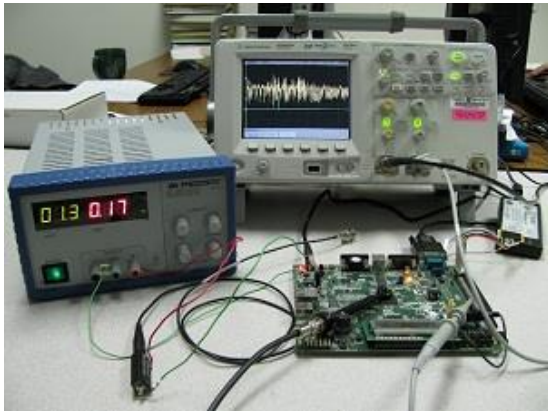
Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model

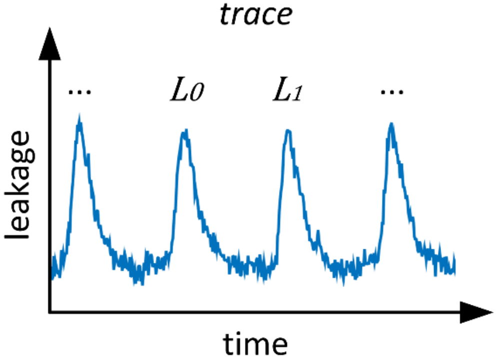


Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, ***François-Xavier Standaert***

TU Darmstadt (Germany), Radbout University Nijmegen (The Netherlands), DarkMatter LLC (UAE), UCLouvain (Belgium)

CHES 2018, Amsterdam, The Netherlands

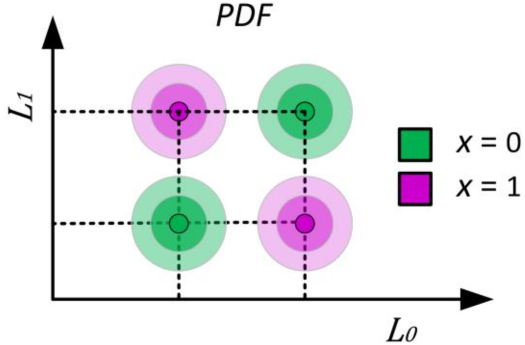
Masking (e.g., Boolean $x = x_1 + x_2 + \cdots + x_d$)



Noisy leakages security: $N \propto \frac{c}{\mathrm{MI}(X;\boldsymbol{L})}$

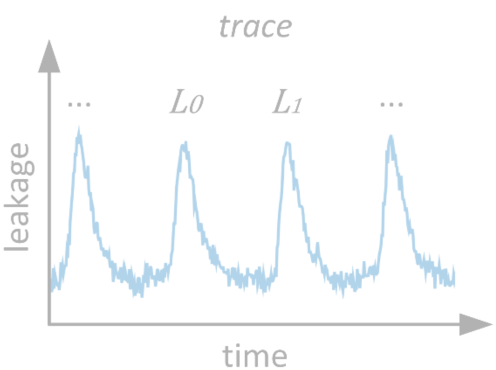
Goal (ideally): $\mathrm{MI}(X;\boldsymbol{L}) < \mathrm{MI}(X_i;L_i)^d$

Masking (e.g., Boolean $x = x_1 + x_2 + \cdots + x_d$)



Bounded moment security:

$$\underbrace{\prod_{i_1, i_2, \ldots, i_{d-1}} L_i}_{(d\text{-}1)\text{th order statistical moment (ideally)}} \perp\!\!\!\perp X$$

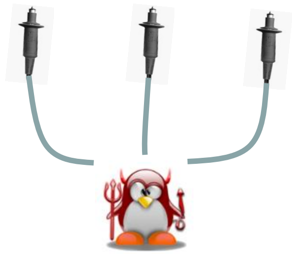


Noisy leakages security: $N \propto \frac{c}{\mathrm{MI}(X;\boldsymbol{L})}$

Goal (ideally): $\mathrm{MI}(X;\boldsymbol{L}) < \mathrm{MI}(X_i;L_i)^d$

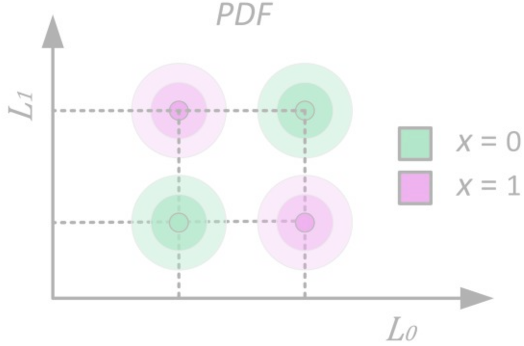
Masking (e.g., Boolean $x = x_1 + x_2 + \cdots + x_d$) 1

$x = x_1 + x_2 + \cdots + x_d$



Probing security:

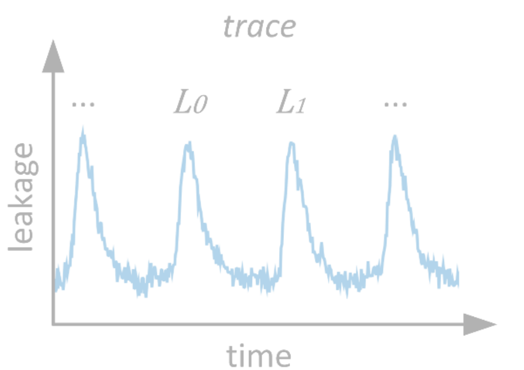
Sets of (d-1) probes are $\perp\!\!\!\perp$ of X (ideally)



Bounded moment security:

$$\underbrace{\prod_{i_1, i_2, \ldots, i_{d-1}}}\; L_i \;\perp\!\!\!\perp\; X$$

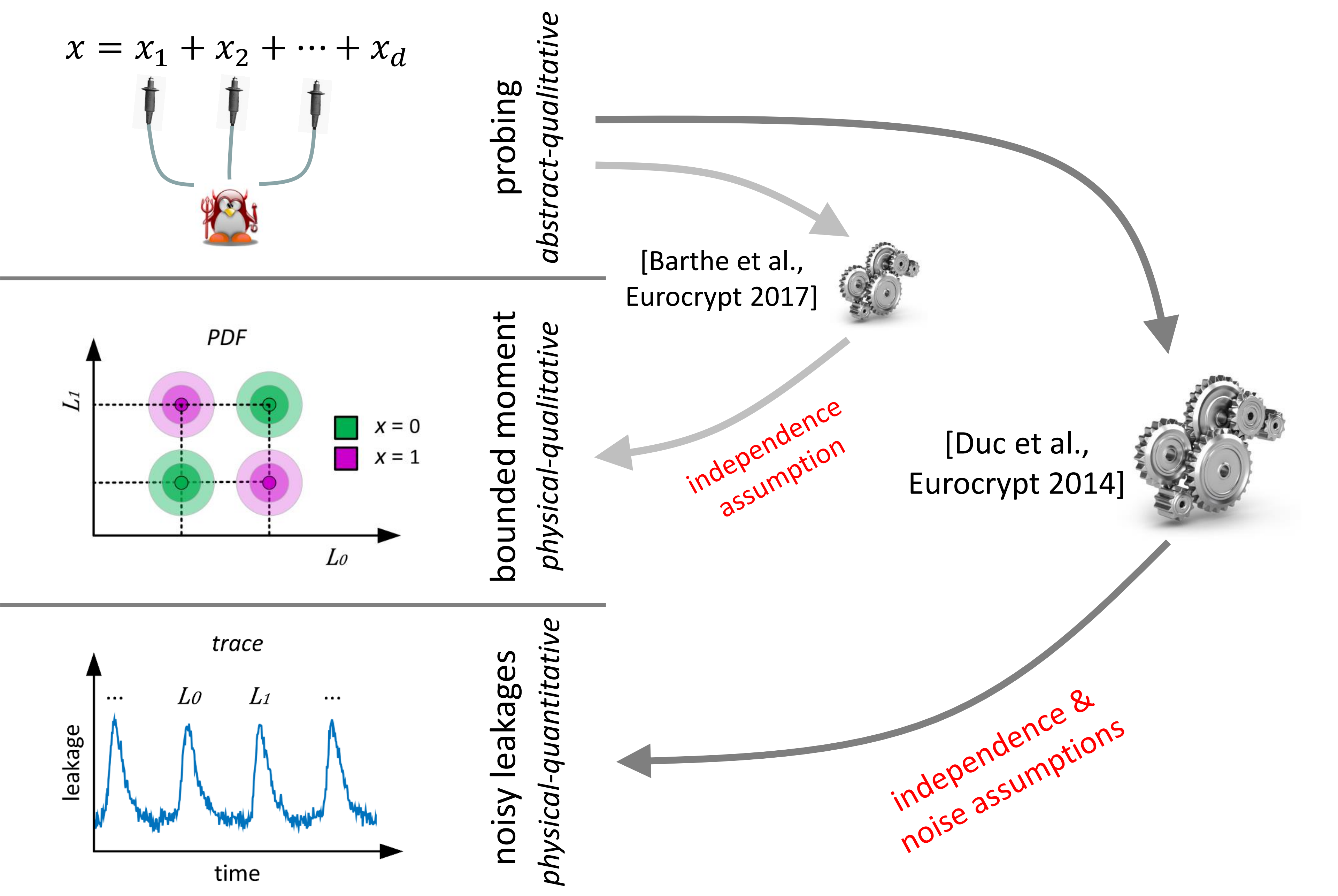
(d-1)th order statistical moment (ideally)



Noisy leakages security: $N \propto \frac{c}{\mathrm{MI}(X;\boldsymbol{L})}$

Goal (ideally): $\mathrm{MI}(X;\boldsymbol{L}) < \mathrm{MI}(X_i;L_i)^d$

Security reductions



What can go wrong? (e.g., when computing $a.b$) 3

Issue #1. Lack of randomness (can break the independence assumption)

$$\begin{pmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \end{pmatrix} \Rightarrow \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

Example: probing $c_1 = a_1.(b_1 + b_2 + b_3)$ reveals information on b (when $c_1 = 1$)

What can go wrong? (e.g., when computing $a.b$)

Issue #1. Lack of randomness (can break the independence assumption)

$$\begin{pmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \end{pmatrix} + \begin{pmatrix} 0 & r_1 & r_2 \\ r_2 & 0 & r_3 \\ r_2 & r_3 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

- mitigated by adding «refreshing gadgets »
- can be analyzed in the probing model

What can go wrong? (e.g., when computing $a.b$)

