PhD defense

—

Methods for protecting intellectual property of IP core designers

Brice Colombier

Univ Lyon, UJM-Saint-Etienne, CNRS
Laboratoire Hubert Curien UMR 5516
F-42023, SAINT-ETIENNE, France

October 19th, 2017

UNIVERSITÉ DE LYON
UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE
La Région Auvergne-Rhône-Alpes
SALWARE
French ANR Project

Semiconductor industry: current situation and issues

The semiconductor market is **highly competitive**, must handle **increasing** design **complexity** and **market pull** by customers.

In last two years:

- **sales** of almost $340 billion a year [1],
- **front-end investment** for manufacturing plants reaches tens of billion dollars [2],
- **merger and acquisitions** made headlines (NXP by Qualcomm, Altera by Intel, etc) and reached almost $100 billion a year [3].

---

[1] World Semiconductor Trade Statistics *"Global Semiconductor Sales Reach $339 Billion in 2016"*

[2] EETimes *"Samsung Breaks Ground on $14 Billion Fab"*

[3] IC Insights *"2015-2016 deals dominate semiconductor M&A ranking"*

The semiconductor market is **highly competitive**, must handle **increasing** design **complexity** and **market pull** by customers.

In last two years:

- **sales** of almost $340 billion a year [1],
- **front-end investment** for manufacturing plants reaches tens of billion dollars [2],
- **merger and acquisitions** made headlines (NXP by Qualcomm, Altera by Intel, etc) and reached almost $100 billion a year [3].

Main shift

Transition from **industrial** economy to **knowledge** economy

---

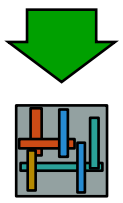
[1] World Semiconductor Trade Statistics *"Global Semiconductor Sales Reach $339 Billion in 2016"*

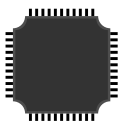
[2] EETimes *"Samsung Breaks Ground on $14 Billion Fab"*

[3] IC Insights *"2015-2016 deals dominate semiconductor M&A ranking"*

Semiconductor industry in 1960's: an industrial economy

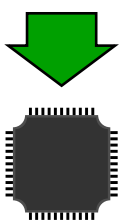
Integrated device manufacturer (*e.g.* Intel)



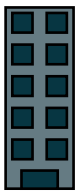


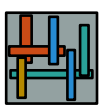
Semiconductor industry in 1980's: an economy in transition

Integrated device manufacturer (*e.g.* Intel)

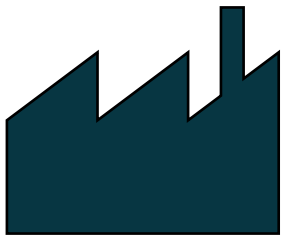


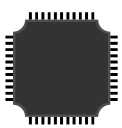
Fabless designer (*e.g.* Xilinx)





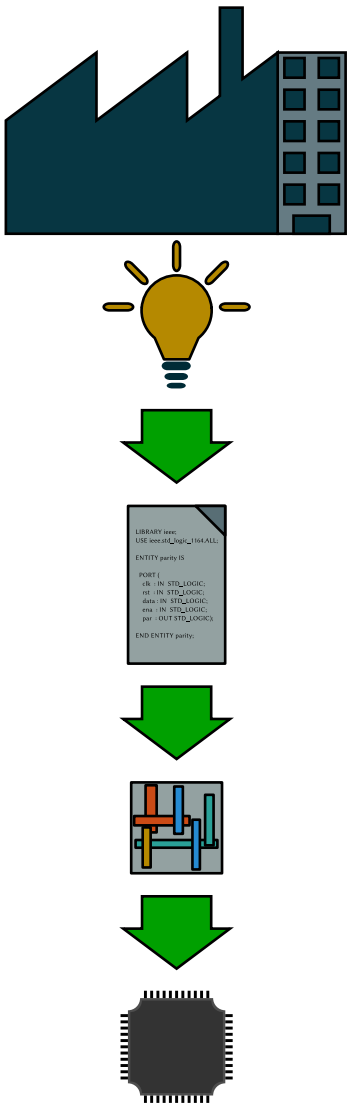
Foundry (*e.g.* TSMC)





Semiconductor industry in 2000's: a knowledge economy

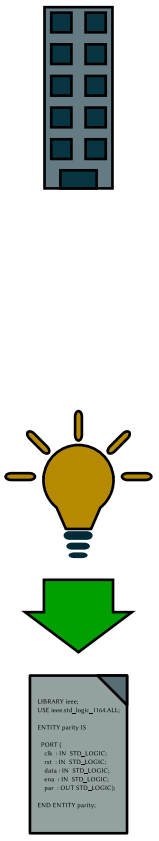
Integrated device manufacturer (*e.g.* Intel)



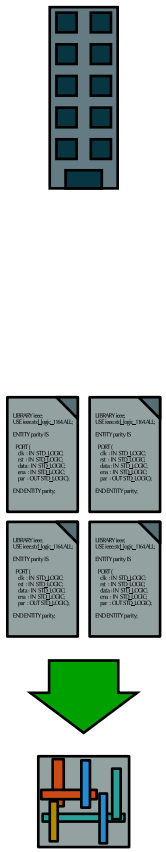
Fabless designer (*e.g.* Xilinx)

Foundry (*e.g.* TSMC)

IP core designer (*e.g.* ARM)

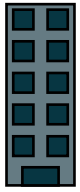


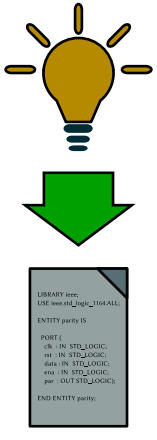
System integrator



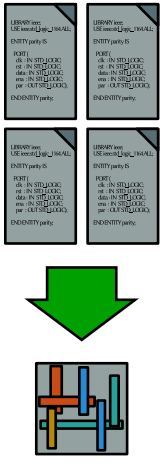
Semiconductor industry in 2000's: a knowledge economy

IP core designer (*e.g.* ARM)



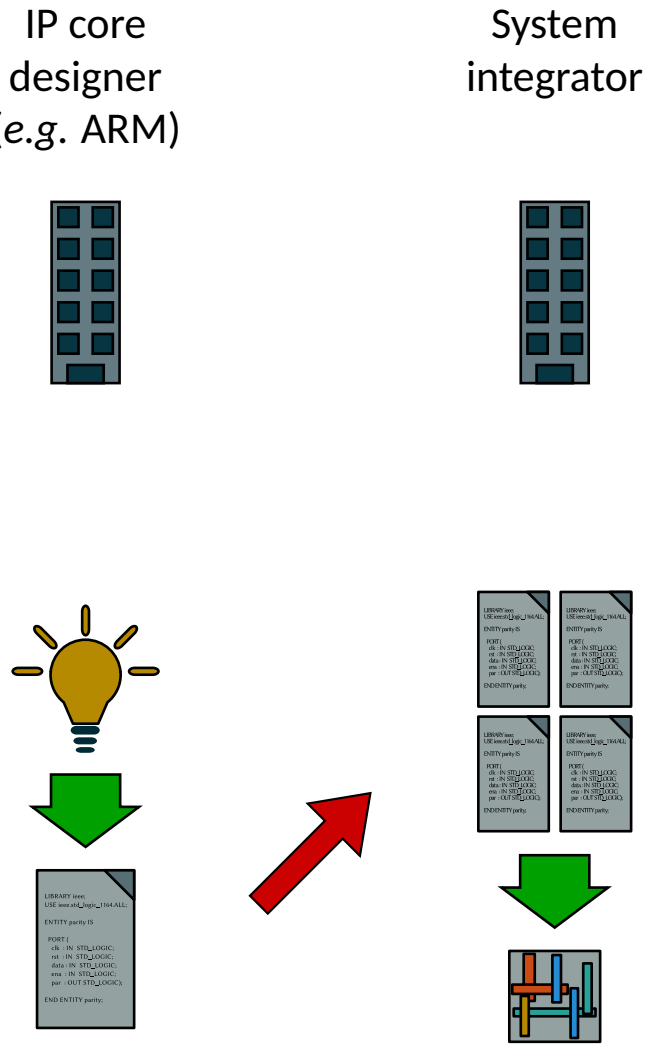


System integrator



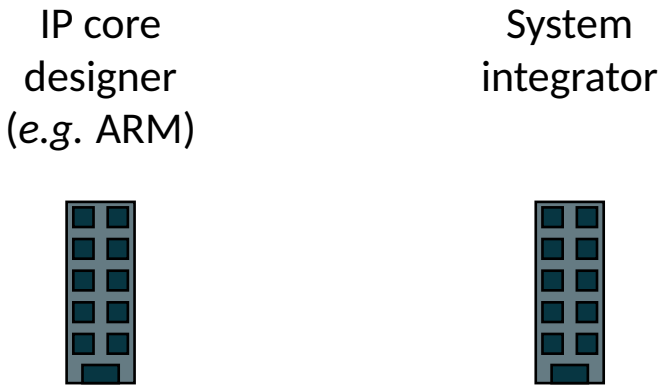
Asymmetric design knowledge transfer

The IP core designer **must give away** all the intellectual property to the system integrator.



Asymmetric design knowledge transfer

The IP core designer **must give away** all the intellectual property to the system integrator.

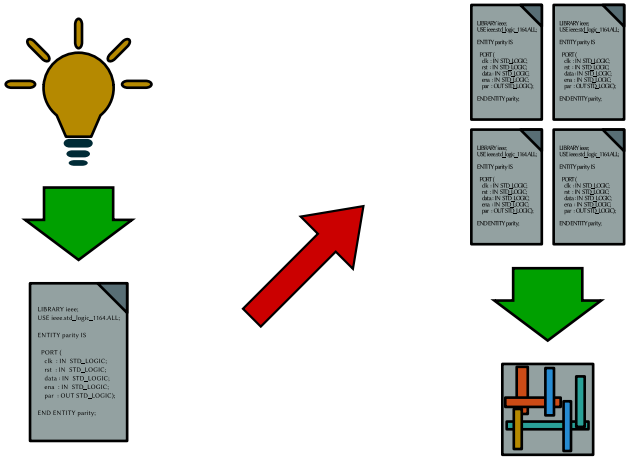


Main issue

The designer cannot know **how many times** the IP core has been instantiated.

This **asymmetric** design transfer:

- inhibits fine-grained licensing schemes,
- prevents proper billing,
- leads to many cases of **illegal copying**.



Threat model and objective

Attacker
(System integrator)

Objective

- Instantiate the IP core **without the consent** of the IP core designer.

Attacker
(System integrator)

Objective

- Instantiate the IP core **without the consent** of the IP core designer.

Capabilities

- Can obtain a **legitimate copy** of the IP core,
- Has all the **technical resources** to instantiate the IP core correctly.

Attacker (System integrator)

Objective

- Instantiate the IP core **without the consent** of the IP core designer.

Capabilities

- Can obtain a **legitimate copy** of the IP core,
- Has all the **technical resources** to instantiate the IP core correctly.

Defender (IP core designer)

Objectives

- Prevent **under-reporting** of IP core instances,
- Make illegal copies **unusable**.

Attacker (System integrator)

Objective

- Instantiate the IP core **without the consent** of the IP core designer.

Capabilities

- Can obtain a **legitimate copy** of the IP core,
- Has all the **technical resources** to instantiate the IP core correctly.

Defender (IP core designer)

Objectives

- Prevent **under-reporting** of IP core instances,
- Make illegal copies **unusable**.

Constraint

- Keep the overall cost **inferior** to the financial losses on illegal copying.

Objective: a **licensing** scheme for IP cores

- ✗ **Before activation:** The IP core does not operate correctly/at all.
- **During activation:** The IP core must be provided with the correct, instance-specific activation word
- ✓ **After activation:** The IP core operates normally.

Business

- Monetization,
- Pay-per-use,
- Royalties.

Software

- Modified design flow,
- Database,
- Key management.

Hardware

- Activation,
- ID,
- Security.

Objective: a **licensing** scheme for IP cores

- ✗ **Before activation:** The IP core does not operate correctly/at all.
- **During activation:** The IP core must be provided with the correct, instance-specific activation word
- ✓ **After activation:** The IP core operates normally.

Business

- Monetization,
- Pay-per-use,
- Royalties.

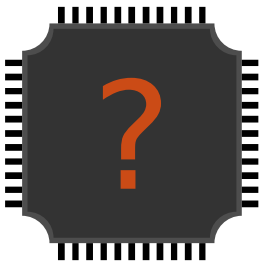
Software

- Modified design flow,
- Database,
- Key management.

Hardware

- Activation,
- ID,
- Security.

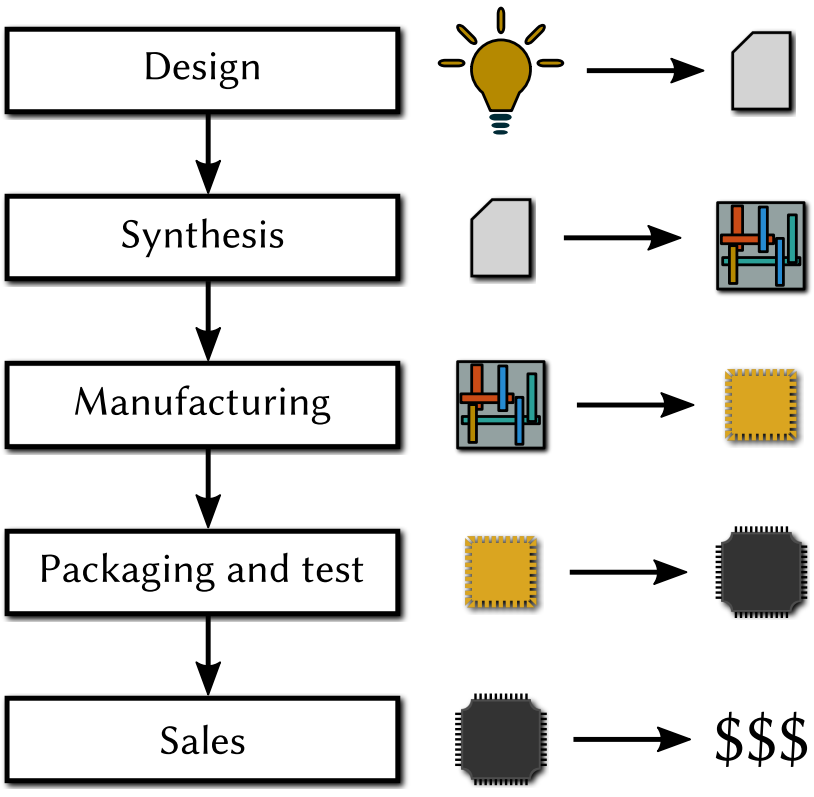
A **secure remote activation scheme** that is:



