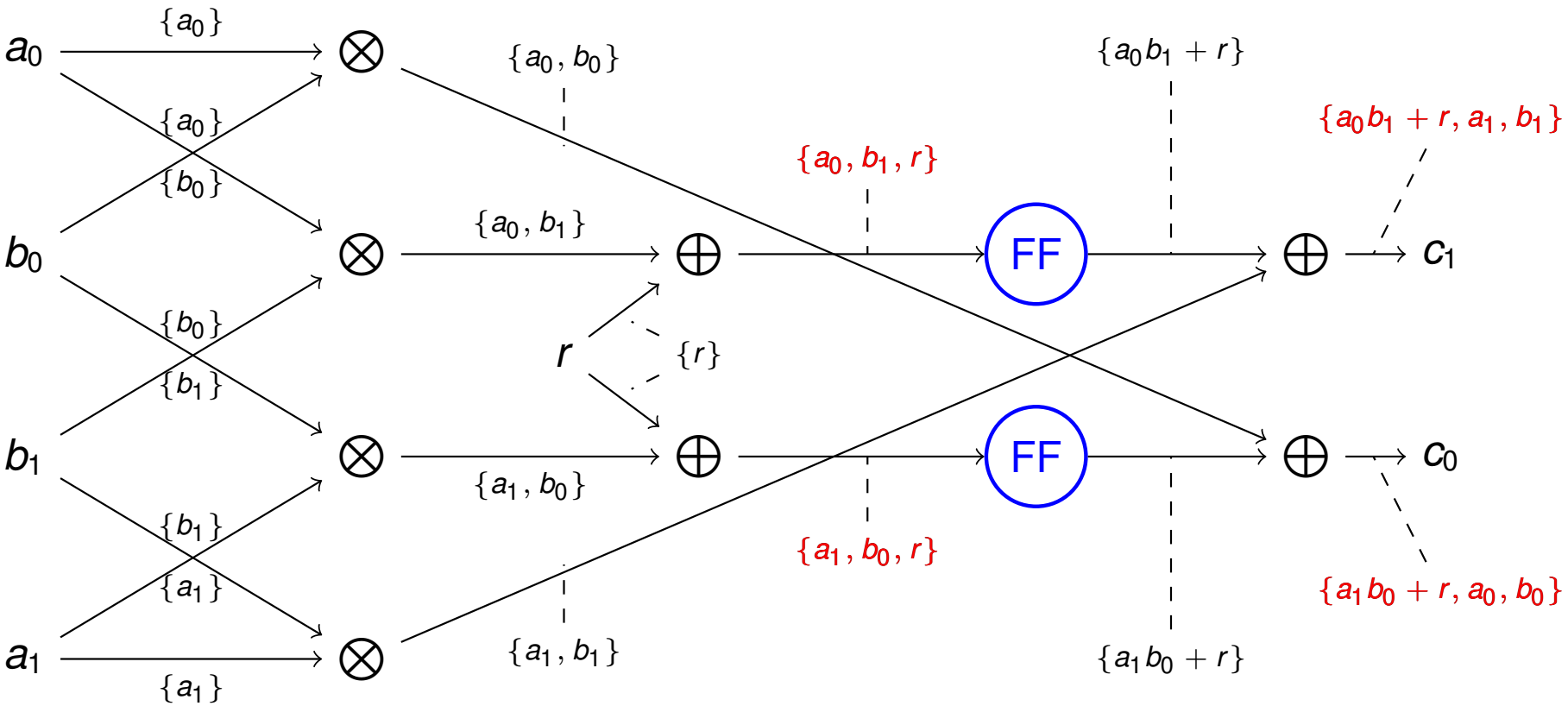
First order masked implementation of an AND gate



First order masked implementation of an AND gate



First order masked implementation of an AND gate



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

Benchmark (provided by Bloen et al)

Algo	# obs		probing		probing (Bloen et al)	
	wG	woG	wG	woG	wG	woG
first-order verification						
Trichina AND	2	13	0.01s ✗	0.01s ✗	$\leq$ 2s ✗	$\leq$ 1s ✗
ISW AND	1	13	0.01s ✗	0.01s	$\leq$2s ✗	$\leq$1s
DOM AND	4	13	0.01s	0.01s	$\leq$2s	$\leq$1s
DOM Keccak S-box	20	76	0.01s	0.01s	$\leq$20s	$\leq$1s
DOM AES S-box	96	571	2.3s	0.4s	$\leq$5-10h*	$\leq$30s*
second-order verification						
DOM Keccak S-box	60	165	0.02s	0.02s	$\leq$40s*	$\leq$10s*
third-order verification						
DOM Keccak S-box	100	290	0.28s	0.25s	$\leq$25m*	$\leq$4m*
fourth-order verification						
DOM Keccak S-box	150	450	11s	14s	_	_
fifth-order verification						
DOM Keccak S-box	210	618	9m44s	18m39s	_	_

Composition

Composition of secure t-probing secure is hard:

- Problem: security is proved assuming an initial perfect sharing of the input

Until recently:

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

Solution:

- Use refreshing (CHESS 2010: Rivain and Prouff)
- Flow for $2 \leq t$ (FSE 2013: Coron, Prouff, Rivain, and Roche)
- Need more powerful refreshing (CCS 2016)
- New security notions: t-SNI $\Rightarrow$ t-NI $\Rightarrow$ t-probing secure
- Enable composition

Tool maskComp

- Take as input a C unmasked implementation with annotation indicating which variables are secrets
- Generate a C masked implementation (by replacing unmasked operations by there corresponding masked implementation)
- Check that the generated implementation is secure (add Refresh when needed)

Used to generate various version of cryptographic schemes: AES, Keccak, Pride, Simon, Speck

Conclusion

- New notion of security (t-NI, t-SNI), enable composition (CCS 2016)
- Efficient implementation of Refresh and Multiplication (less randomness, parallel implementation) (EuroCrypt 2017)
- AES, Vectorizing Higher-Order Masking (COSADE 2018)
- `maskVerif` tool (t-probing, t-NI, t-SNI, w/wo glitches) (EuroCrypt 2015)
- `maskComp` compiler

Future work

- Complete method for independence
- Extend `maskComp` to hardware implementation
- More industrial versions of our tools