Issue #1. Lack of randomness (can break the independence assumption)

$$\begin{pmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \end{pmatrix} + \begin{pmatrix} 0 & r_1 & r_2 \\ r_2 & 0 & r_3 \\ r_2 & r_3 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

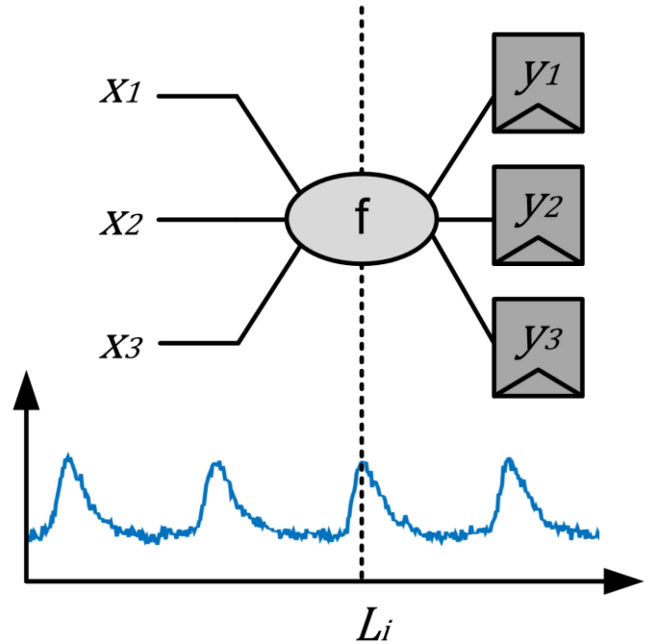
- mitigated by adding «refreshing gadgets »
- can be analyzed in the probing model

Issue #2. Physical defaults
(can break the independence assumption)

Example: glitches (transcient values)
« re-combine » the shares such that:

$$L_i = \delta(x_1 \cdot x_2 \cdot x_3)$$

(detected in the bounded moment model)



What can go wrong? (e.g., when computing $a.b$)

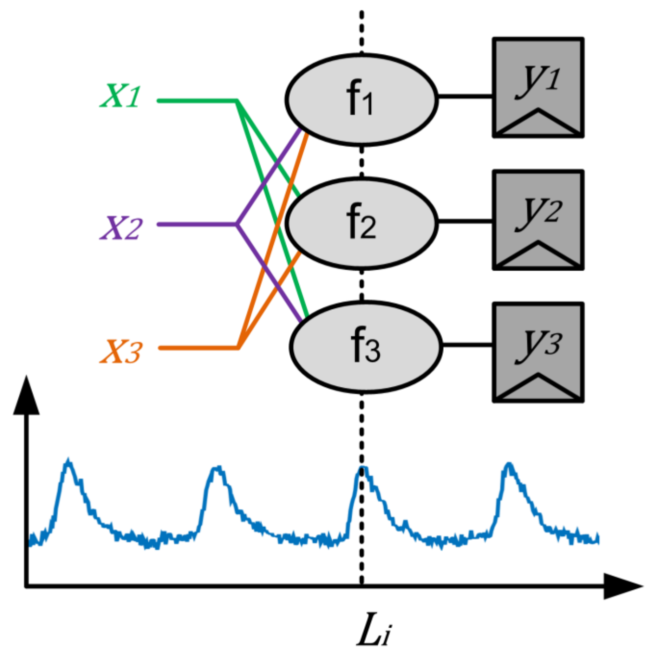
Issue #1. Lack of randomness (can break the independence assumption)

$$\begin{pmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{pmatrix} + \begin{pmatrix} 0 & r_1 & r_2 \\ r_2 & 0 & r_3 \\ r_2 & r_3 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

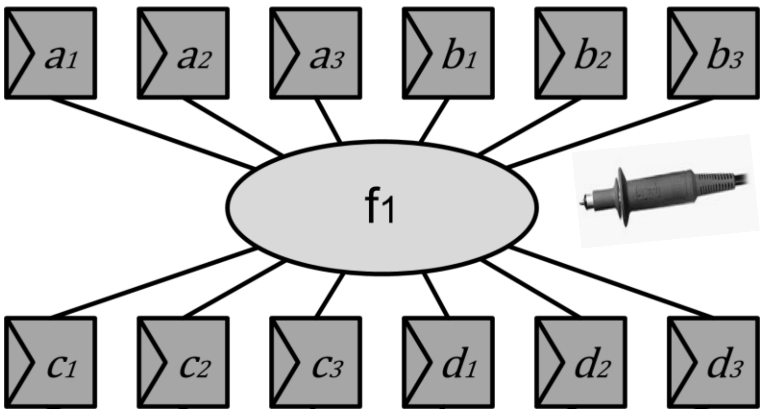
- mitigated by adding «refreshing gadgets »
- can be analyzed in the probing model

Issue #2. Physical defaults
(can break the independence assumption)

- mitigated by adding a « non-completeness » property [≈ Theshold Implementations]
- *abstract property: can be analyzed in the probing model!*

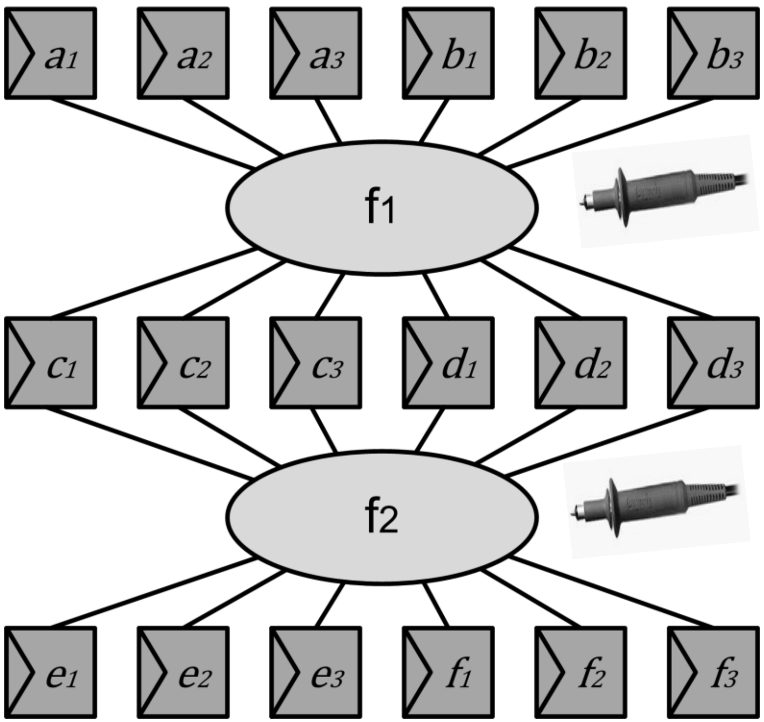


Security notions (and scalability)



***q*-probing security** [ISW, 2004]**:** any q-tuple of shares in the protected circuit is independent of any sensitive variable

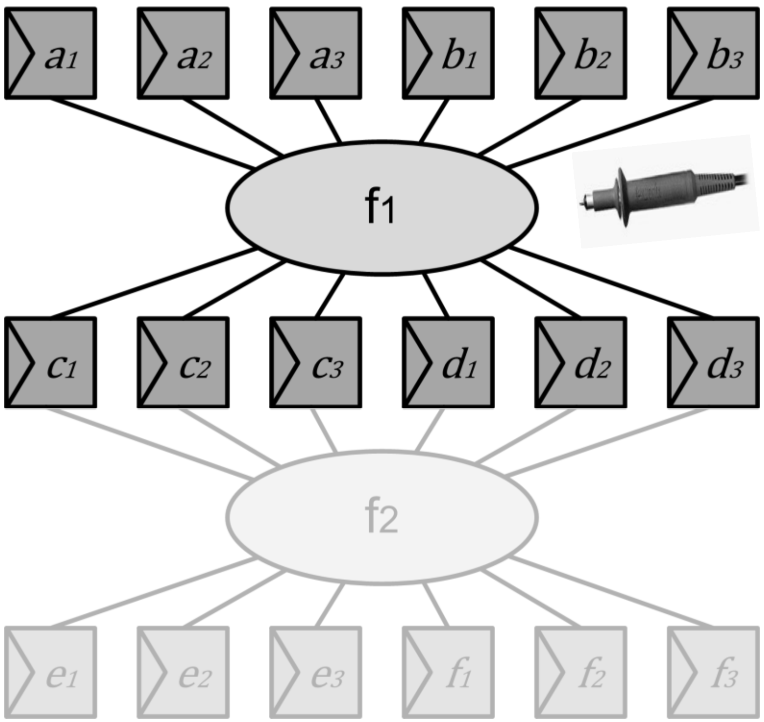
Security notions (and scalability)



***q*-probing security** [ISW, 2004]**:** any q-tuple of shares in the protected circuit is independent of any sensitive variable

Problem: the cost of testing probing security increases (very) fast with circuit size and the # of shares (since ∃ many tuples) [Barthe et al., Eurocrypt 2015]

Security notions (and scalability)



$$q_1 + q_2 \leq q$$

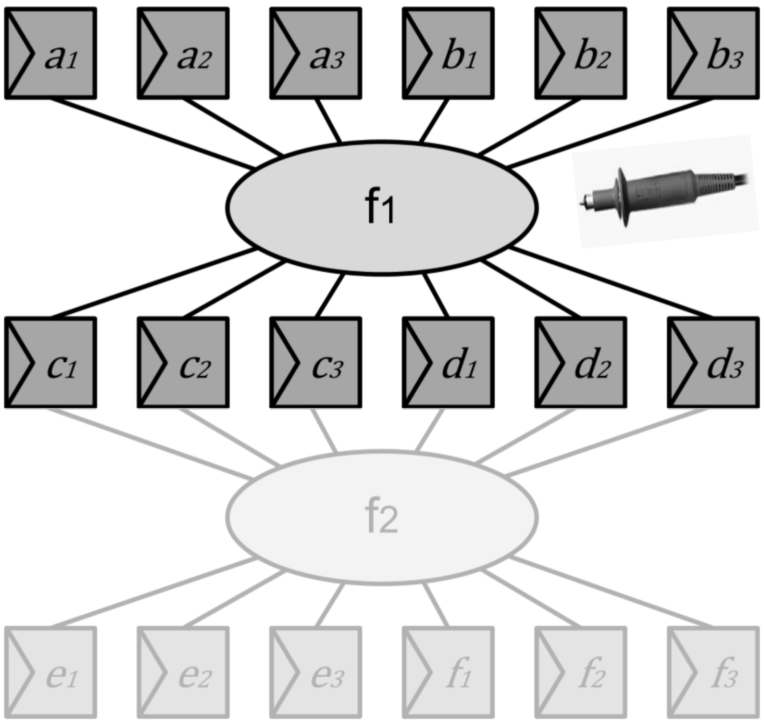
q_1 internal probes

q_2 output probes

***q*-probing security** [ISW, 2004]: any q-tuple of shares in the protected circuit is independent of any sensitive variable