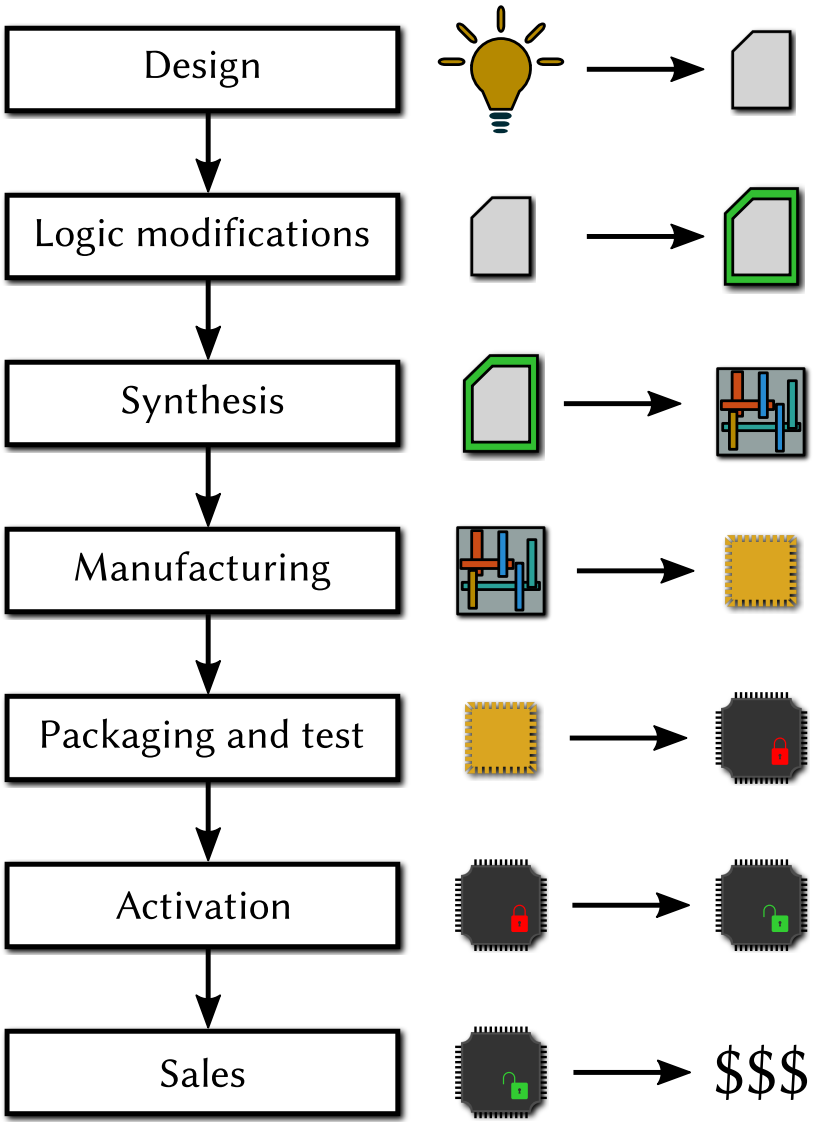
- **Easy to use** by legitimate parties,
  - seamless **integration** into the standard design flow,
  - low impact on the **performances** of the IP core,
  - usable for any IP core (**universal**),
- **Hard to circumvent** for an adversary,
  - **impossibility** to use illegal copies,
  - **security** guaranteed by a cipher,
  - **instance**-specific identifier.

Contributions and outline

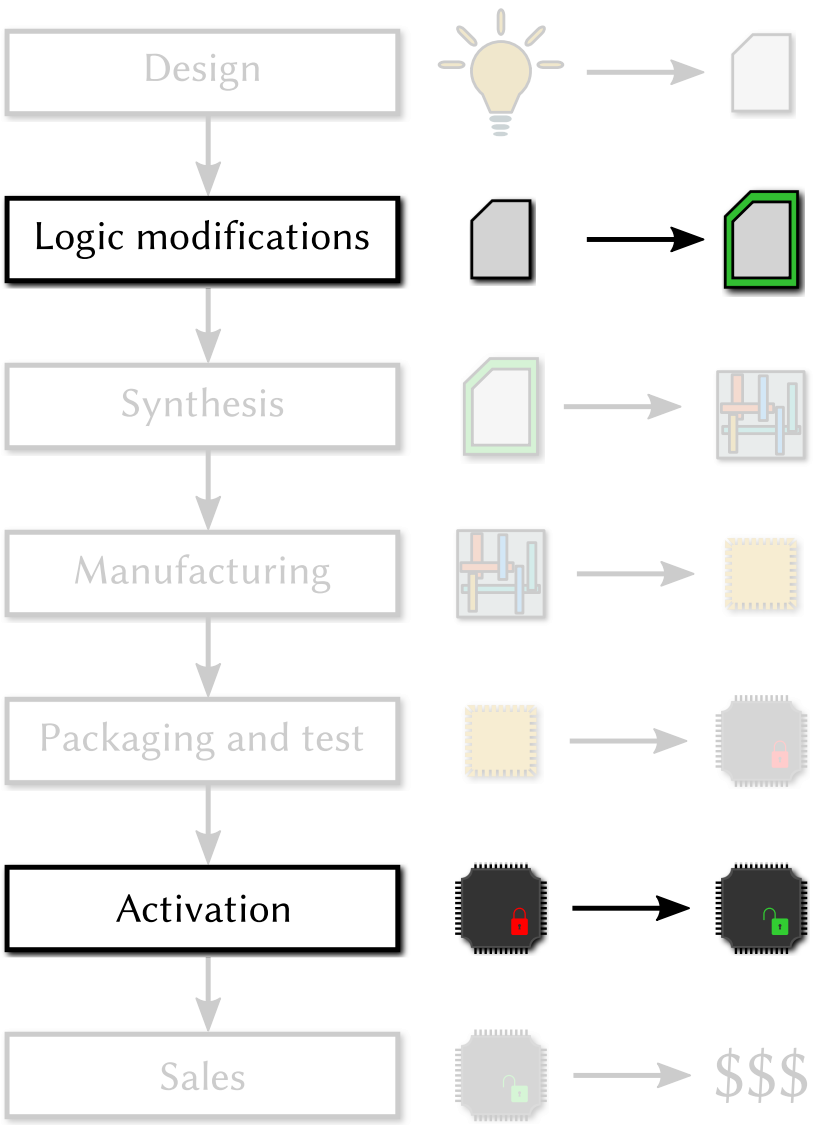
Standard (and heavily simplified) design flow



Augmented design flow



Augmented design flow



Logic modifications

Modify the combinational logic to allow to **controllably**:

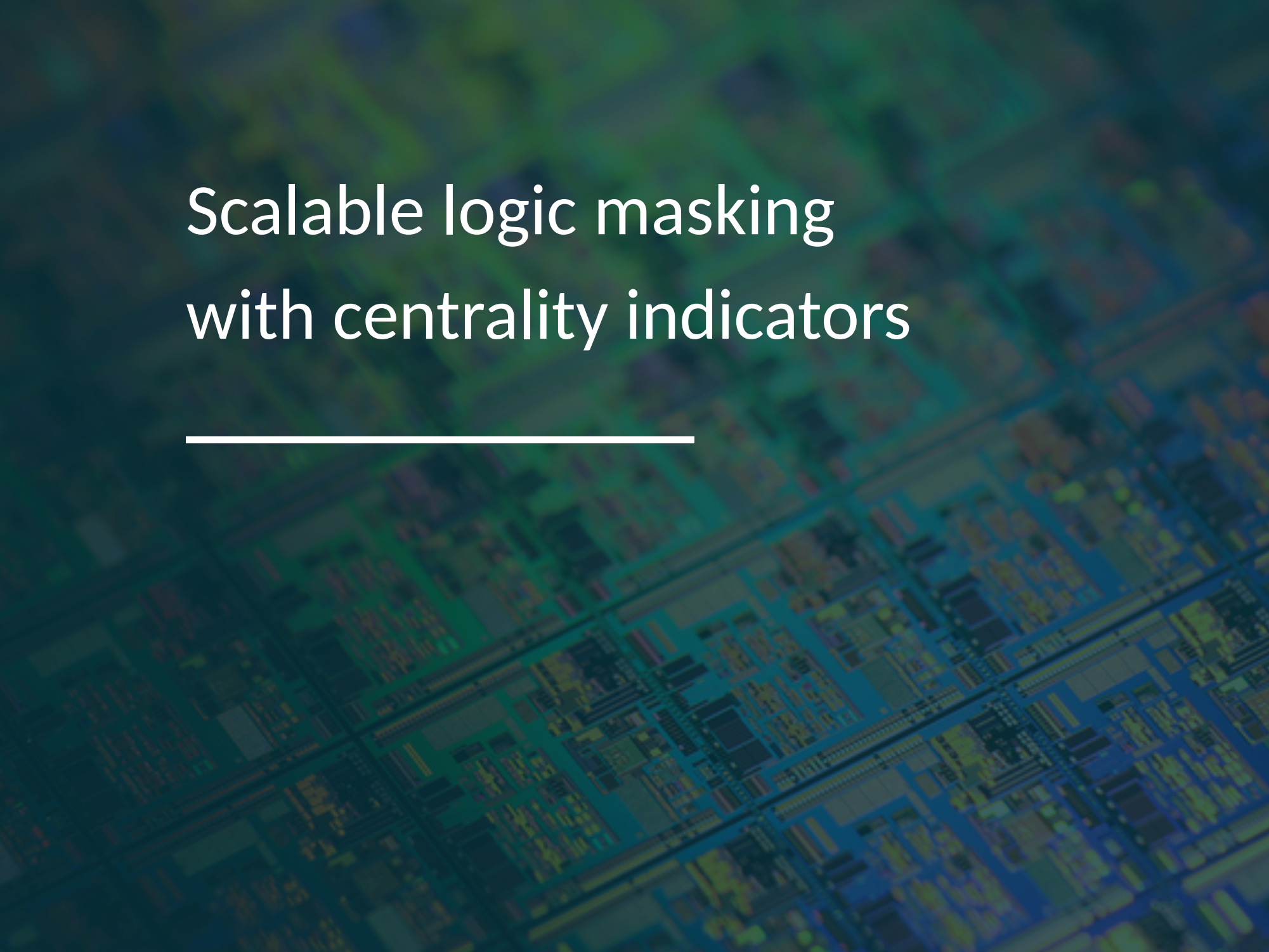
- **Lock** the outputs (logic locking),
- **Alter** the outputs (logic masking).

Activation

- **Intrinsic** identification of the **instances** of the IP core.
- **Secure** transfer of the **instance-specific** activation word.

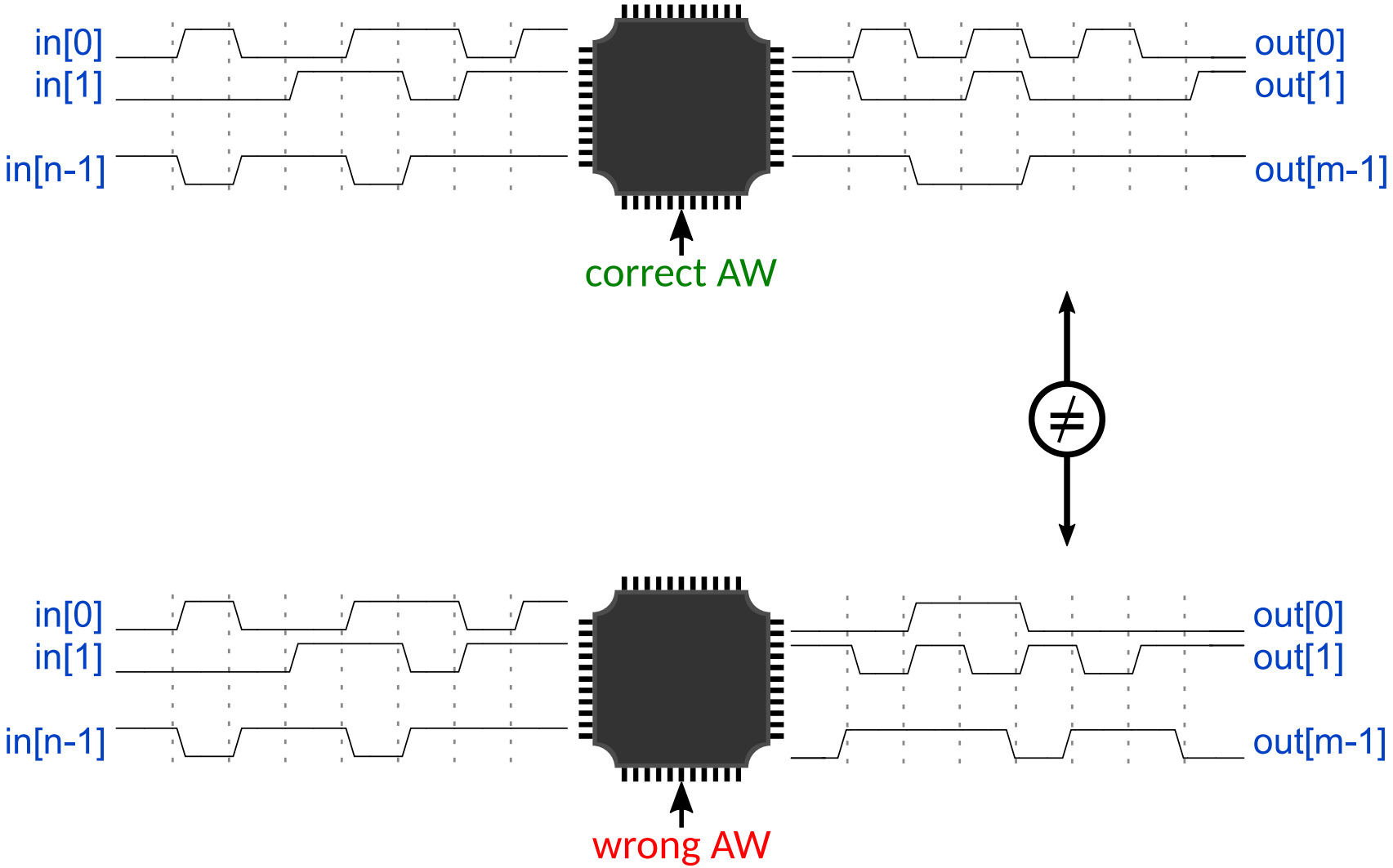
Outline:

1. Computationally-efficient selection heuristics for logic masking
2. Combinational logic locking for very large netlists
3. Lightweight error-correction module for PUF responses
4. Overall integration and demonstrator



Scalable logic masking with centrality indicators

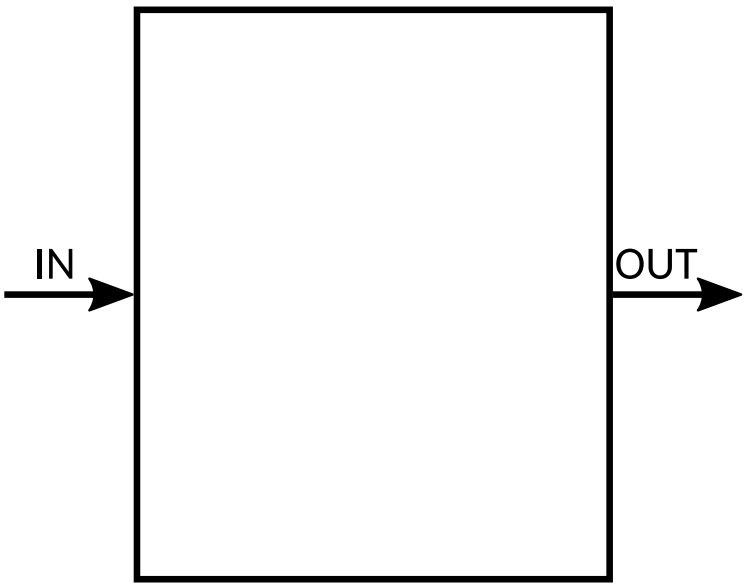
The netlist is modified and an associated AW is obtained [4].