***q*-(Strong) Non Interference** [Barthe et al., CCS 2016]**:** a circuit gadget (e.g., f1) is NI (SNI) if any set of $q_1 + q_2$ probes can be simulated with at most $q_1 + q_2$ (only q_1) shares of each input

$$\mathrm{D(input\ shares||probes)} \approx \mathrm{D(input\ shares||simulation)}$$



$$q_1 + q_2 \leq q$$

q_1 internal probes


q_2 output probes

***q*-probing security** [ISW, 2004]: any q-tuple of shares in the protected circuit is independent of any sensitive variable

***q*-(Strong) Non Interference** [Barthe et al., CCS 2016]**:** a circuit gadget (e.g., f1) is NI (SNI) if any set of $q_1 + q_2$ probes can be simulated with at most $q_1 + q_2$ (only q_1) shares of each input

D(input shares||probes) ≈ D(input shares||simulation)

Problem statement (simplified)

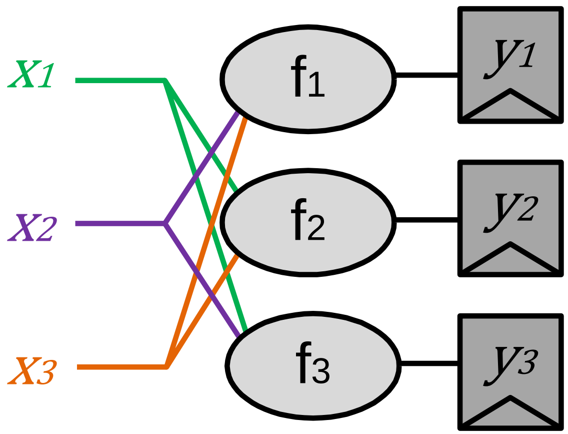
- Composable masking schemes ignore physical defaults such as glitches

$$\begin{pmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{pmatrix} + \begin{pmatrix} 0 & r_1 & r_2 \\ r_2 & 0 & r_3 \\ r_2 & r_3 & 0 \end{pmatrix}$$

Problem statement (simplified)

- Composable masking schemes ignore physical defaults such as glitches

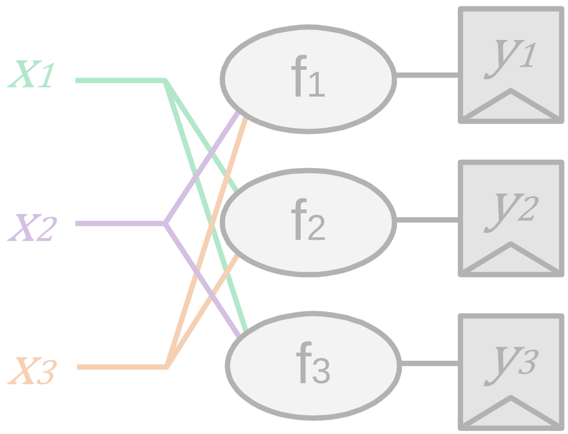
$$\begin{pmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \end{pmatrix} + \begin{pmatrix} 0 & r_1 & r_2 \\ r_2 & 0 & r_3 \\ r_2 & r_3 & 0 \end{pmatrix}$$



- Treshold implementations mitigate glitches but are only proven “uniform” (≈ probing secure) ⇒ testing scales badly

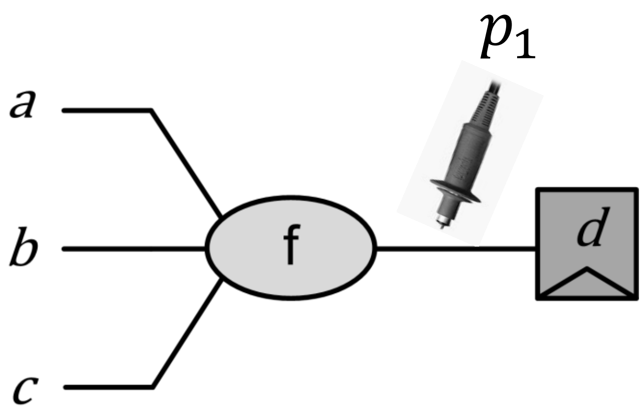
- Composable masking schemes ignore physical defaults such as glitches

$$\begin{pmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \end{pmatrix} + \begin{pmatrix} 0 & r_1 & r_2 \\ r_2 & 0 & r_3 \\ r_2 & r_3 & 0 \end{pmatrix}$$



- Treshold implementations mitigate glitches but are only proven “uniform” (≈ probing secure) ⇒ testing scales badly

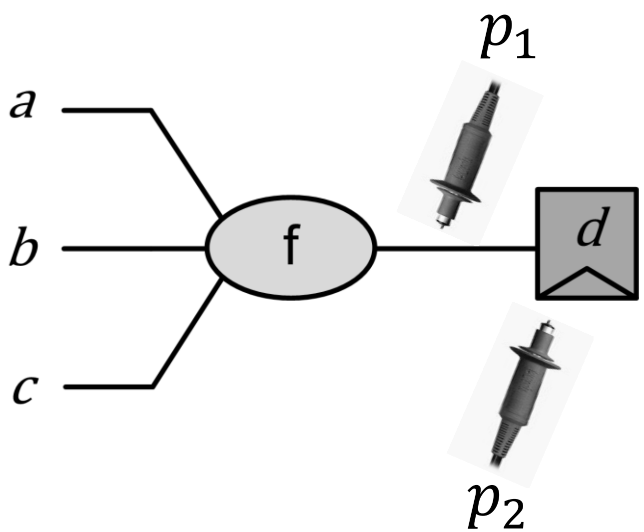
- Design & prove masked implementations that are (*jointly!*) robust against glitches and composable



Glitch-extended probes: probing any output of a combinatorial sub-circuit allows the adversary to observe all the sub-circuit inputs

Example: p_1 gives a, b and c

(Refined) model and security definition

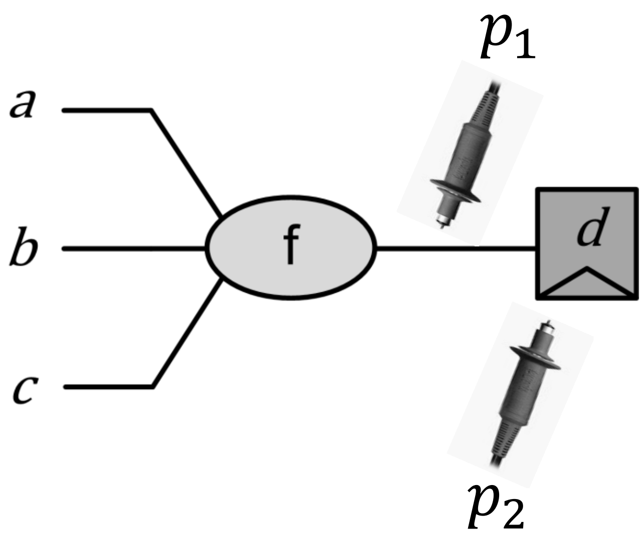


Glitch-extended probes: probing any output of a combinatorial sub-circuit allows the adversary to observe all the sub-circuit inputs

Example: p_1 gives a, b and c

Technical clarification: non-extended probes on the stable registers' values have to be considered in the simulation too

(Refined) model and security definition



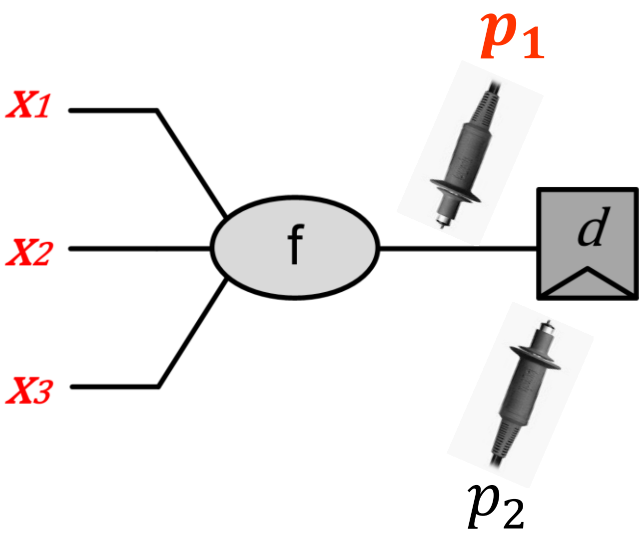
Glitch-extended probes: probing any output of a combinatorial sub-circuit allows the adversary to observe all the sub-circuit inputs

Example: p_1 gives a, b and c

Technical clarification: non-extended probes on the stable registers' values have to be considered in the simulation too

Definition: a gadget is **glitch-robust q-SNI** if it is q-SNI in the "glitch-extended" probing model

(Refined) model and security definition



Glitch-extended probes: probing any output of a combinatorial sub-circuit allows the adversary to observe all the sub-circuit inputs

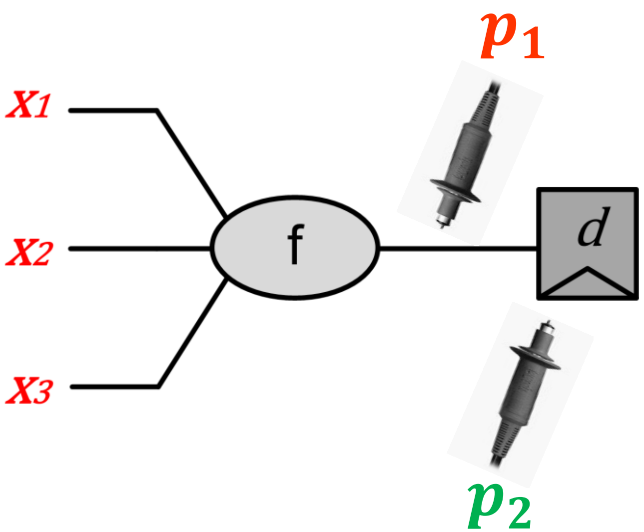
Example: p_1 gives a, b and c

Technical clarification: non-extended probes on the stable registers' values have to be considered in the simulation too

Definition: a gadget is **glitch-robust q-SNI** if it is q-SNI in the "glitch-extended" probing model

⇒ Shares' fan in of robust gadgets should be minimum

(Refined) model and security definition



Glitch-extended probes: probing any output of a combinatorial sub-circuit allows the adversary to observe all the sub-circuit inputs

Example: p_1 gives a, b and c

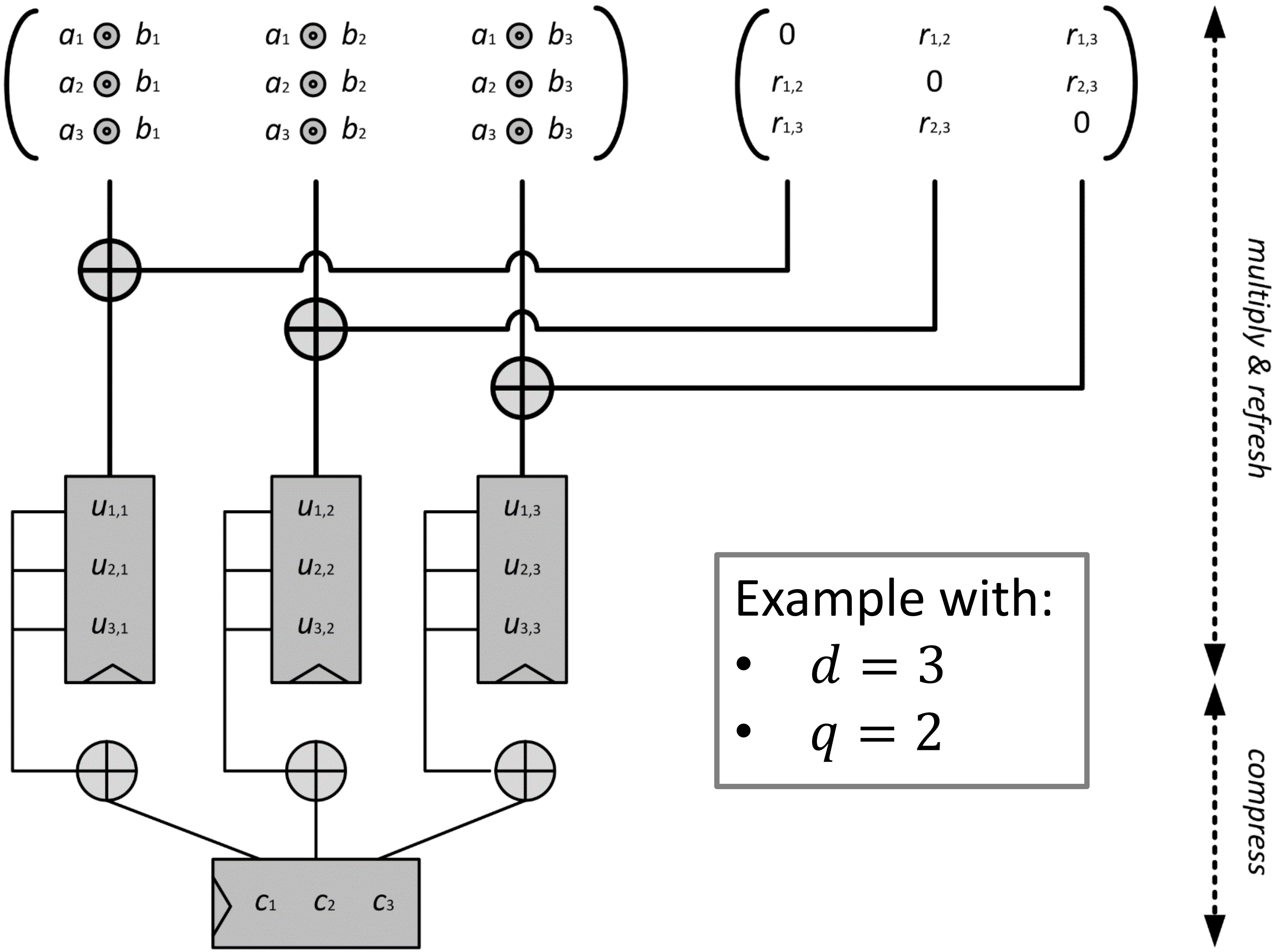
Technical clarification: non-extended probes on the stable registers' values have to be considered in the simulation too

Definition: a gadget is **glitch-robust q-SNI** if it is q-SNI in the "glitch-extended" probing model

⇒ Shares' fan in of robust gadgets should be minimum

⇒ Outputs of SNI gadgets should be stored in registers

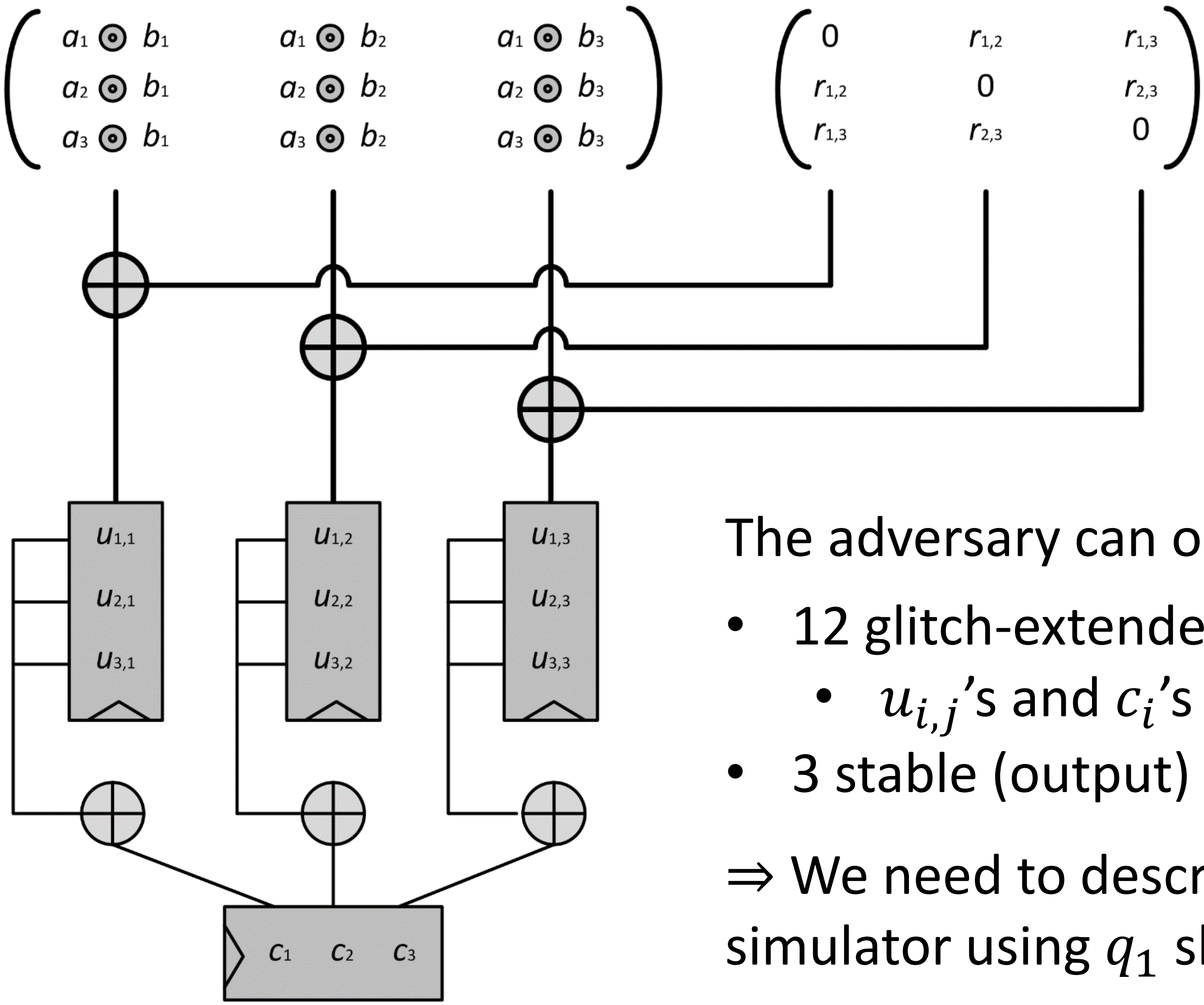
ISW mult. is glitch-robust q-SNI in 2 cycles



Example with:

- $d = 3$
- $q = 2$

ISW mult. is glitch-robust q-SNI in 2 cycles

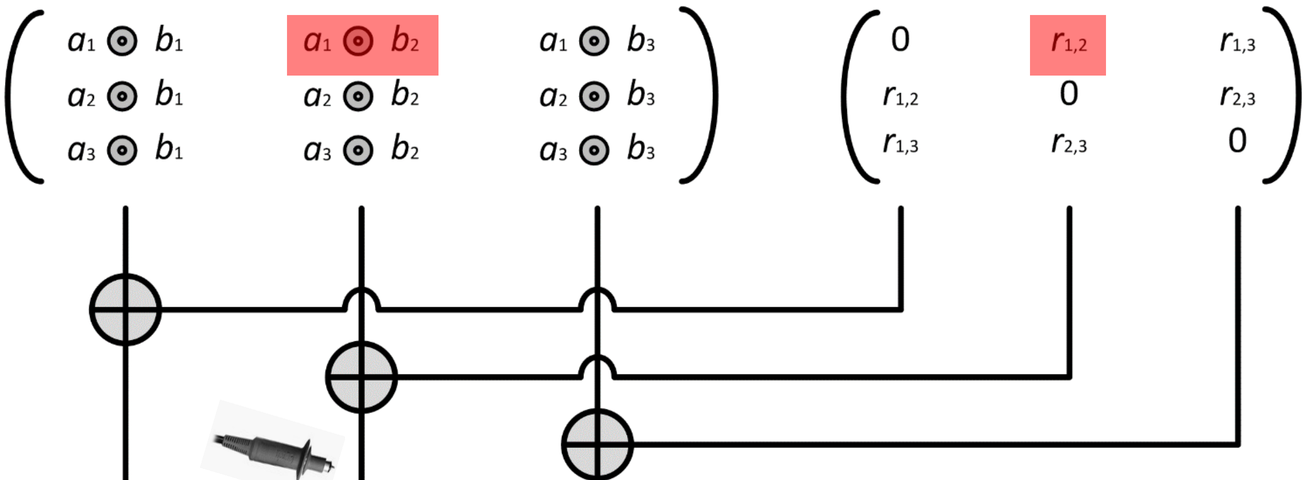


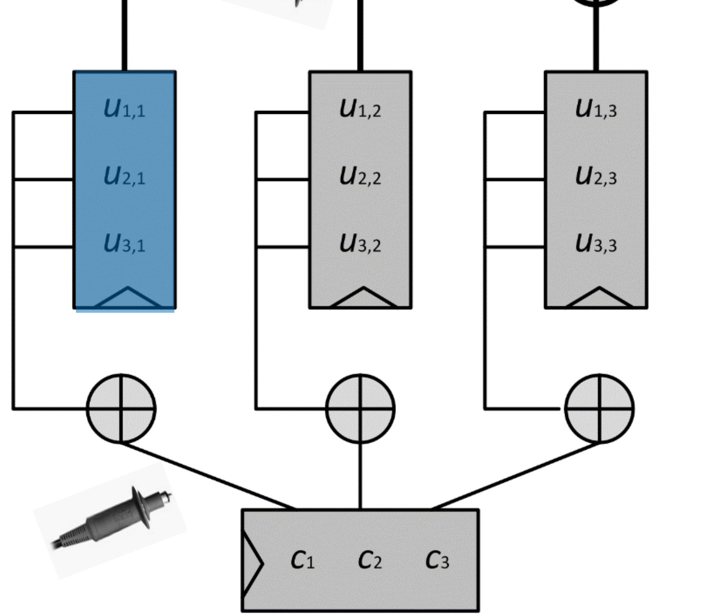
The adversary can observe:

- 12 glitch-extended probes
  - $u_{i,j}$'s and c_i's
- 3 stable (output) probes c_i's

⇒ We need to describe a simulator using q_1 shares/input

ISW mult. is glitch-robust q-SNI in 2 cycles

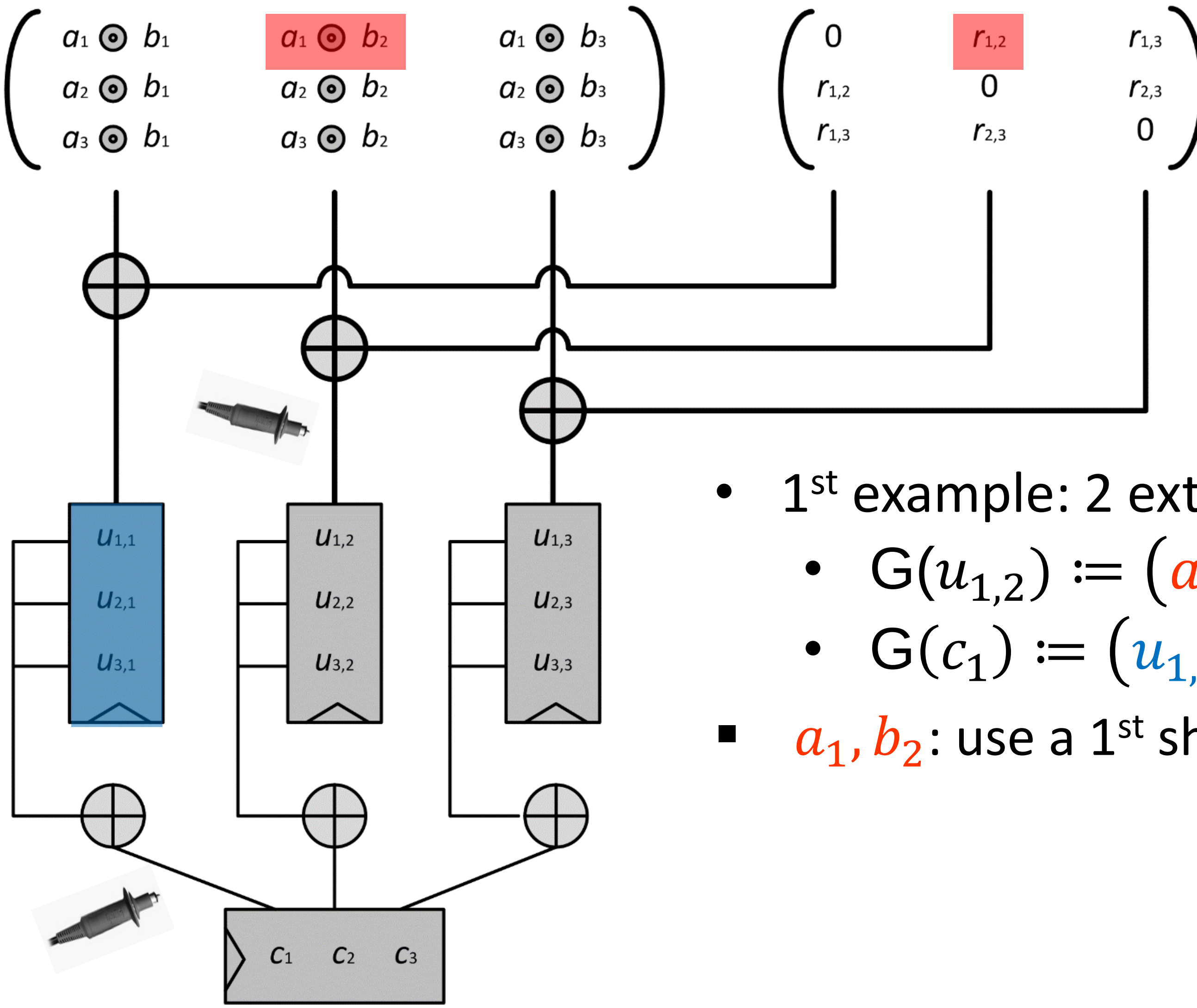




- 1st example: 2 extended probes
  - $\mathrm{G}(u_{1,2}) \coloneqq (a_1, b_2, r_{1,2})$
  - $\mathrm{G}(c_1) \coloneqq (u_{1,1}, u_{2,1}, u_{3,1})$