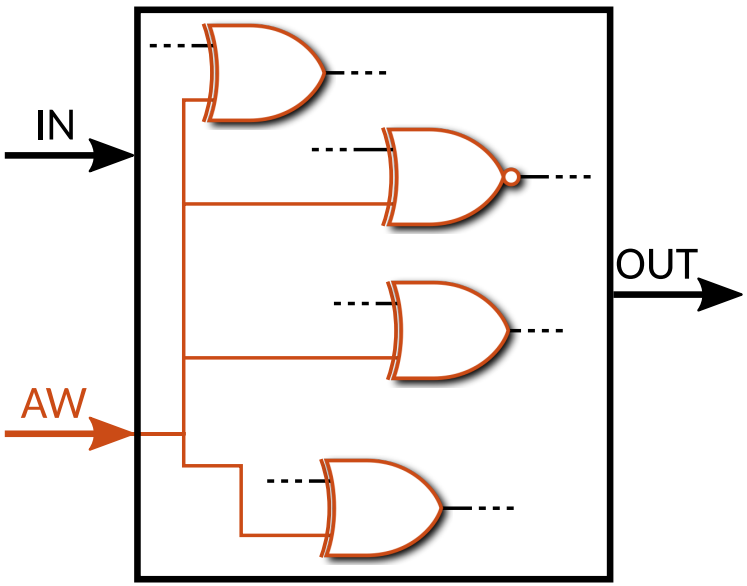
[4] J. A. Roy, F. Koushanfar, and I. Markov. "EPIC: Ending Piracy of Integrated Circuits". *DATE*. 2008, pp. 1069–1074

Principle

Insert **XOR** and **XNOR** gates at **specific** locations in the netlist



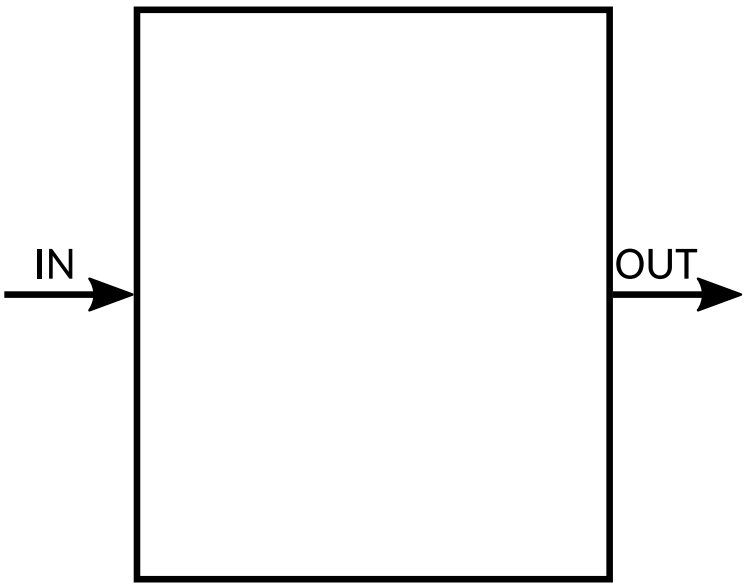
Original netlist



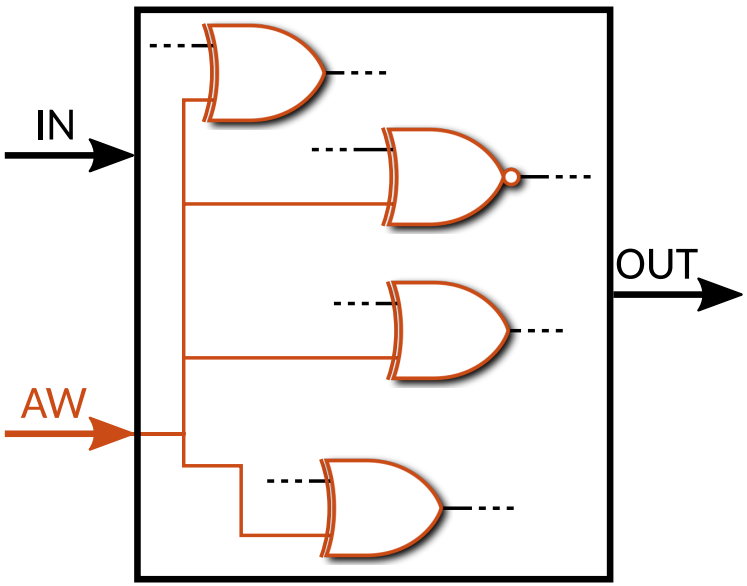
Masked netlist

Principle

Insert **XOR** and **XNOR** gates at **specific** locations in the netlist



Original netlist



Masked netlist

Objective: masking efficiency

The outputs should be as **different** as possible if AW is wrong

Several heuristics exist to select the place of insertion:

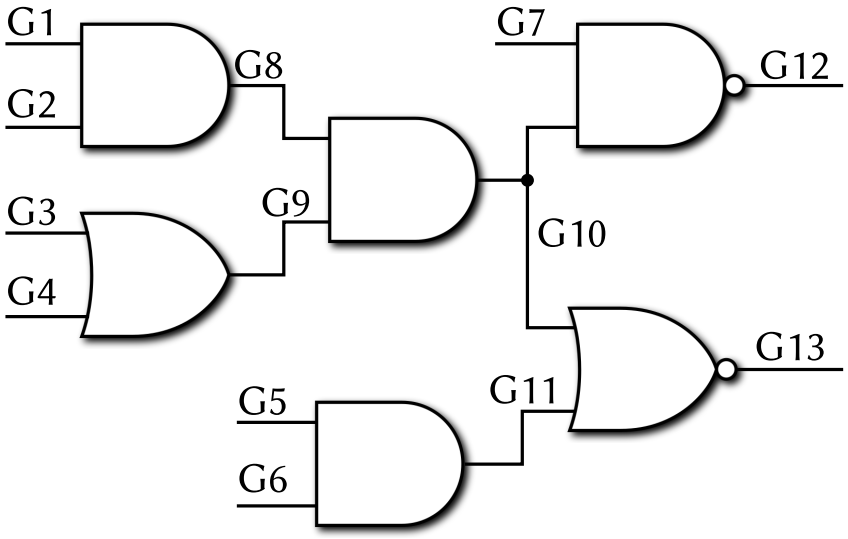
Selection heuristic	Year	**Masking efficiency**	**Computational complexity**
Random [4]	2008	✗	✓
Fan-in/fan-out cones [5]	2009	✗	✓
Fault analysis [6]	2015	✓	✗

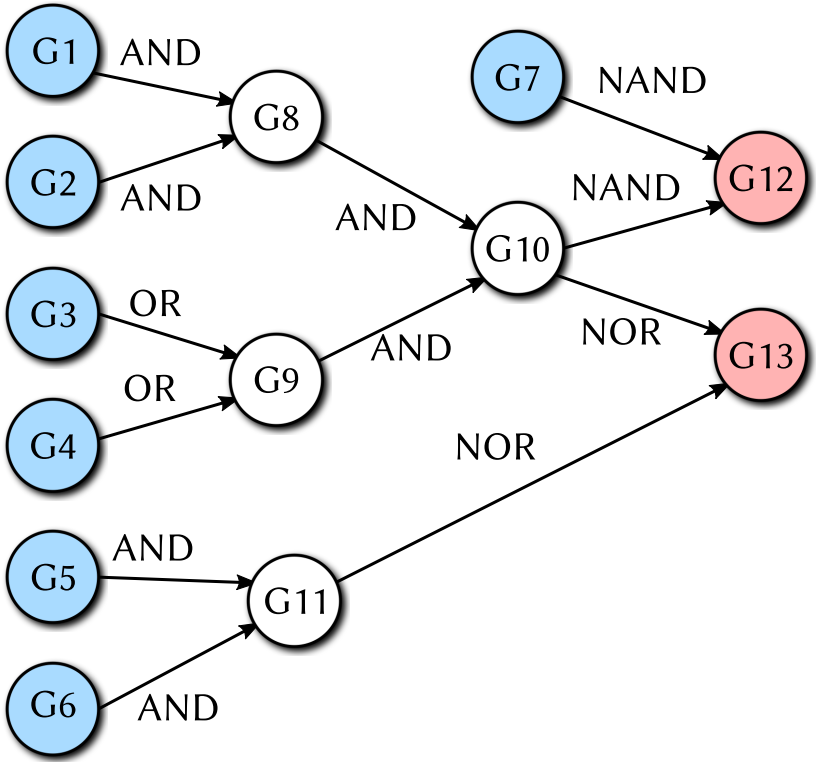
[4] J. A. Roy, F. Koushanfar, and I. Markov. "EPIC: Ending Piracy of Integrated Circuits". *DATE*. 2008, pp. 1069–1074

[5] R. S. Chakraborty and S. Bhunia. "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection". *IEEE Trans. on CAD of IC and Systems* 28.10 (2009), pp. 1493–1502

[6] J. Rajendran et al. "Fault Analysis-Based Logic Encryption". *IEEE Transactions on Computers* 64.2 (2015), pp. 410–424

Graph representation of a gate-level netlist

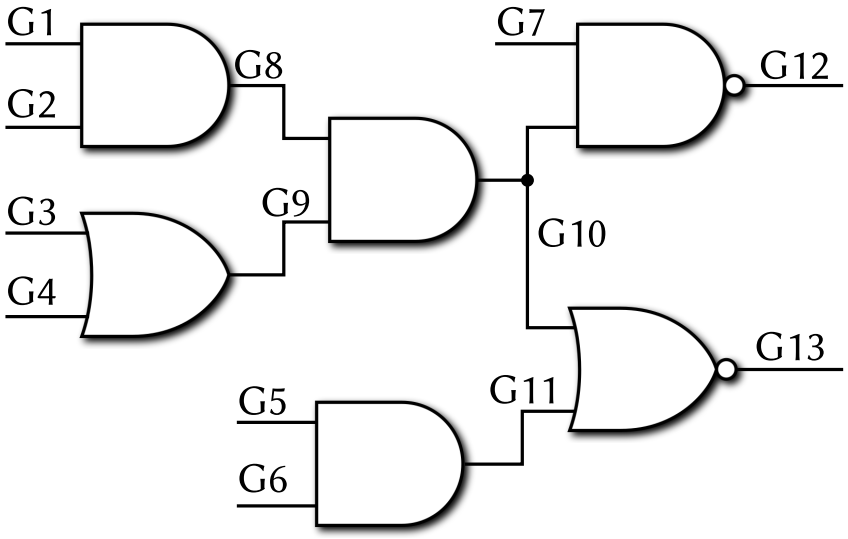


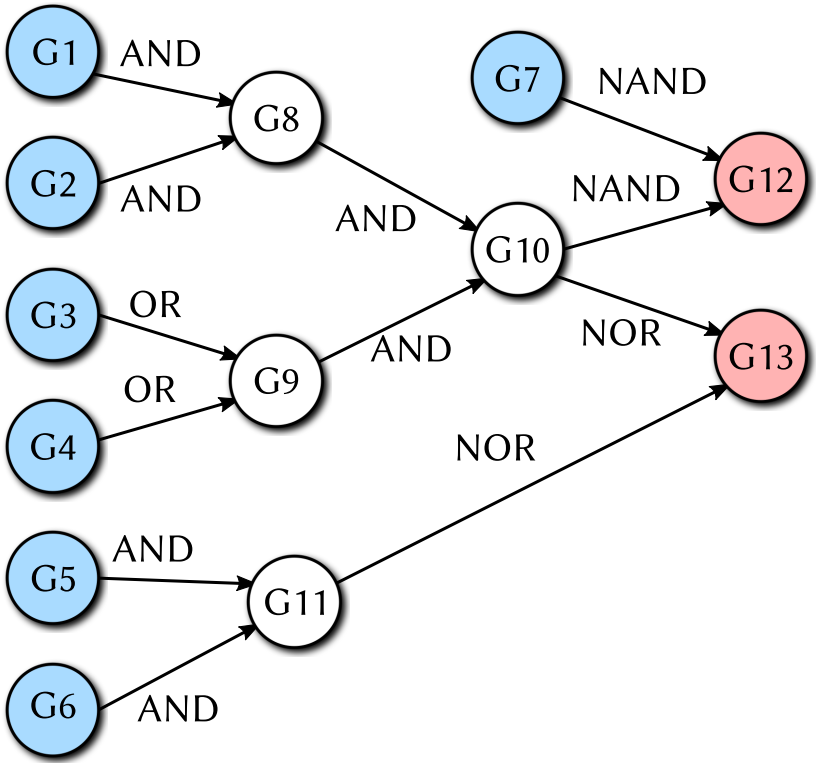


Conversion rules

Wires	$\longrightarrow$	Vertices
Logic gates	$\longrightarrow$	Edges

Graph representation of a gate-level netlist





Conversion rules

Wires	$\longrightarrow$	Vertices
Logic gates	$\longrightarrow$	Edges

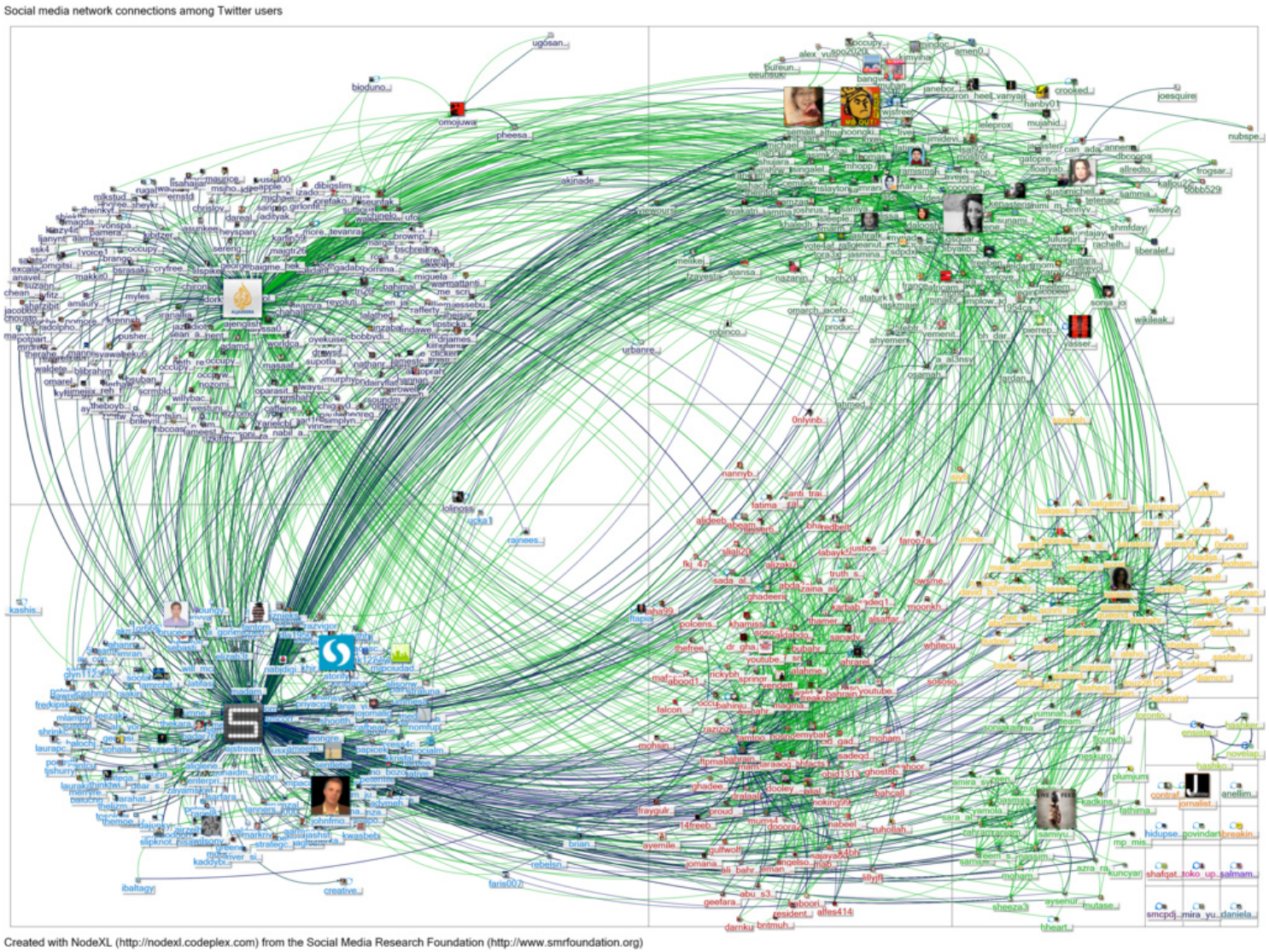
Logic masking heuristic

Insert XOR/XNOR gates on vertices that have the **highest centrality**.

Centrality indicators

Centrality: **Importance** of a given **vertex** inside a graph

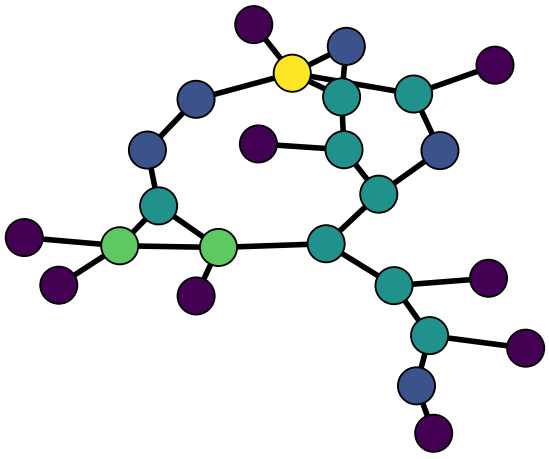
Centrality: **Importance** of a given **vertex** inside a graph



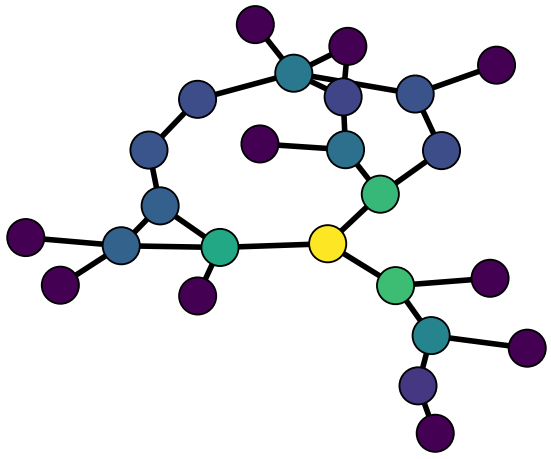
Twitter network graph, Copyright Marc Smith on Flickr under license CC BY 2.0

Comparison of centrality indicators

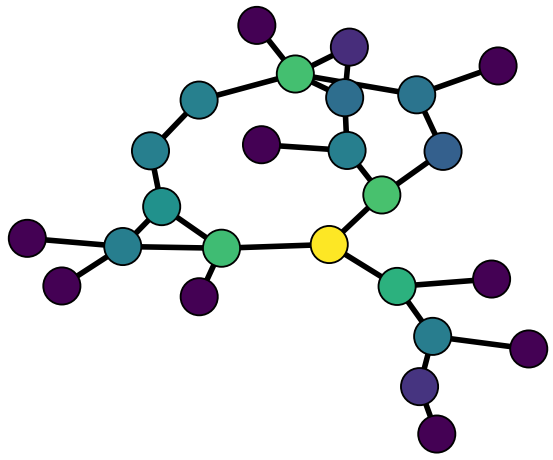
Degree

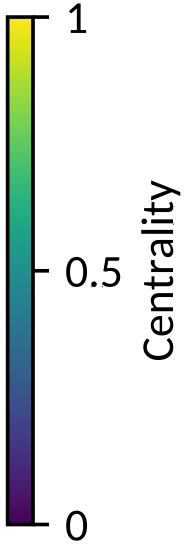


Betwenness

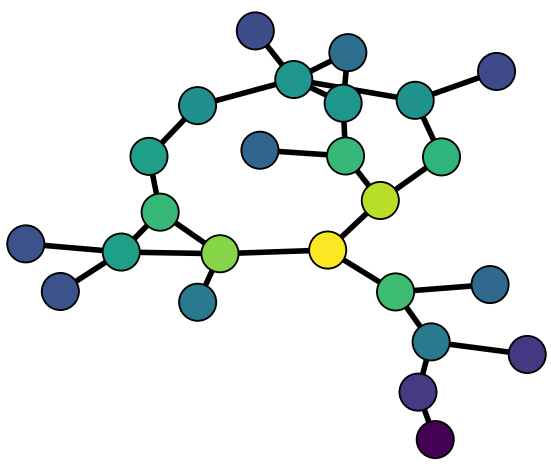


Current-flow betweenness

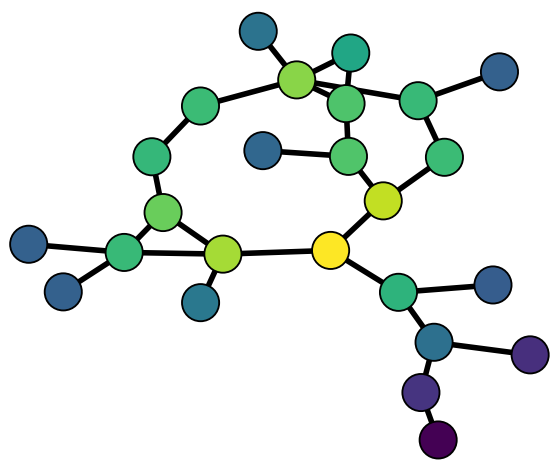




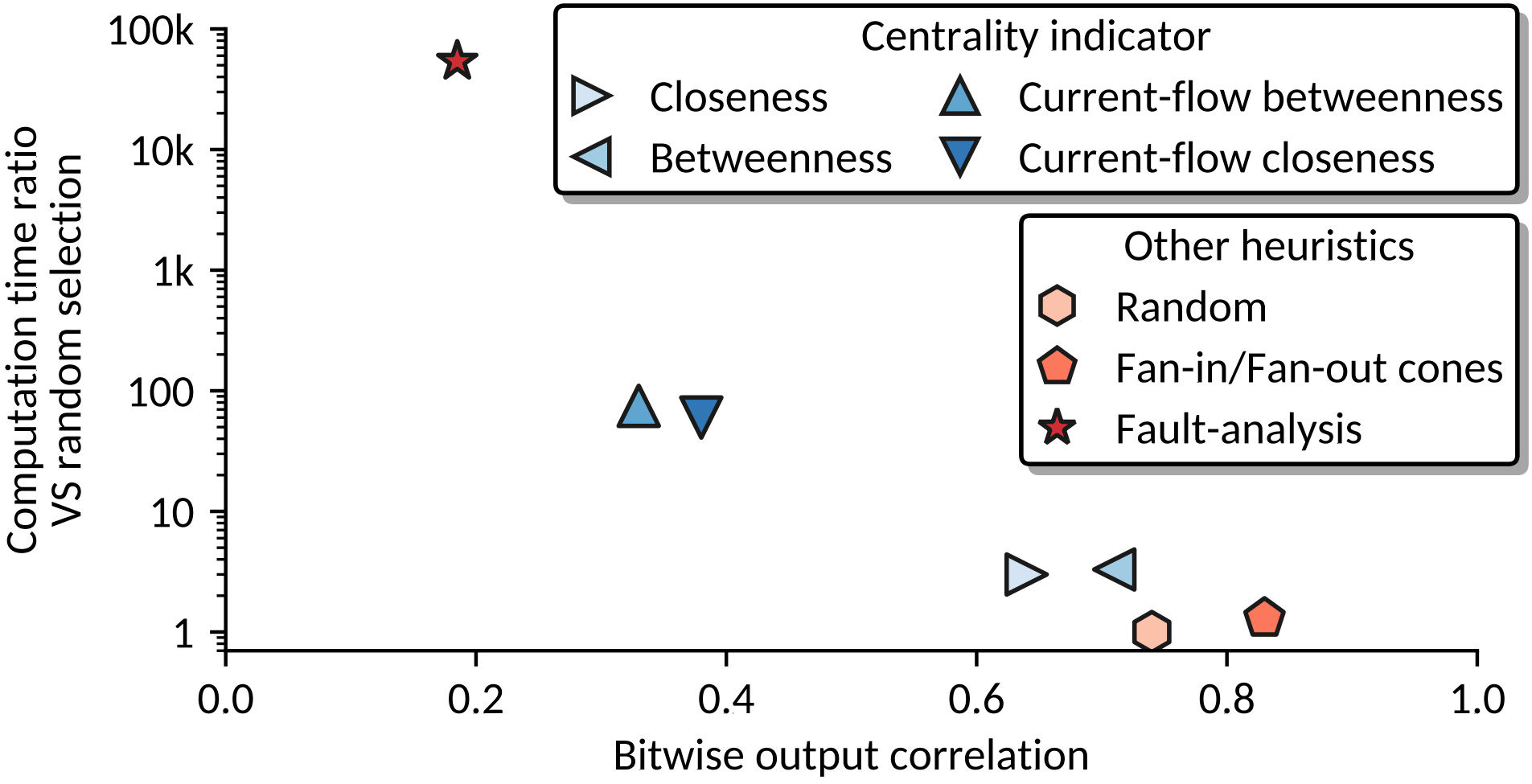
Closeness



Current-flow closeness



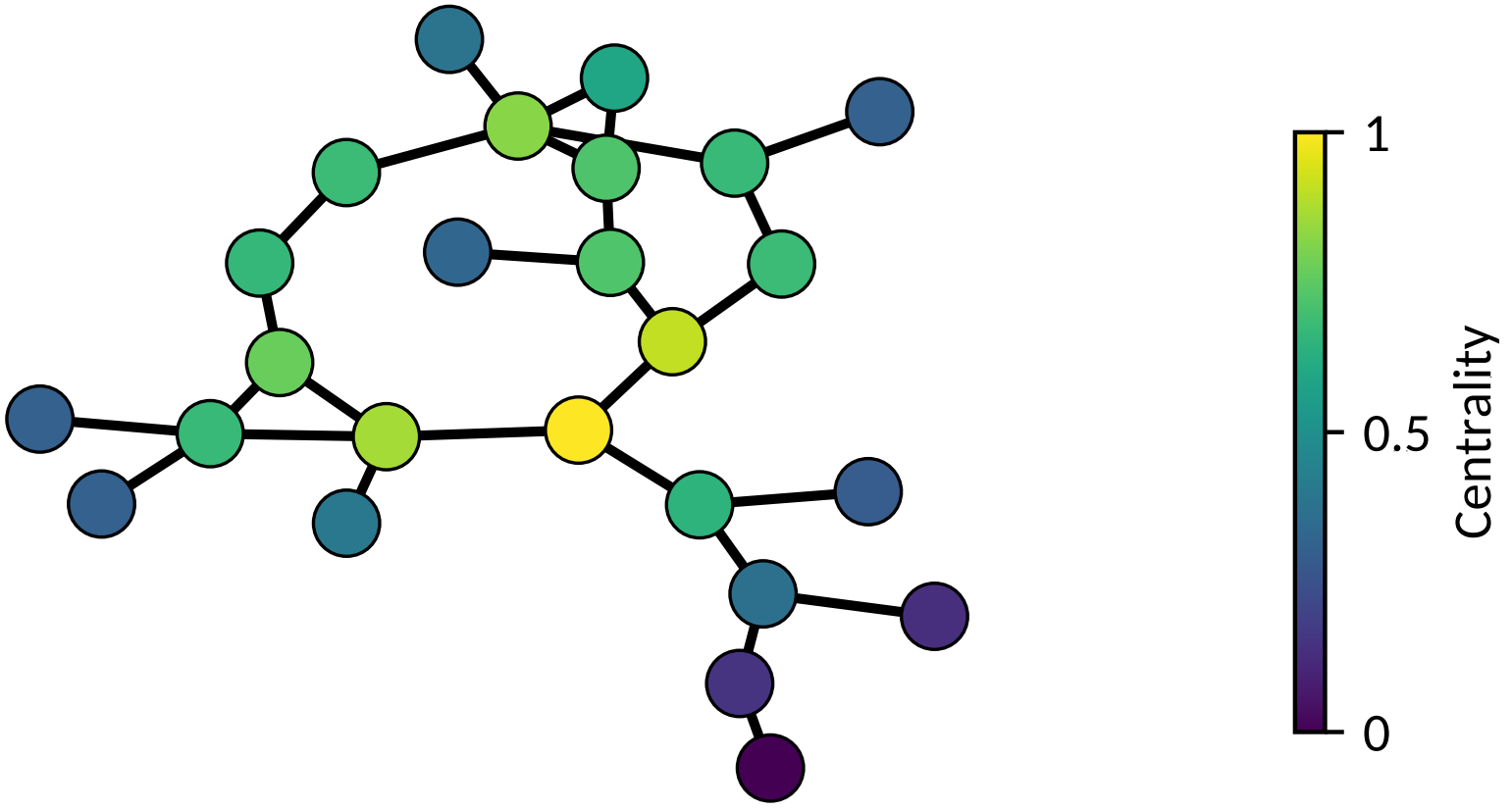
Masking efficiency VS computational complexity

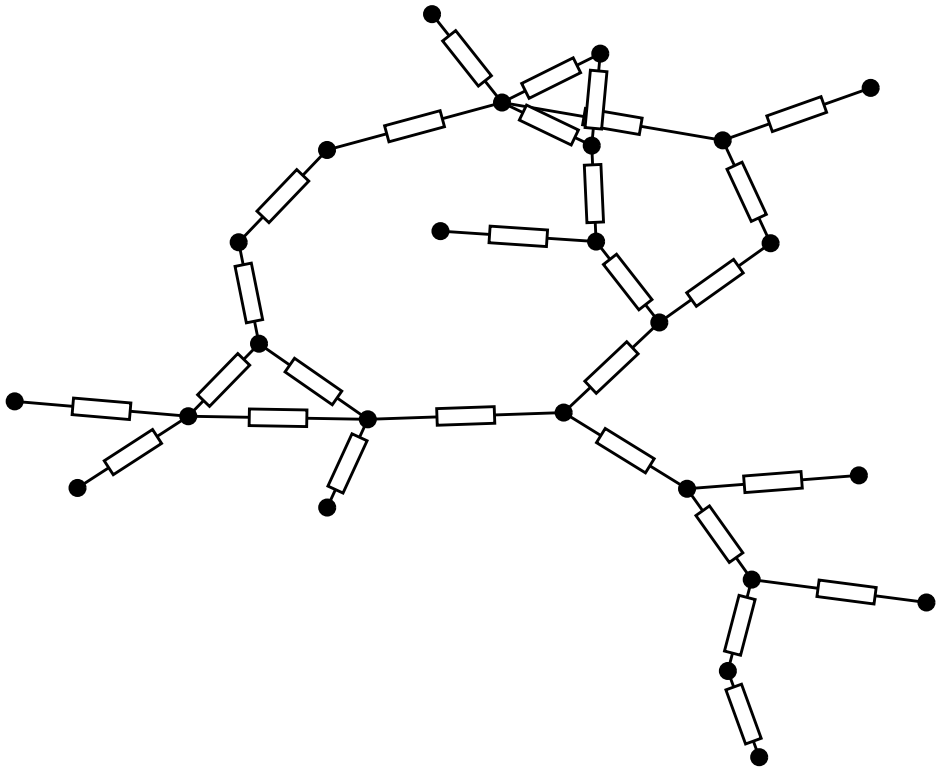


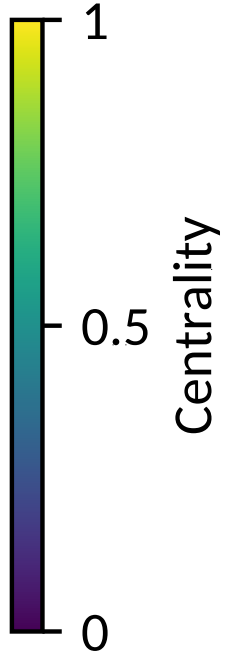
Current-flow closeness

Bitwise correlation is **almost as low** as for the fault analysis-based heuristic, for a **1000x shorter** run-time.