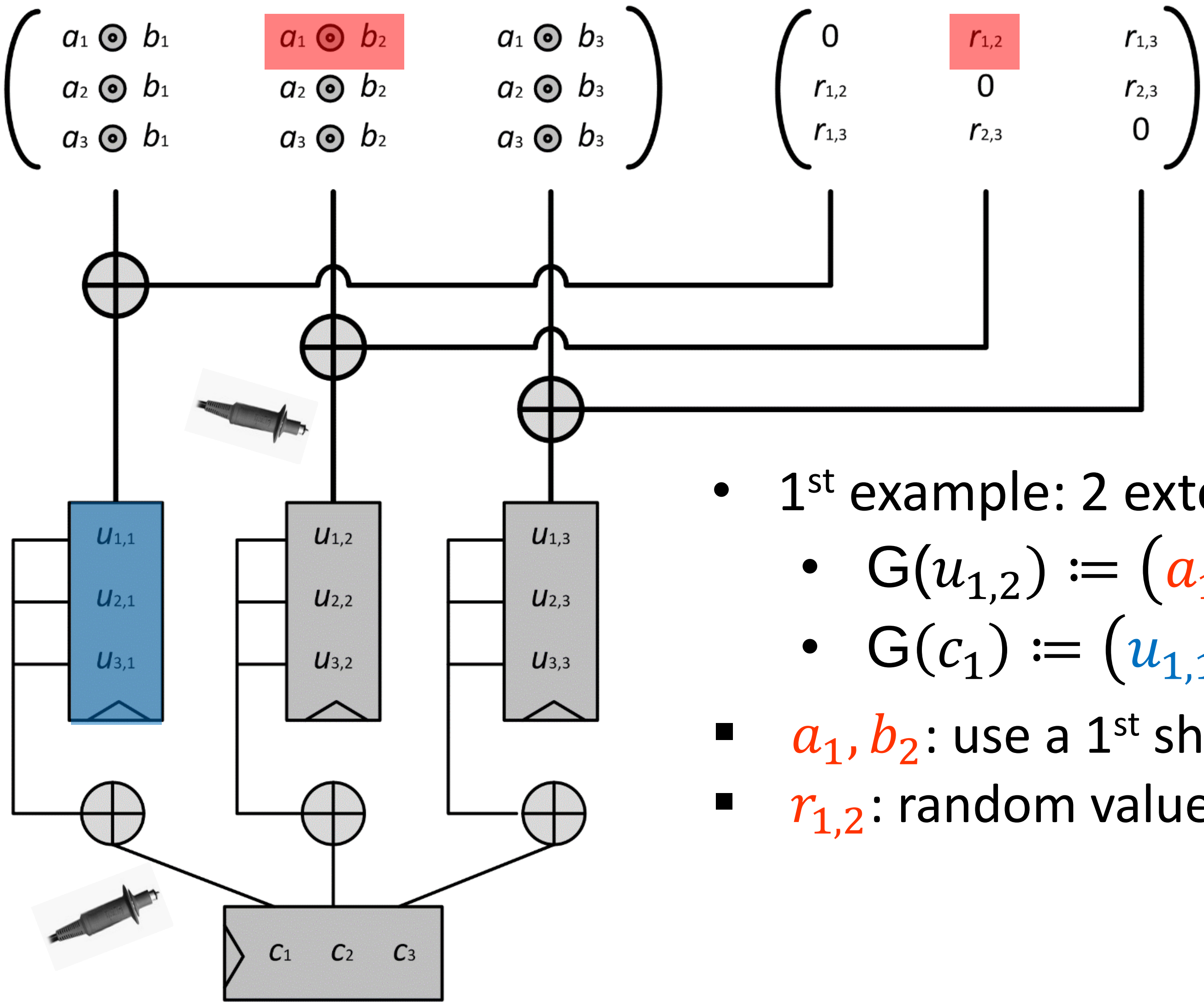
to simul. with 2 shares/input

ISW mult. is glitch-robust q-SNI in 2 cycles



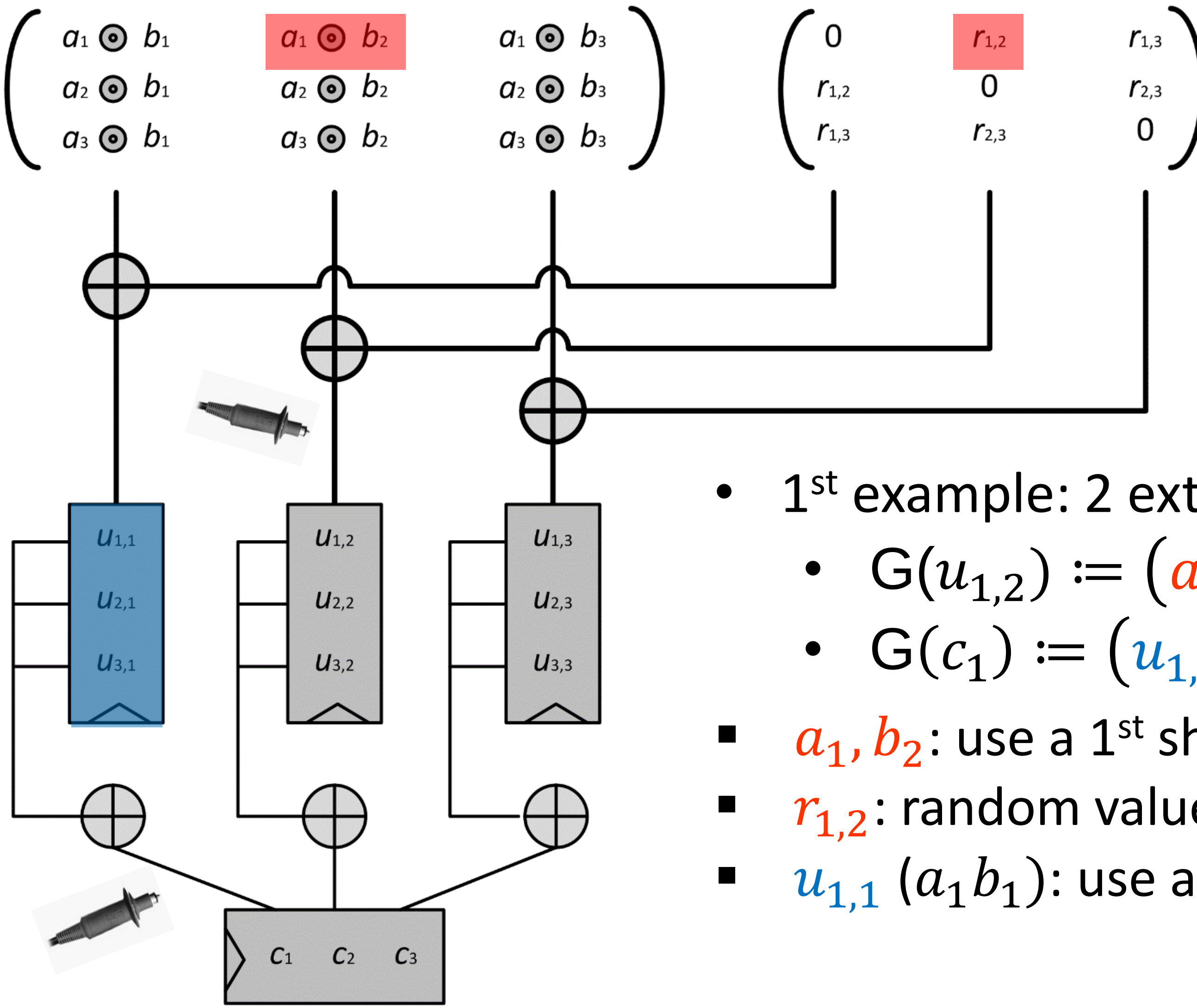
- 1st example: 2 extended probes
  - $\mathrm{G}(u_{1,2}) \coloneqq (a_1, b_2, r_{1,2})$
  - $\mathrm{G}(c_1) \coloneqq (u_{1,1}, u_{2,1}, u_{3,1})$
- a_1, b_2: use a 1st share of a, b

ISW mult. is glitch-robust q-SNI in 2 cycles



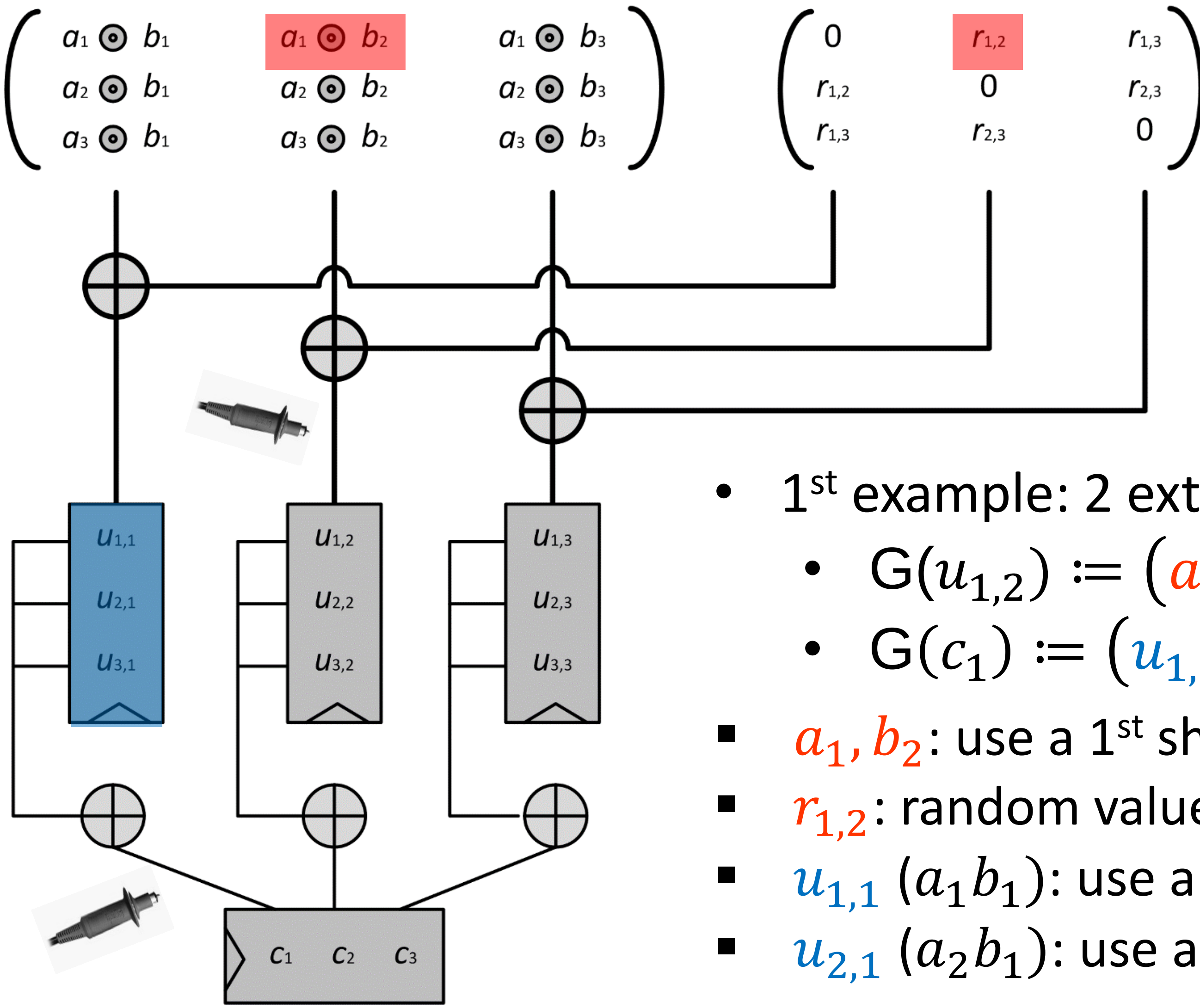
- 1^{st} example: 2 extended probes
  - $\mathrm{G}(u_{1,2}) := (a_1, b_2, r_{1,2})$
  - $\mathrm{G}(c_1) := (u_{1,1}, u_{2,1}, u_{3,1})$
- a_1, b_2: use a 1^{st} share of a, b
- $r_{1,2}$: random value

ISW mult. is glitch-robust q-SNI in 2 cycles



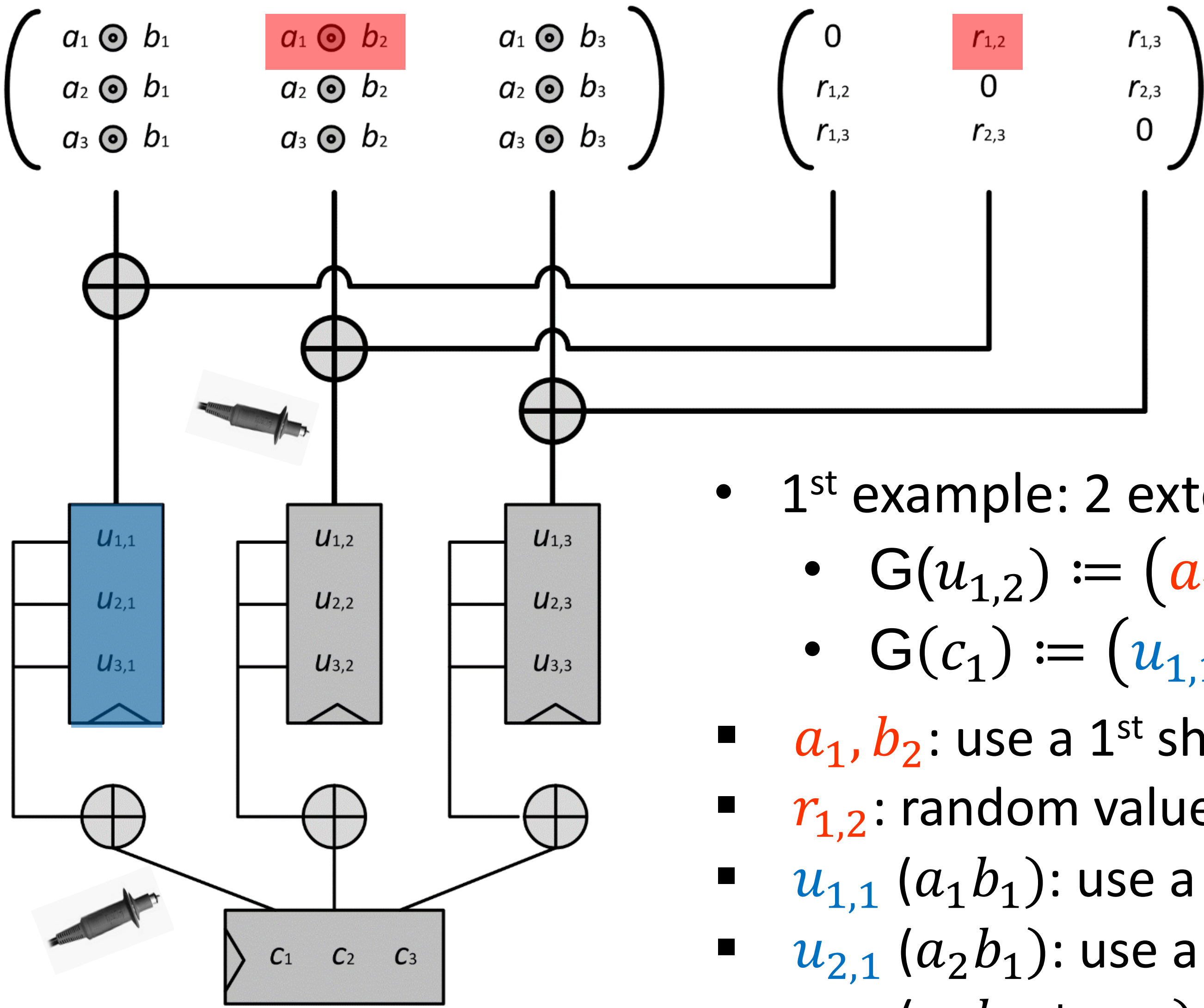
- 1^{st} example: 2 extended probes
  - $\mathrm{G}(u_{1,2}) \coloneqq (a_1, b_2, r_{1,2})$
  - $\mathrm{G}(c_1) \coloneqq (u_{1,1}, u_{2,1}, u_{3,1})$
- a_1, b_2: use a 1^{st} share of a, b
- $r_{1,2}$: random value
- $u_{1,1}$ $(a_1 b_1)$: use a 2^{nd} share of b

ISW mult. is glitch-robust q-SNI in 2 cycles



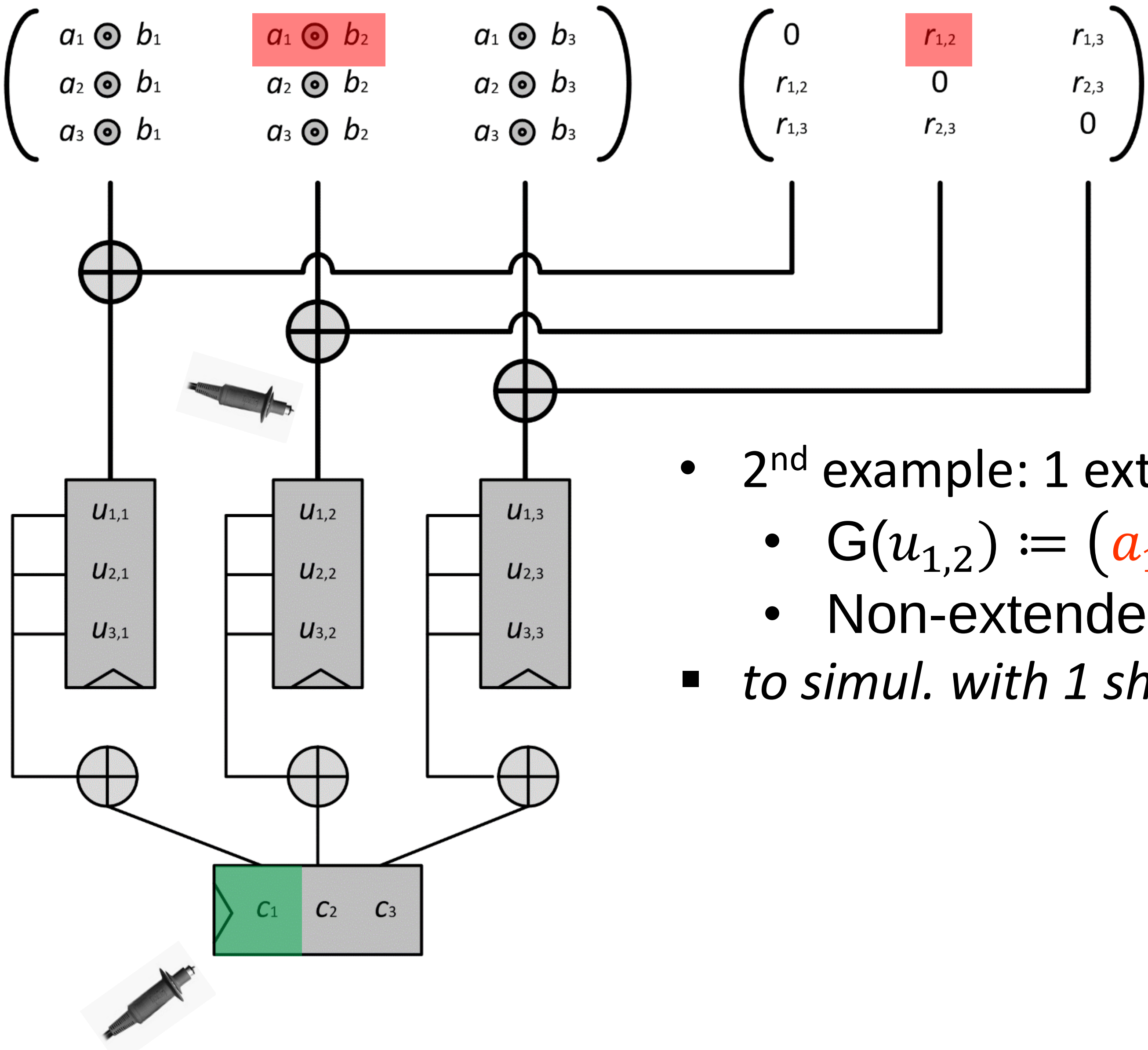
- 1^{st} example: 2 extended probes
  - $\mathrm{G}(u_{1,2}) \coloneqq (a_1, b_2, r_{1,2})$
  - $\mathrm{G}(c_1) \coloneqq (u_{1,1}, u_{2,1}, u_{3,1})$
- a_1, b_2: use a 1^{st} share of a, b
- $r_{1,2}$: random value
- $u_{1,1}$ $(a_1 b_1)$: use a 2^{nd} share of b
- $u_{2,1}$ $(a_2 b_1)$: use a 2^{nd} share of a

ISW mult. is glitch-robust q-SNI in 2 cycles

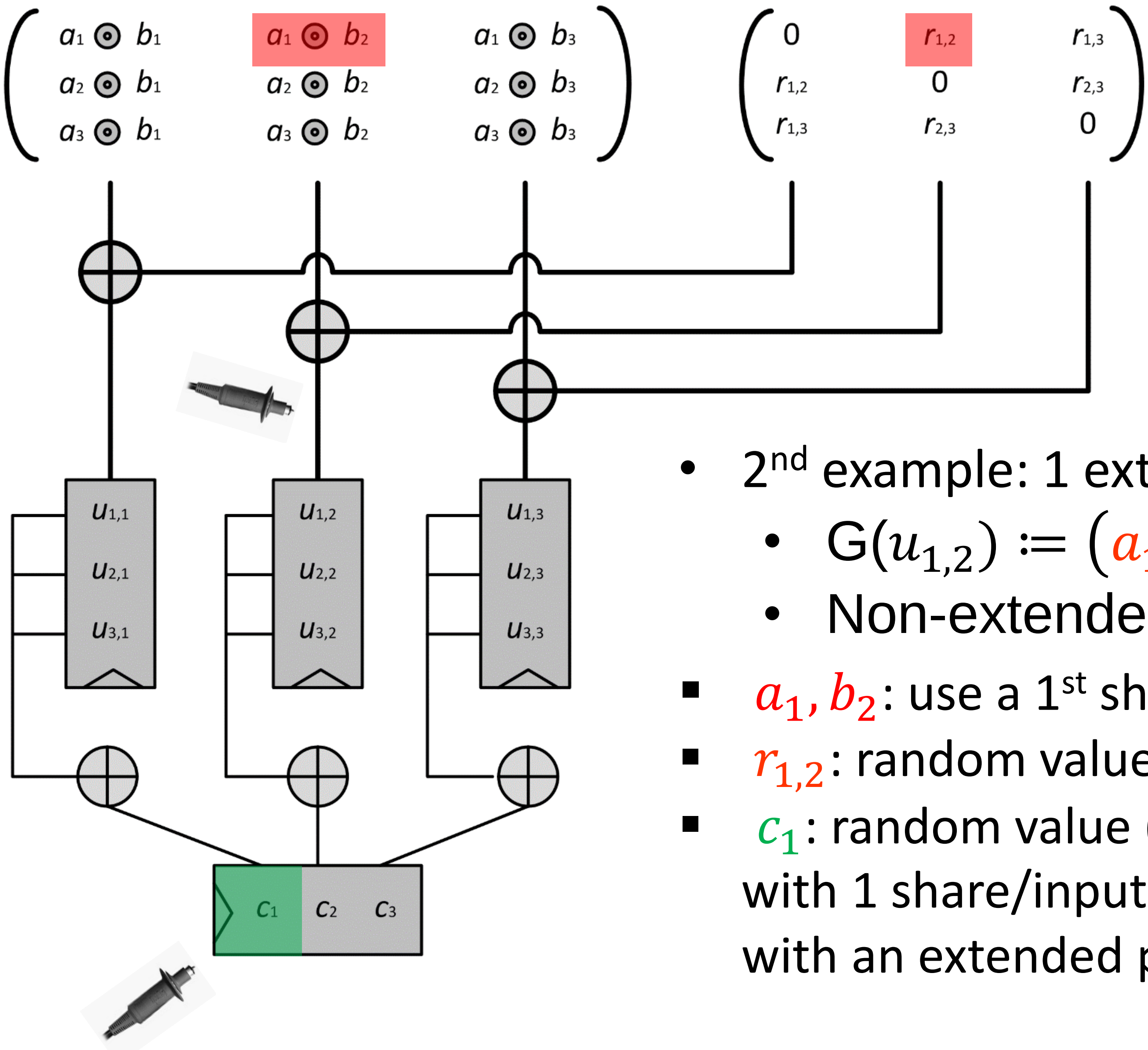


- 1^{st} example: 2 extended probes
  - $\mathrm{G}(u_{1,2}) \coloneqq (a_1, b_2, r_{1,2})$
  - $\mathrm{G}(c_1) \coloneqq (u_{1,1}, u_{2,1}, u_{3,1})$
- a_1, b_2: use a 1^{st} share of a, b
- $r_{1,2}$: random value
- $u_{1,1}$ $(a_1 b_1)$: use a 2^{nd} share of b
- $u_{2,1}$ $(a_2 b_1)$: use a 2^{nd} share of a
- $u_{3,1}$ $(a_3 b_1 + r_{1,3})$: random value

ISW mult. is glitch-robust q-SNI in 2 cycles



- 2nd example: 1 extended probe
  - $\mathrm{G}(u_{1,2}) \coloneqq (a_1, b_2, r_{1,2})$
  - Non-extended c_1
- *to simul. with 1 share/input*