Current-flow closeness centrality



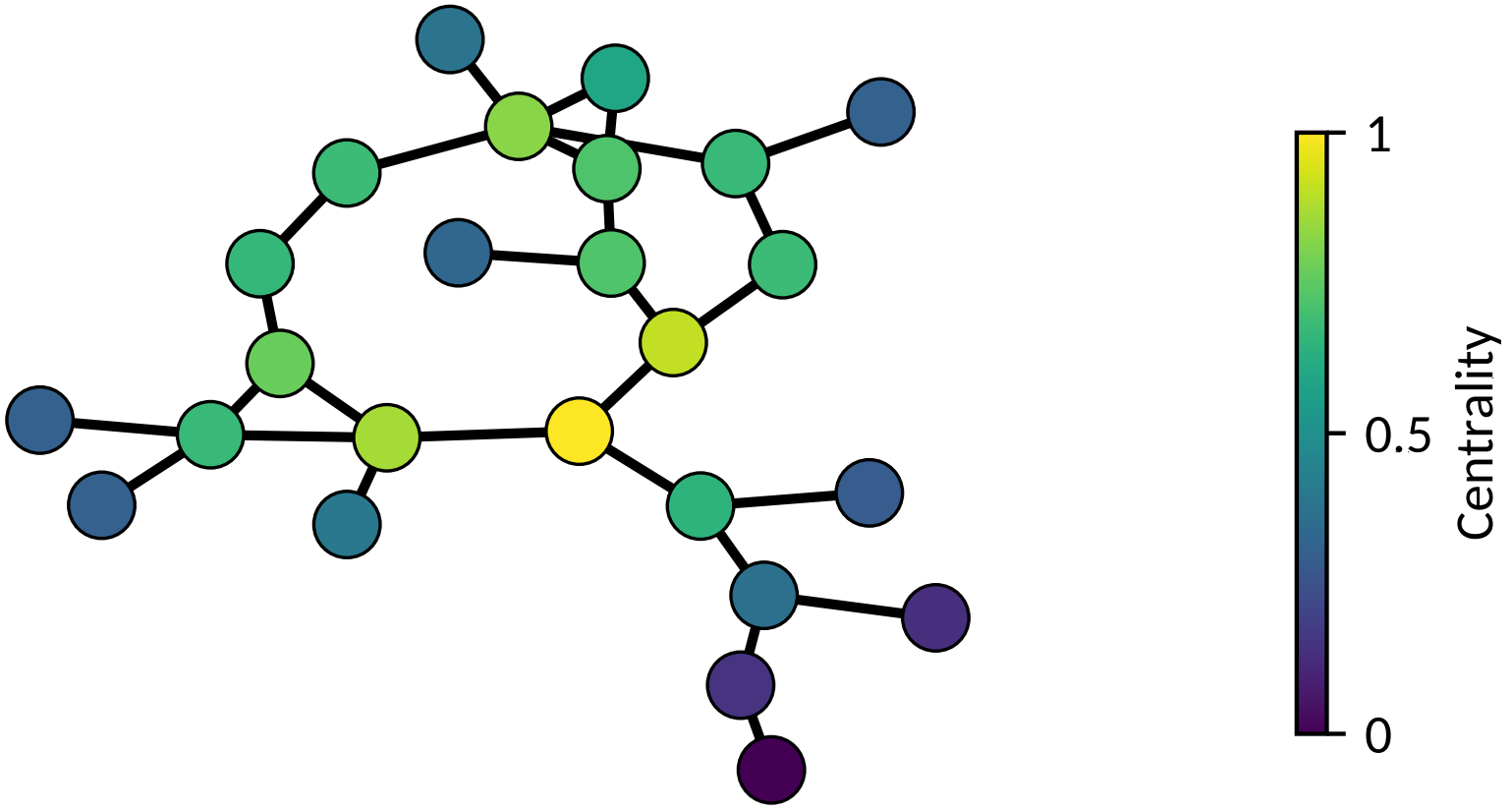




Definition:

Inverse of the sum of effective resistance between the vertex of interest and all the other vertices of the graph.

$$C(v) = \frac{1}{\sum_{y \in V} R_{eff}(v,y)}$$



Definition:

Inverse of the sum of effective resistance between the vertex of interest and all the other vertices of the graph.

$$C(v) = \frac{1}{\sum_{y \in V} R_{eff}(v,y)}$$

Logic masking using centrality indicators: conclusion

Logic masking using **centrality indicators**:

- ✔ better **trade-off** than existing heuristics between masking efficiency and computational complexity,
- ✔ only **efficient** masking selection heuristic for **real-life** netlists.

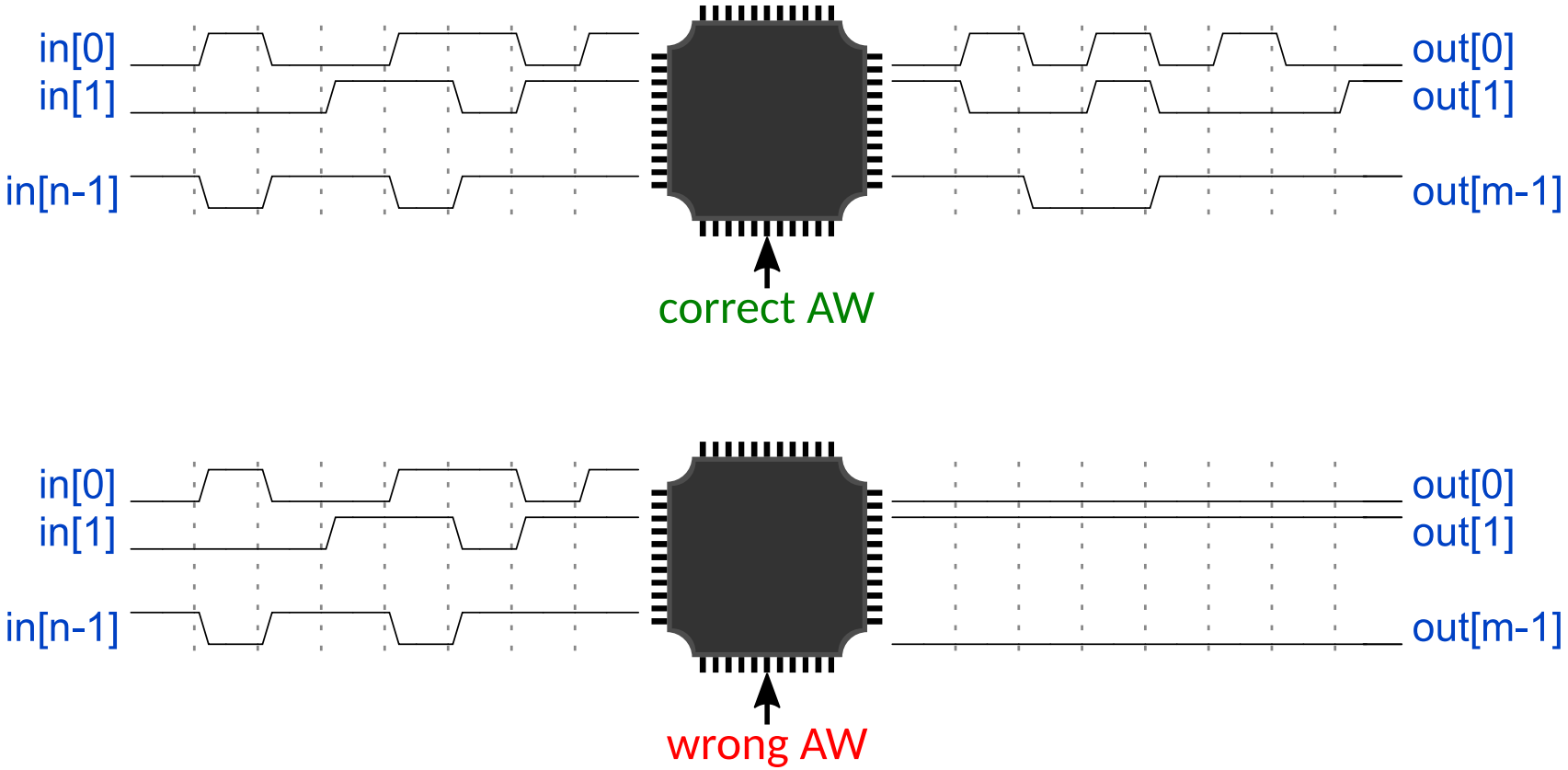
- ▪▪▪ could be **parallelized**.

Associated publication:

B. Colombier, L. Bossuet, and D. Hély. "Centrality Indicators for Efficient and Scalable Logic Masking". *ISVLSI*. 2017, pp. 98–103

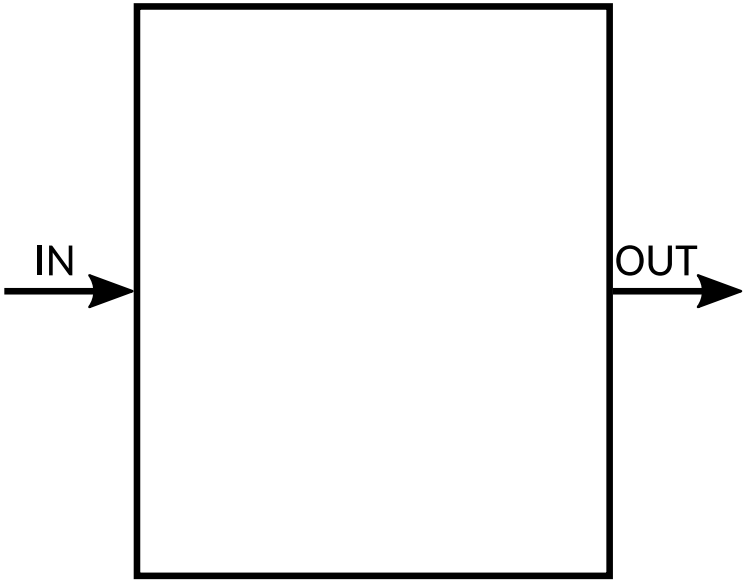
Combinational logic locking

The netlist is modified and an AW (activation word) is obtained.

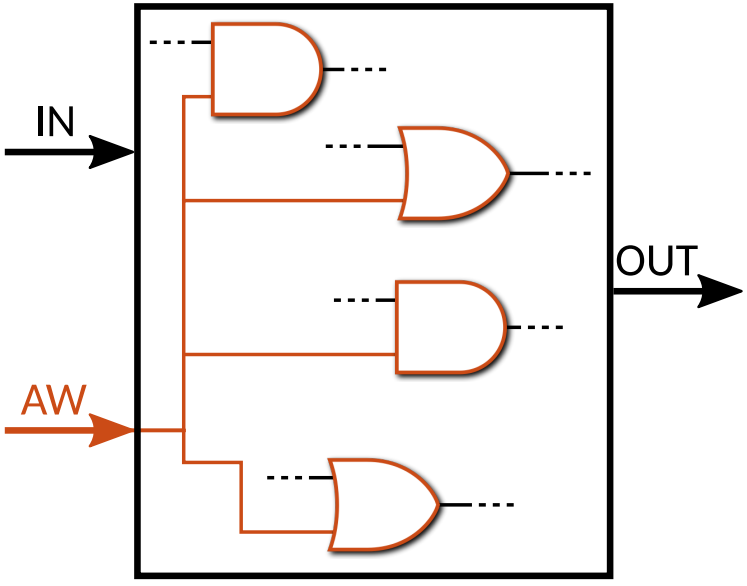


Principle

Insert **AND** and **OR** gates at **specific** locations in the netlist



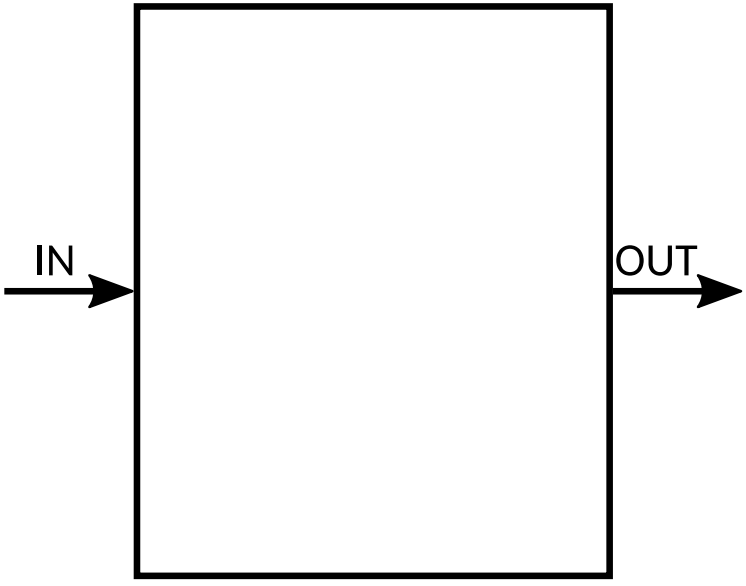
Original netlist



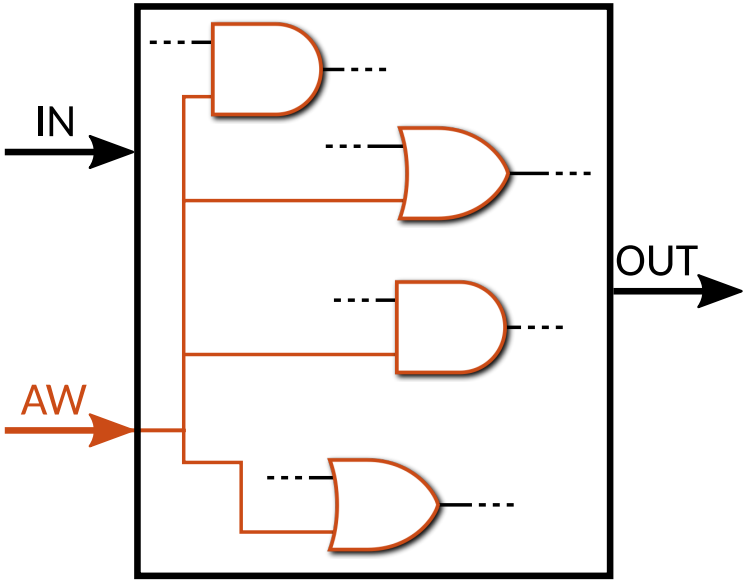
Locked netlist

Principle

Insert **AND** and **OR** gates at **specific** locations in the netlist



Original netlist

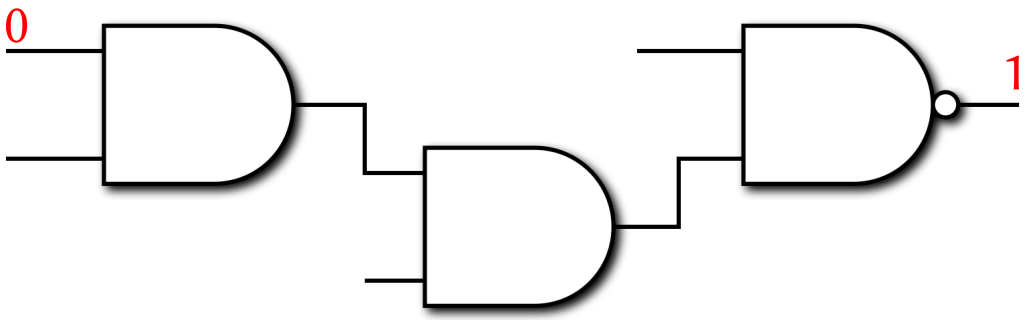


Locked netlist

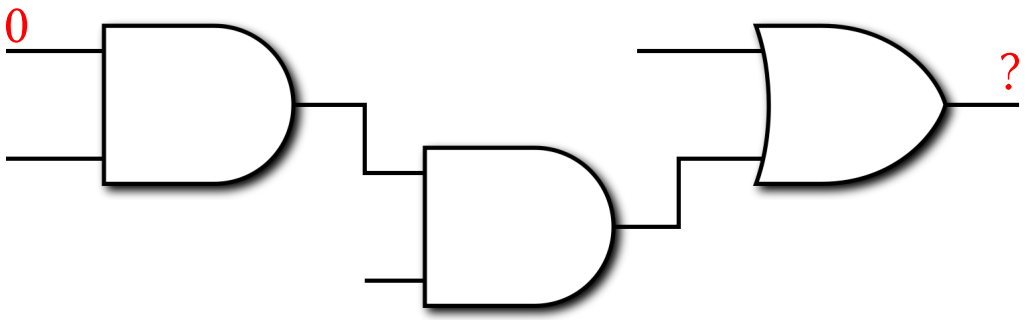
Objective

The outputs should be forced to a **fixed** logic value.

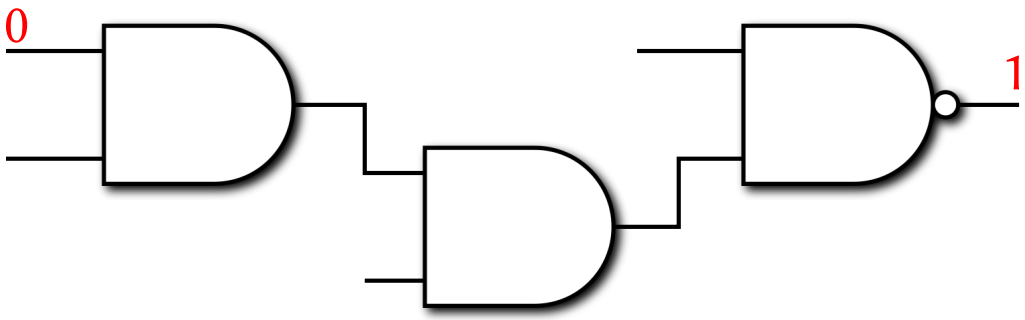
Identify nodes inside the netlist that propagate a locking value.



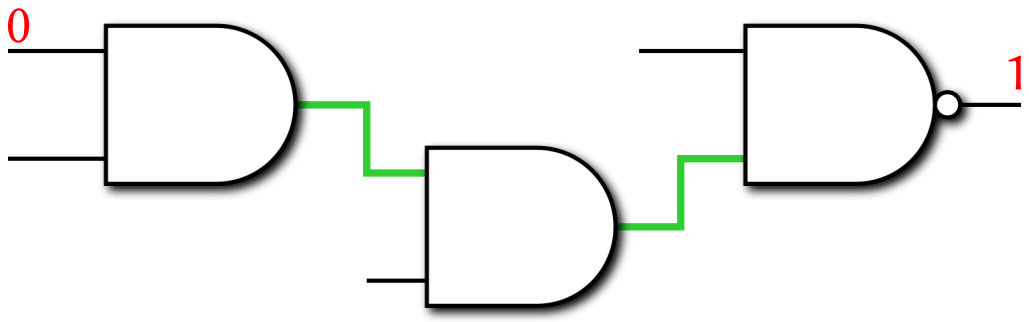
Identify nodes inside the netlist that propagate a locking value.



Identify nodes inside the netlist that propagate a locking value.

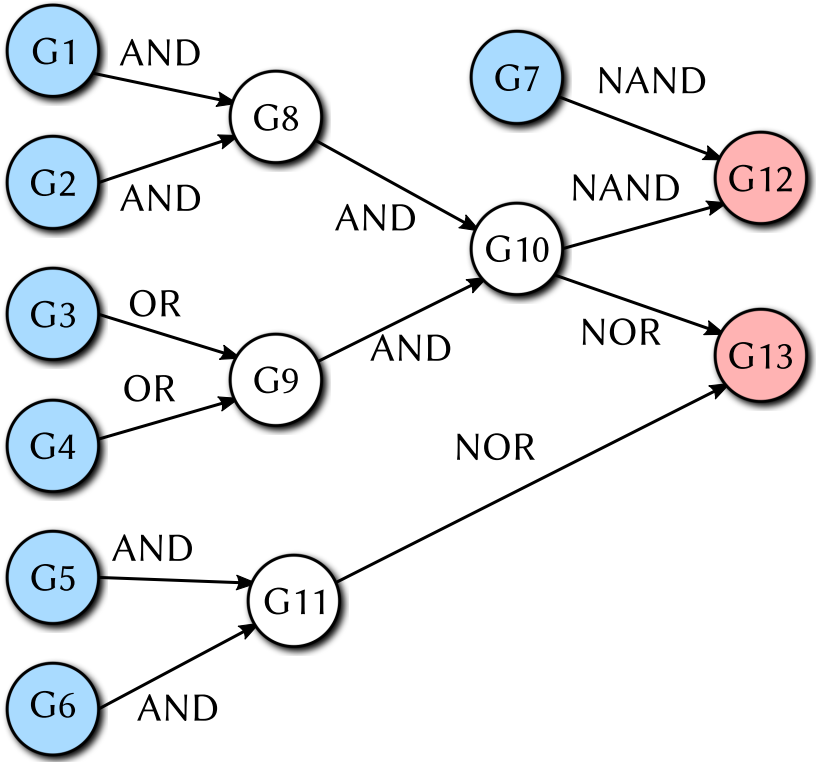


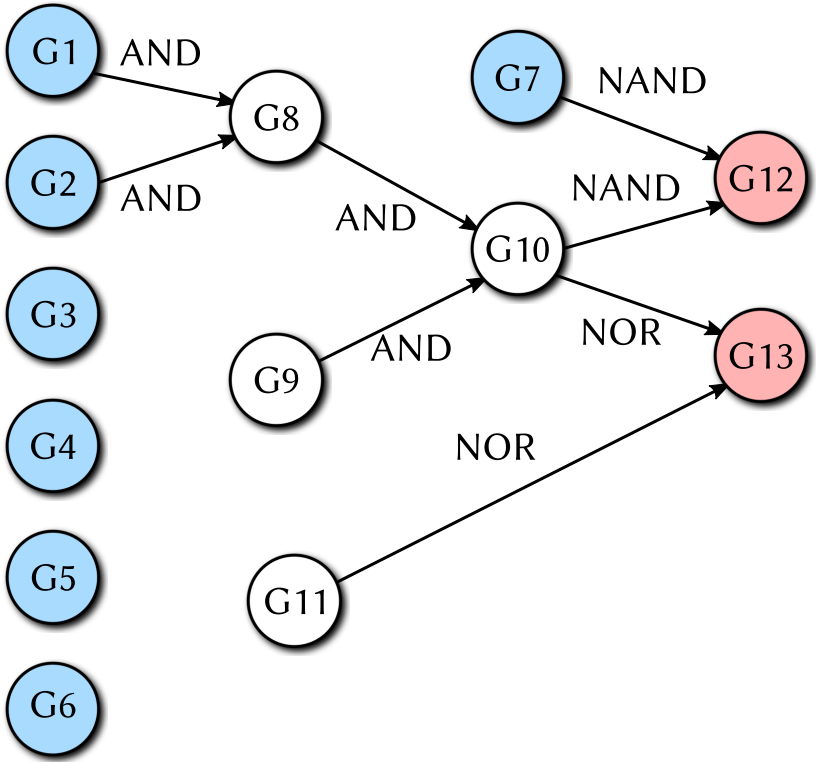
Identify nodes inside the netlist that propagate a locking value.



Condition for propagating a locking value

The node is forced to the controlling value of the next gate.

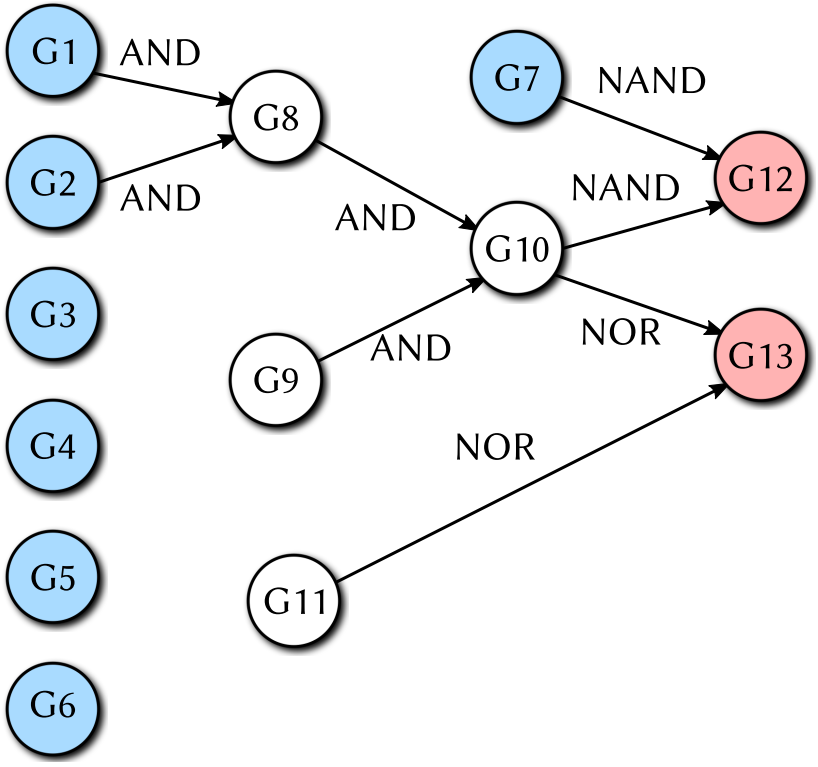


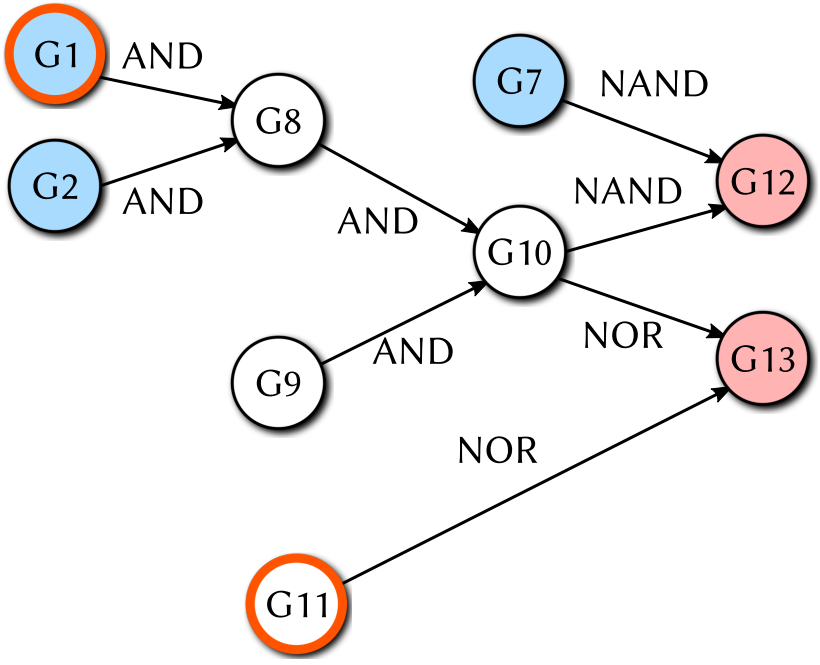


Processing

Remove the **incoming edges** of vertices that are associated with nodes that **cannot propagate** a locking value.

Selection of the nodes to lock

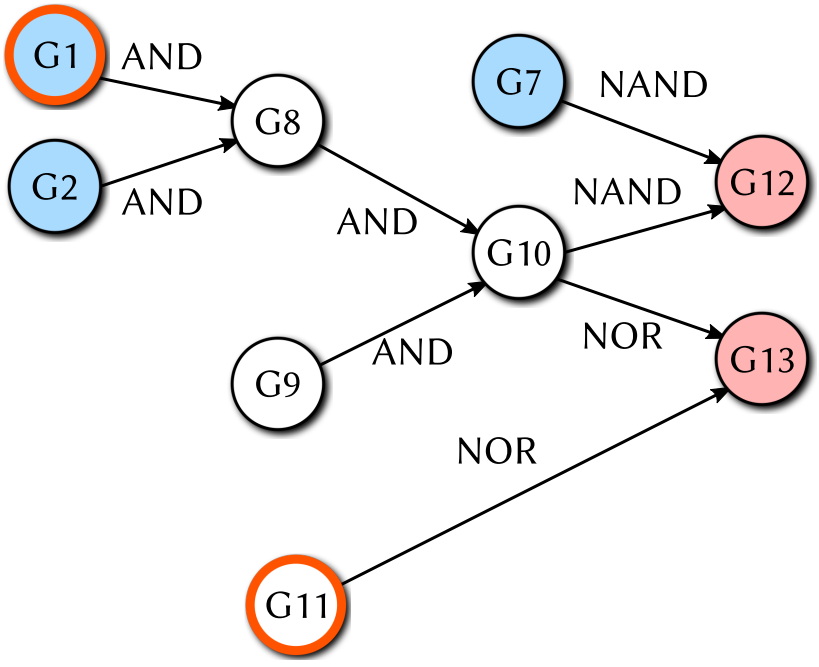


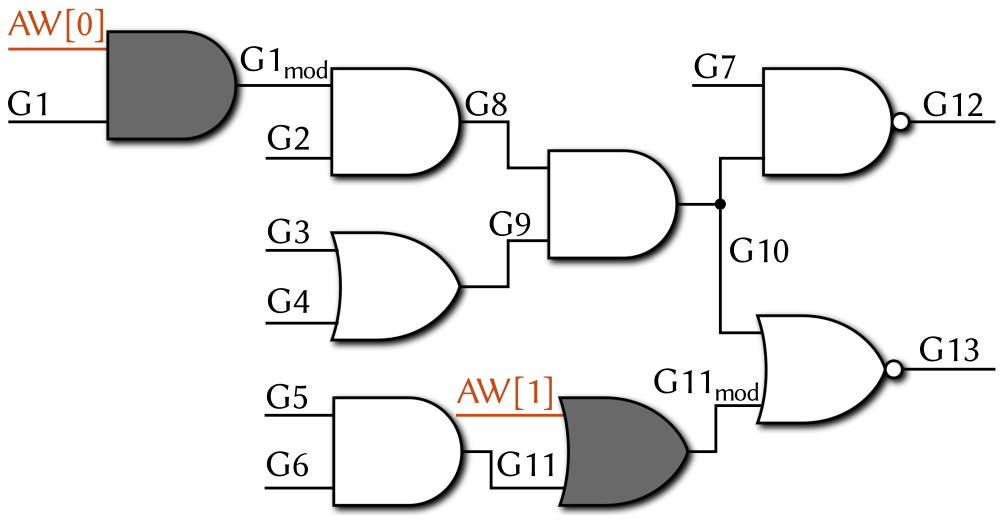


Selection

Select the nodes that are as **far** as possible from the outputs and lock as **many** outputs as possible.