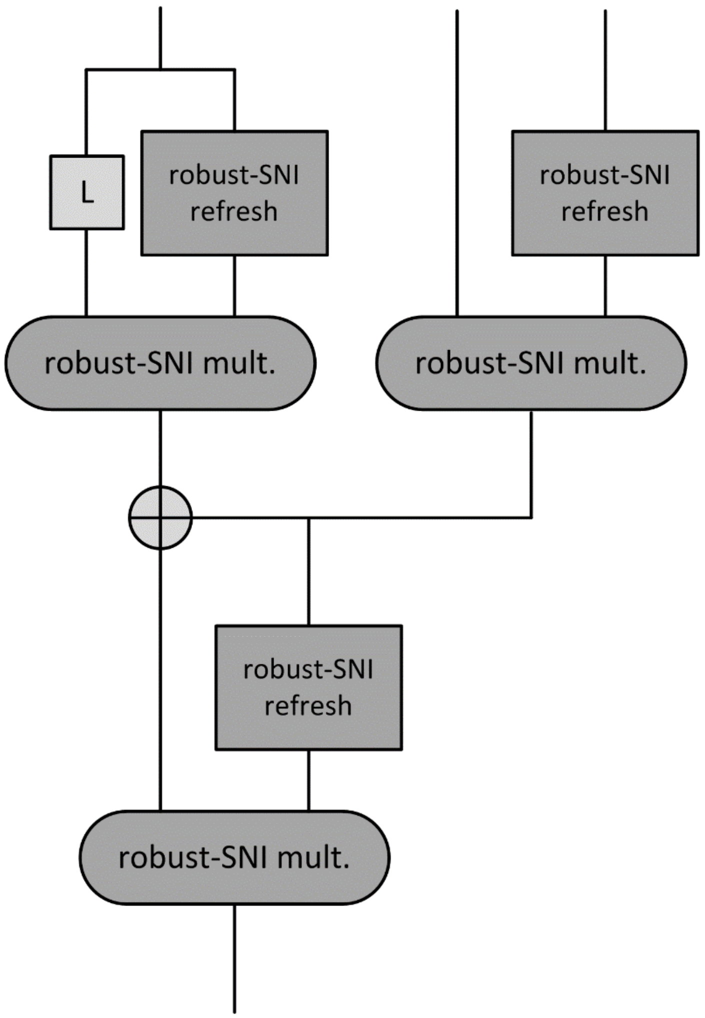


- 2^{nd} example: 1 extended probe
  - $\mathrm{G}(u_{1,2}) \coloneqq (a_1, b_2, r_{1,2})$
  - Non-extended c_1
- a_1, b_2: use a 1^{st} share of a, b
- $r_{1,2}$: random value
- c_1: random value (simulation with 1 share/input impossible with an extended probe on c_1)

How to compose (simply)

- Multiplications: use only robust-SNI multiplications with one input refreshed in a robust-SNI manner
- Perform linear operations independently on each share

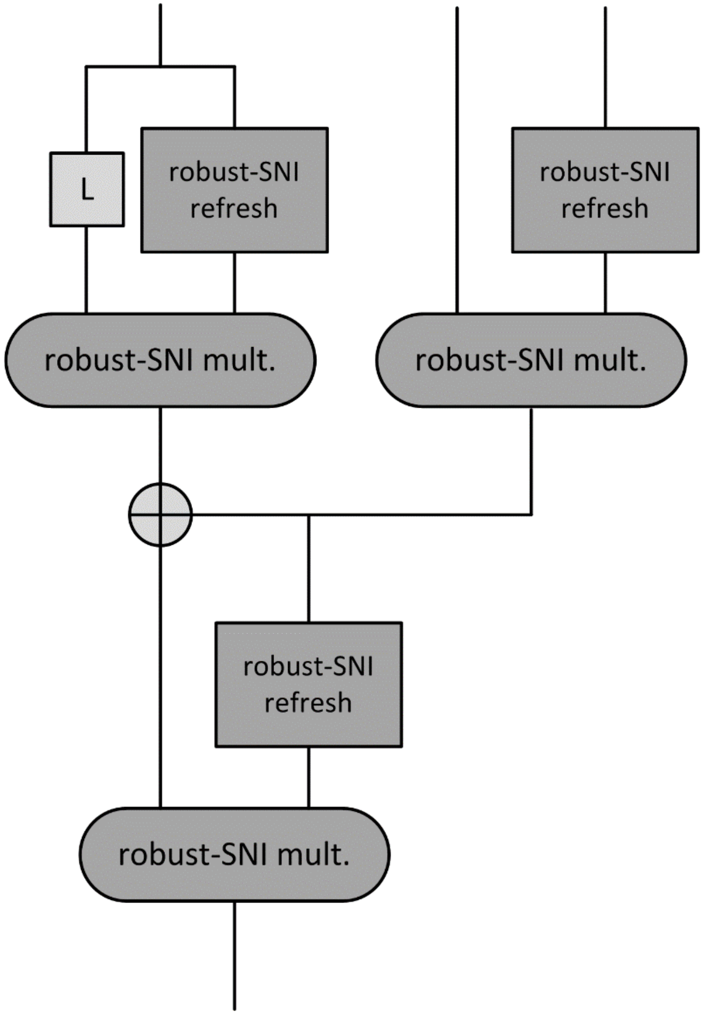
[Goudarzi & Rivain, Eurocrypt 2018], [Cassiers & Standaert, ePrint 2018]



- Multiplications: use only robust-SNI multiplications with one input refreshed in a robust-SNI manner
- Perform linear operations independently on each share

[Goudarzi & Rivain, Eurocrypt 2018], [Cassiers & Standaert, ePrint 2018]

⇒ Allows building arbitrary circuits without risk of glitches nor compositional flaws

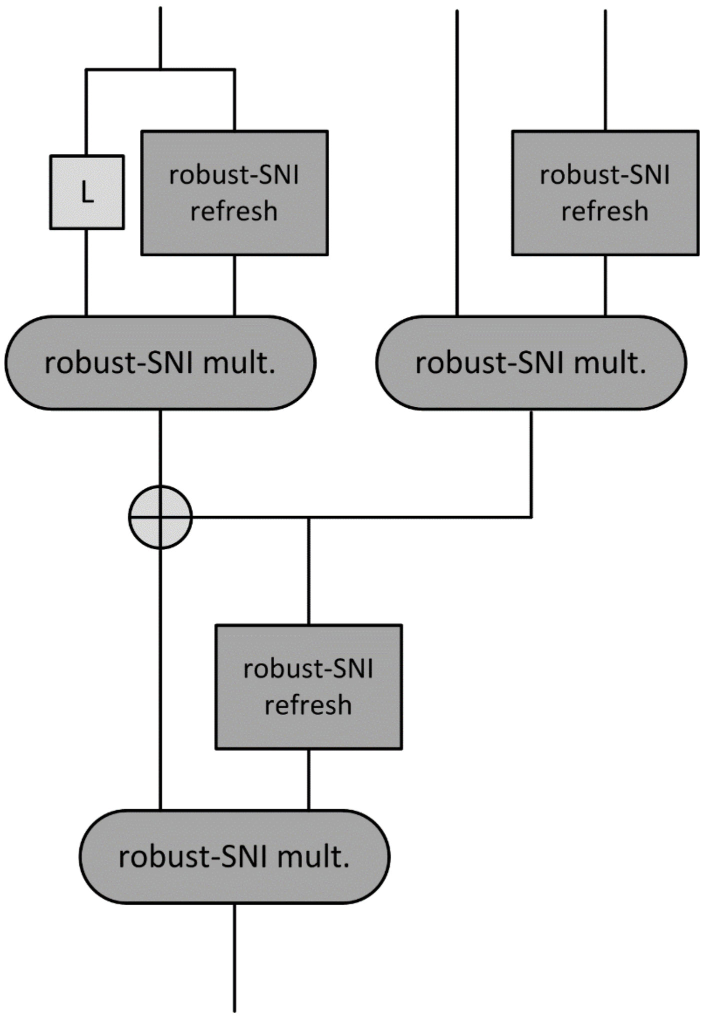


- Multiplications: use only robust-SNI multiplications with one input refreshed in a robust-SNI manner
- Perform linear operations independently on each share

[Goudarzi & Rivain, Eurocrypt 2018], [Cassiers & Standaert, ePrint 2018]

⇒ Allows building arbitrary circuits without risk of glitches nor compositional flaws

(Sufficient but not necessary!)



- Main contributions:

  1. Robust probing model
     - Allows analyzing formally and confirming the relevance of many designs ideas (e.g., Threshold Implementations, Domain Oriented Masking, Unified Masking Approach, Generic Low Latency Masking, …)
     - ***Not only a theoretical concern!***
       - Higher-order flaws in many published designs
         - https://eprint.iacr.org/2018/490

  2. A 1st multiplication algorithm/implementation proven robust against glitches and composable at any order

Other results

- “Glitch Locality Principle”
  - Glitch-robust NI + SNI (wo glitches) = glitch-robust SNI
    - By contrast, glitch-robust probing security + SNI (wo glitches) ≠ glitch-robust SNI
- More general model to capture other physical defaults (e.g., transitions-based leakages, coupling)
  - And a discussion of how they are combined
- Empirical validation (for 2-share and 3-share designs)
- More results on Threshold Implementations
  - *Pseudo-composability and reduced randomness*
  - # of cycles vs. randomness tradeoff
  - More TI decompositions based on Feistel nets.

THANKS

http://perso.uclouvain.be/fstandae/

Pseudo-composability

- Typical example: Toffoli gate $c = x \cdot y + z$
- Threshold implementation:

$$c_1 = (x_1 \cdot y_1) + (x_1 \cdot y_2) + (x_2 \cdot y_1) + z_1$$
$$c_2 = (x_2 \cdot y_2) + (x_2 \cdot y_3) + (x_3 \cdot y_2) + z_2$$
$$c_3 = (x_3 \cdot y_3) + (x_1 \cdot y_3) + (x_3 \cdot y_1) + z_3$$

Pseudo-composability a

- Typical example: Toffoli gate $c = x \cdot y + z$

- Threshold implementation:

$$c_1 = (x_1 \cdot y_1) + (x_1 \cdot y_2) + (x_2 \cdot y_1) + z_1$$
$$c_2 = (x_2 \cdot y_2) + (x_2 \cdot y_3) + (x_3 \cdot y_2) + z_2$$
$$c_3 = (x_3 \cdot y_3) + (x_1 \cdot y_3) + (x_3 \cdot y_1) + z_3$$

- Not NI nor SNI (e.g., it is impossible to simulate a probe on c_1 with a single share per input (lack of internal rand)
  - But “pseudo-NI/pseudo-SNI” if the monomials of z are used once and one assumes that can be considered as random
  - Can lead to nice randomness optimizations at low orders!