Locking gates insertion



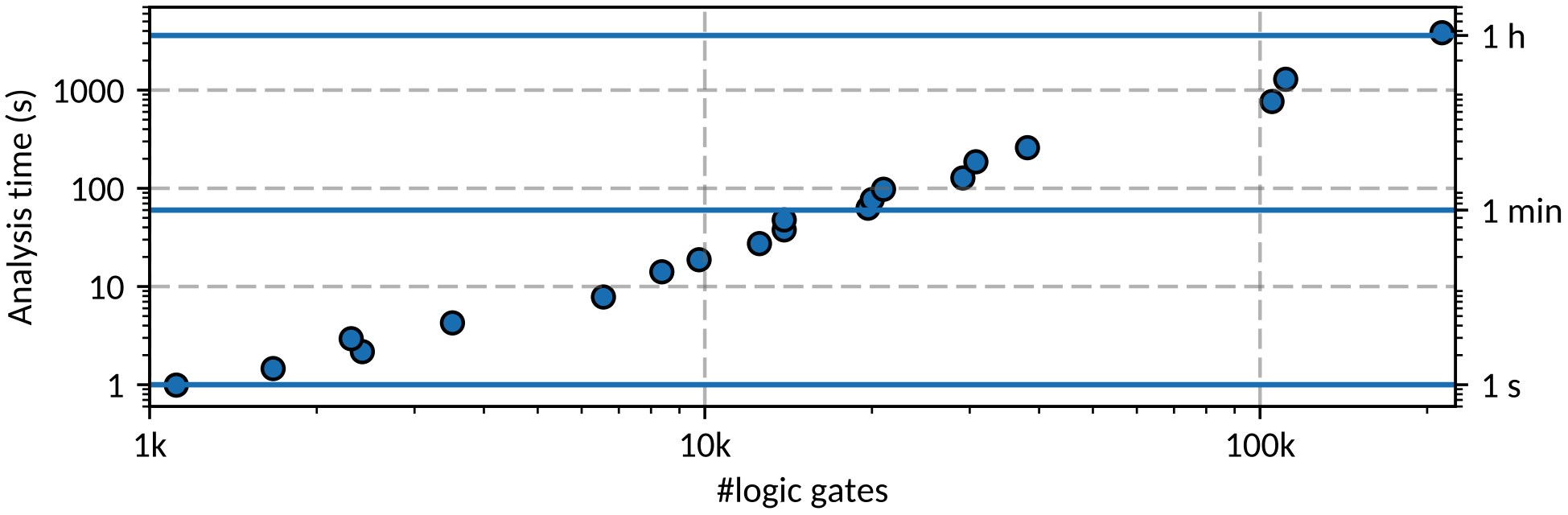


Choice of the type of locking gate

- Force to 0: insert an **AND** gate,
- Force to 1: insert an **OR** gate.

Experimental results: processing time

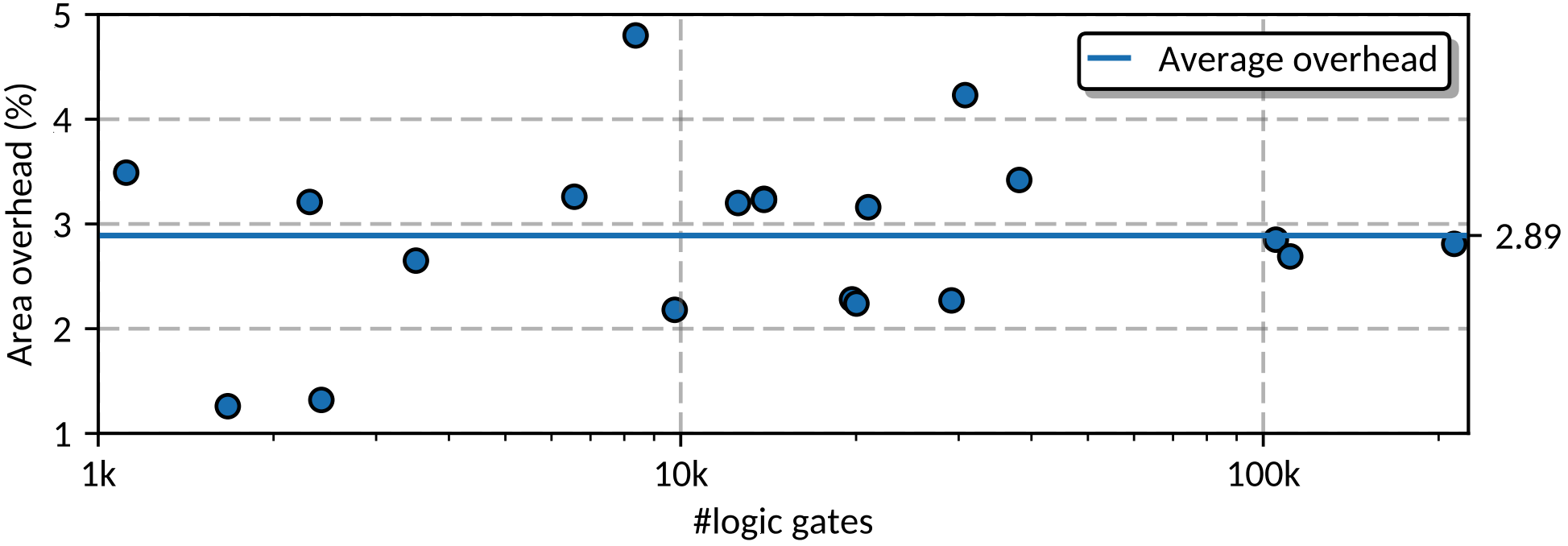
ITC'99 benchmarks: 1k to 200k gates.



Netlists of up to **20k** gates are processed in less than a **minute**.

Netlists of up to **200k** gates are processed in less than an **hour**.

Experimental results: area overhead



Locking **all** the outputs requires an area overhead **below 3%.**

Combinational logic locking: conclusion

Combinational logic locking:

- ✔ is the **only** method for logic modifications that can handle **very large** netlists (**200k logic gates** in less than **one hour**),
- ✔ has a **low** area overhead below **3%**.
- ✔ offers **obfuscation** possibilities: interleaving, NAND/NOR gates.

Associated publications:

B. Colombier, L. Bossuet, and D. Hély. "From Secured Logic to IP Protection". *Elsevier Microprocessors and Microsystems* 47 (2016), pp. 44–54

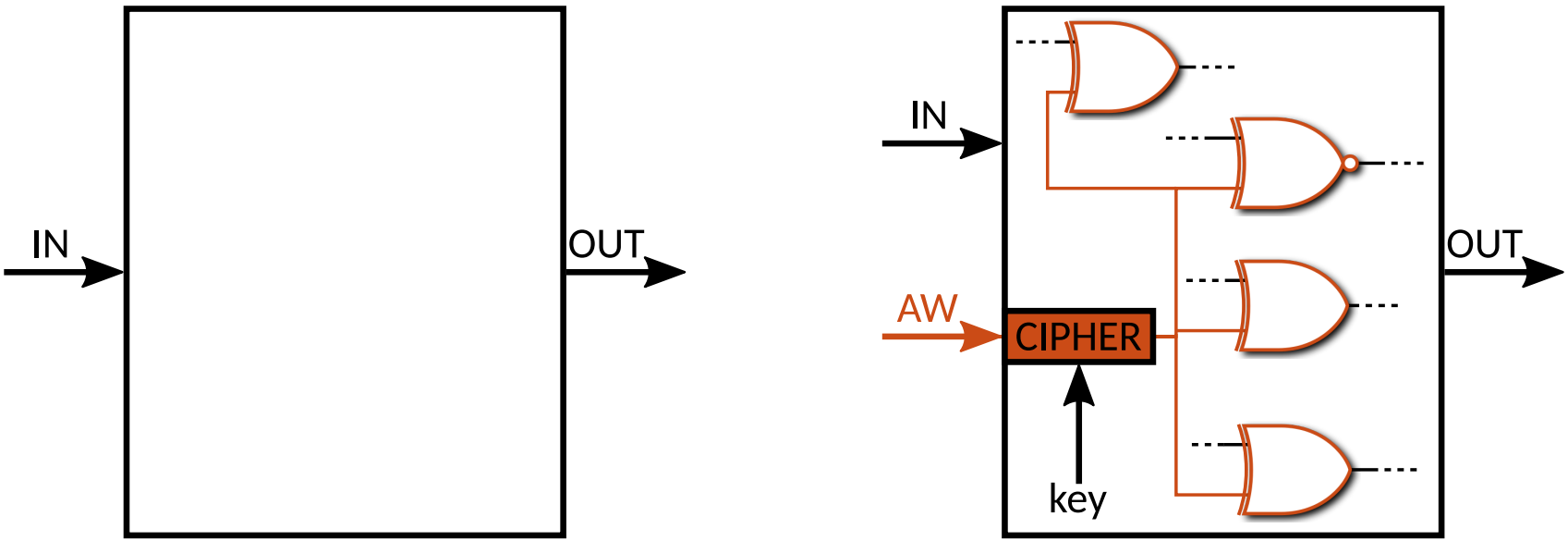
B. Colombier, L. Bossuet, and D. Hély. "Reversible Denial-of-Service by Locking Gates Insertion for IP Cores Design Protection". *ISVLSI*. 2015, pp. 210–215

Efficient masking methods are vulnerable to attacks targetting **AW**:

- Hill-climbing [7]
- SAT [8]

Solution

Not expose AW input, drive it with the output of a **cipher**.

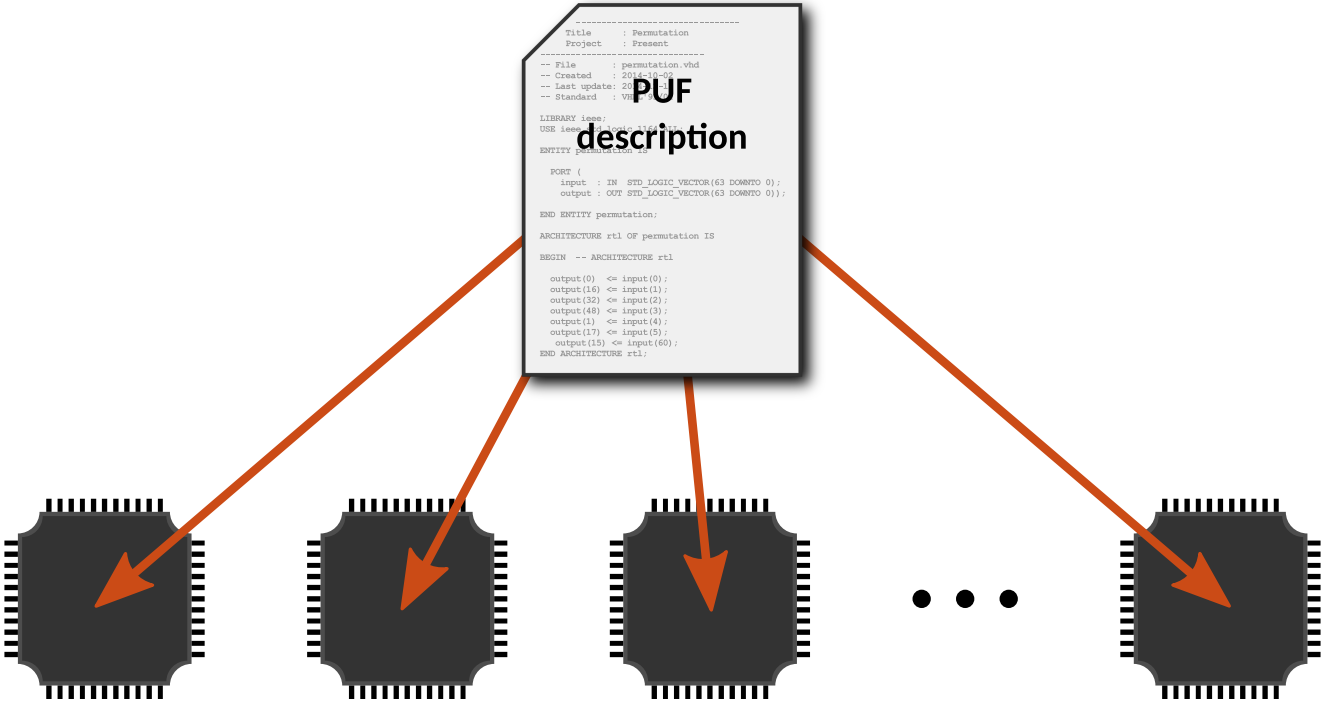


[7] S. M. Plaza and I. L. Markov. "Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking". *IEEE Trans. on CAD of Integrated Circuits and Systems* 34.6 (2015), pp. 961–971

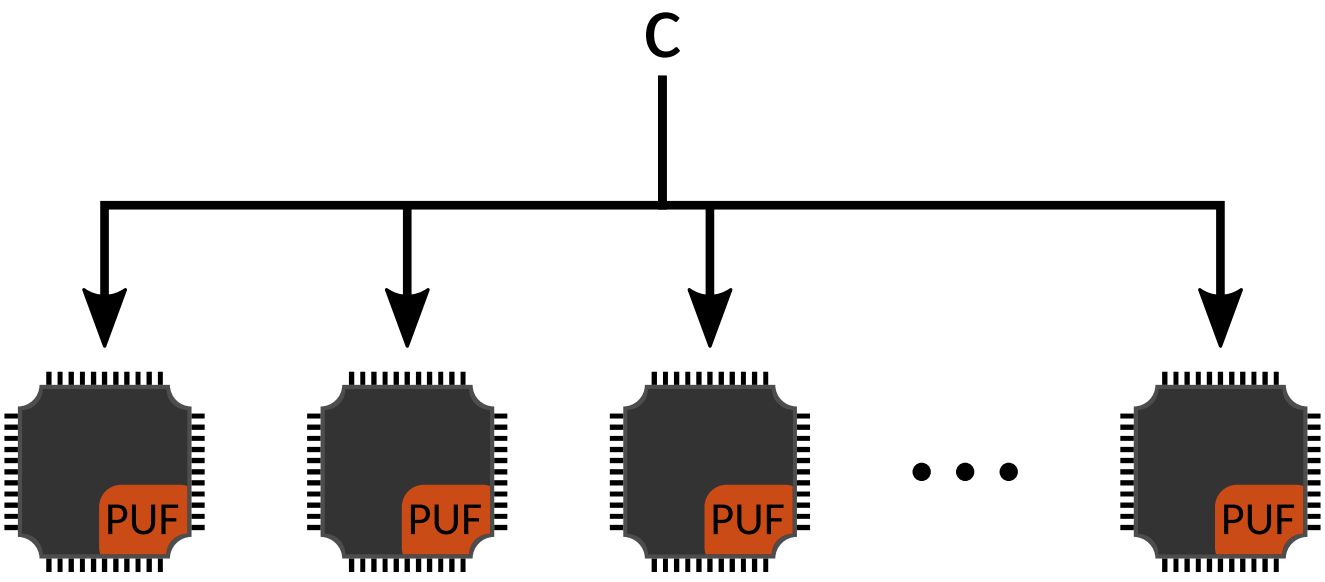
[8] P. Subramanyan, S. Ray, and S. Malik. "Evaluating the security of logic encryption algorithms". *HOST*. 2015, pp. 137–143

Key reconciliation protocol for error correction in PUF responses

PUFs are the **fingerprints** of integrated circuits.

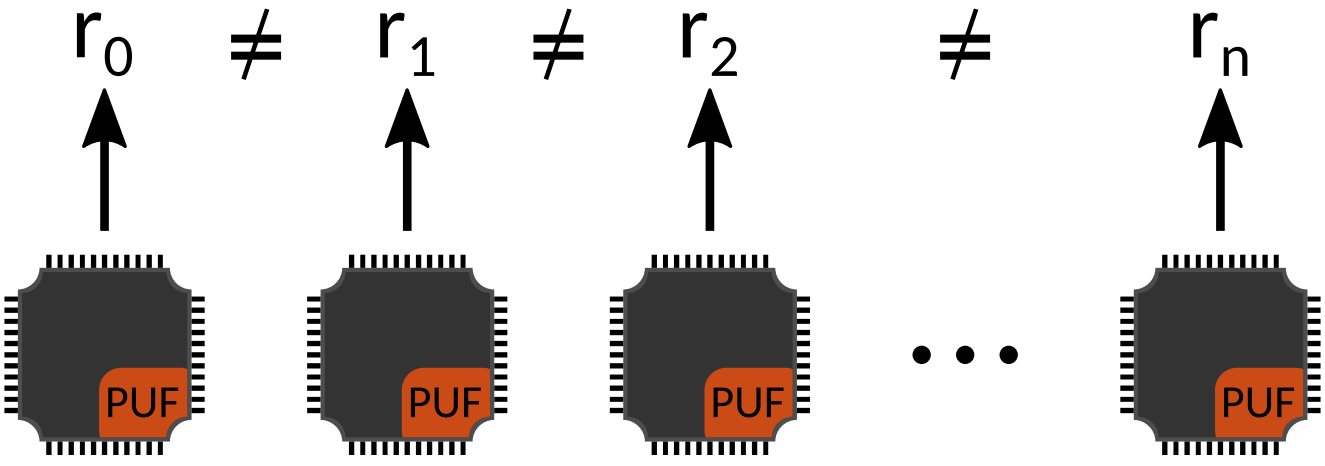


PUFs are the **fingerprints** of integrated circuits.

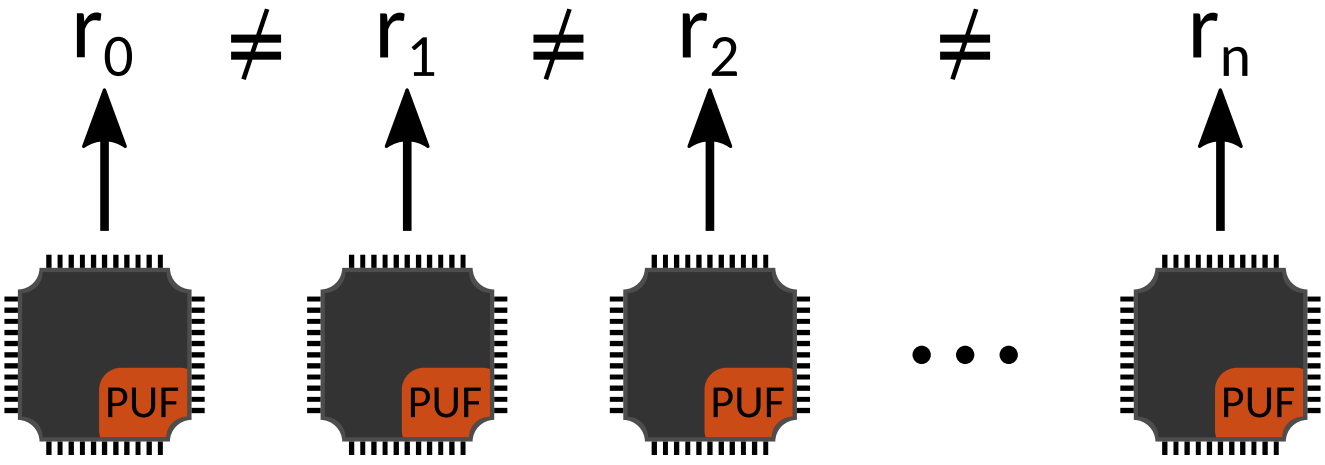


Physical Unclonable Functions (PUFs)

PUFs are the **fingerprints** of integrated circuits.



PUFs are the **fingerprints** of integrated circuits.



Uniqueness property

Because of **random** manufacturing process variations, r is **different** from one PUF instance to the other.

Problem

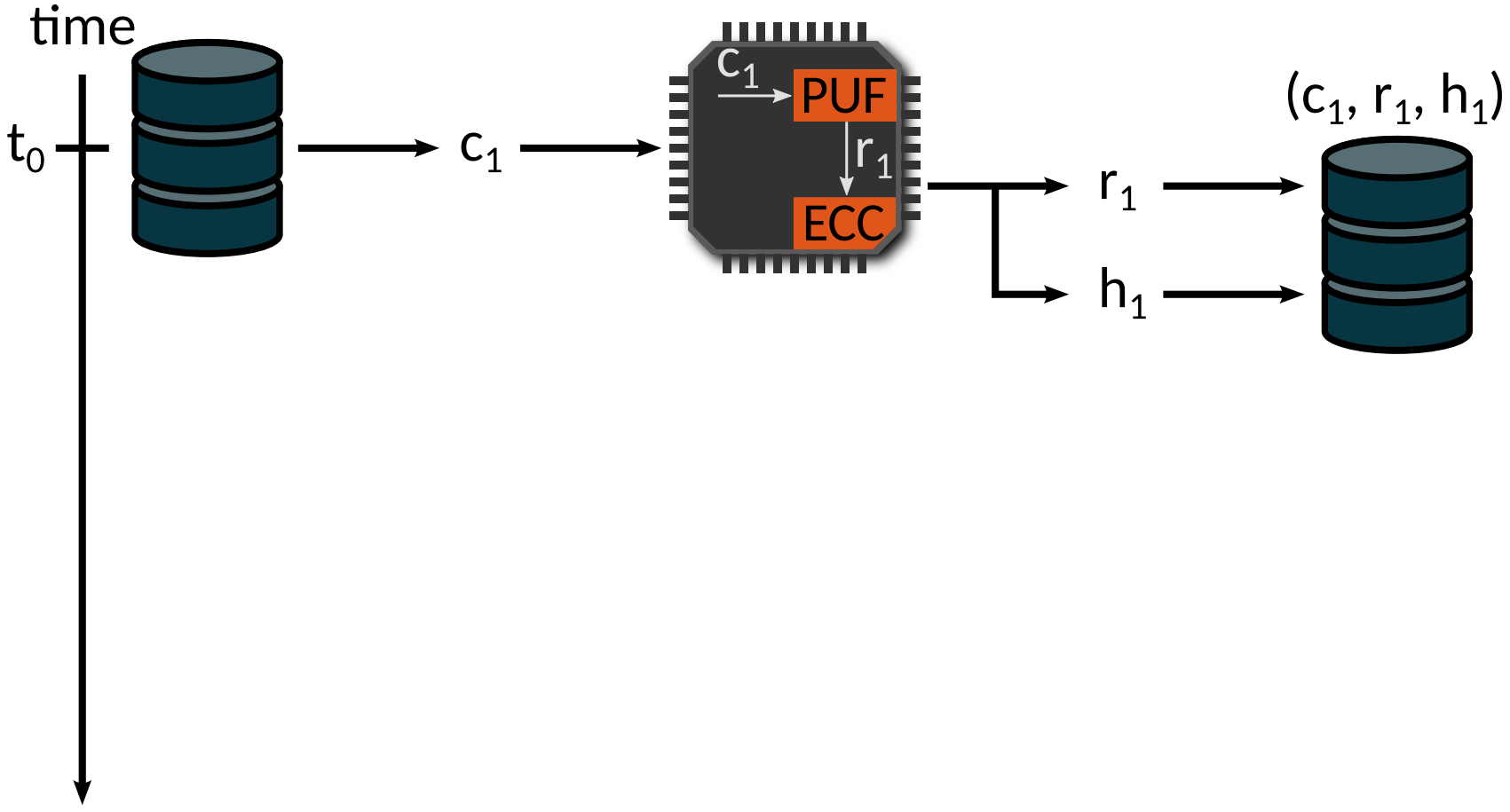
PUF responses to the **same** challenge **change** over time.

This variation depends on multiple parameters:

- PUF architecture,
- Process node,
- Aging,
- Temperature,
- Environment,
- etc.

Solution

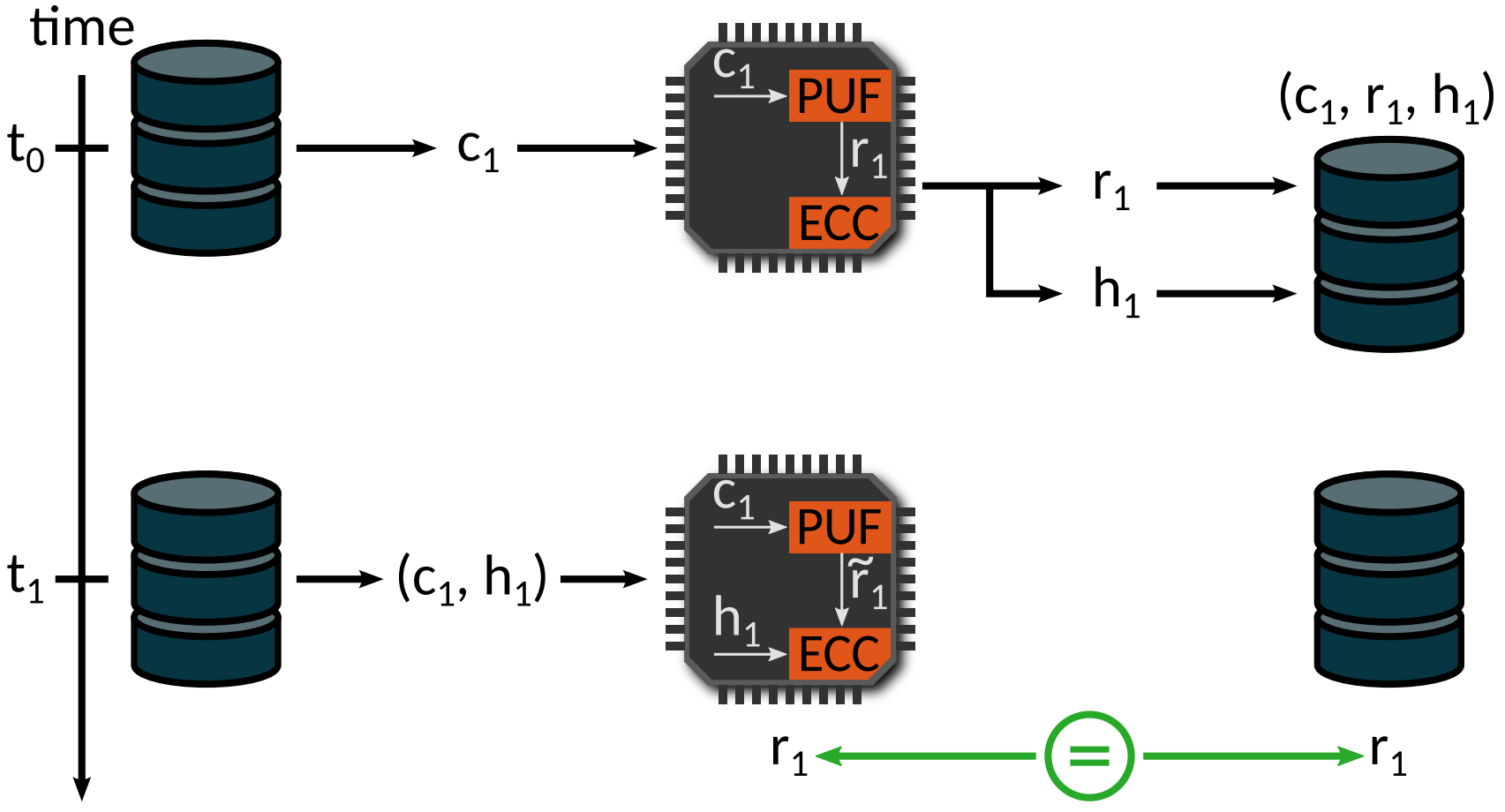
Apply a technique of **error correction** to the PUF response



Error correction for PUF responses

Solution

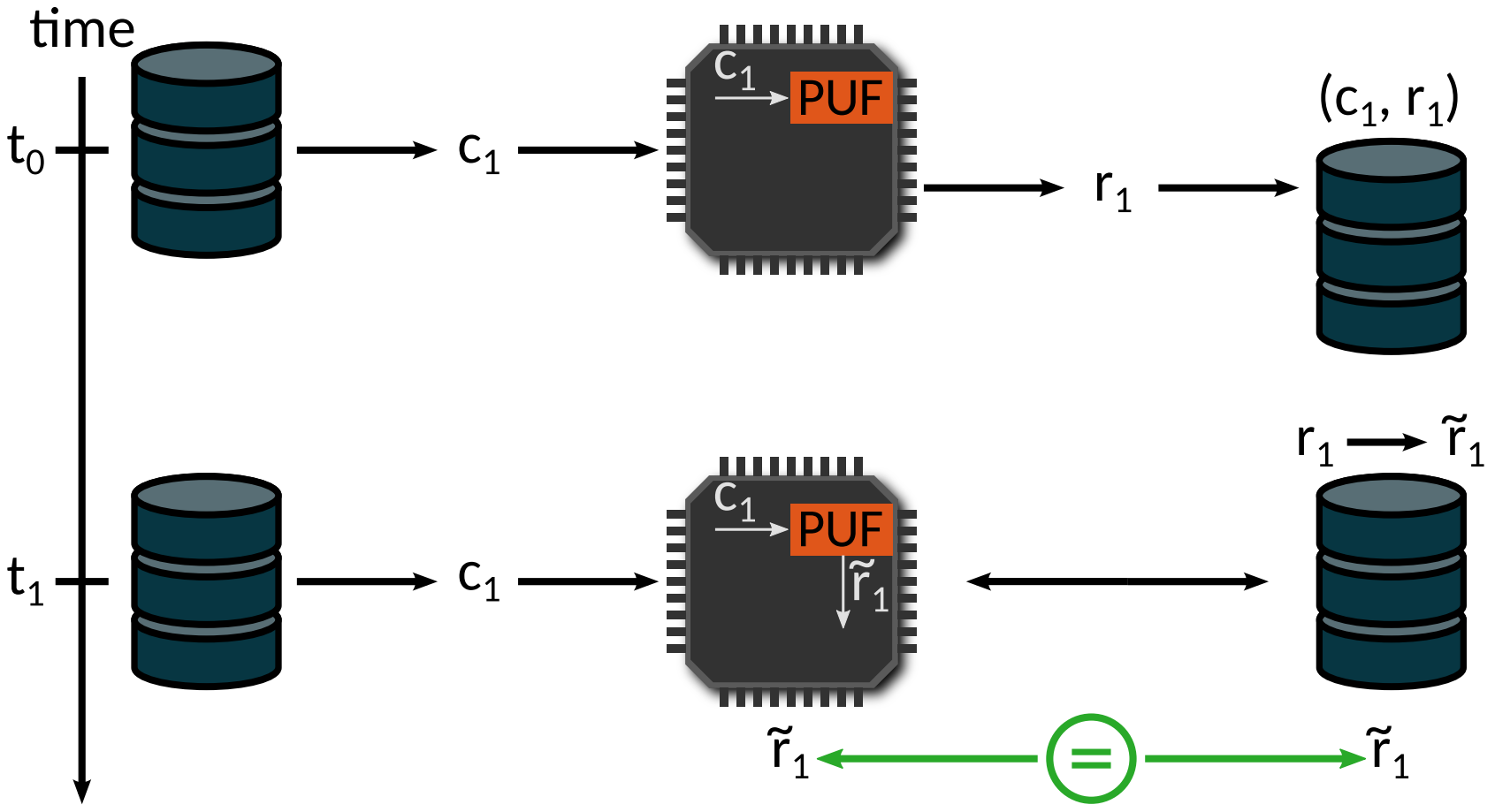
Apply a technique of **error correction** to the PUF response



Error correction for PUF responses

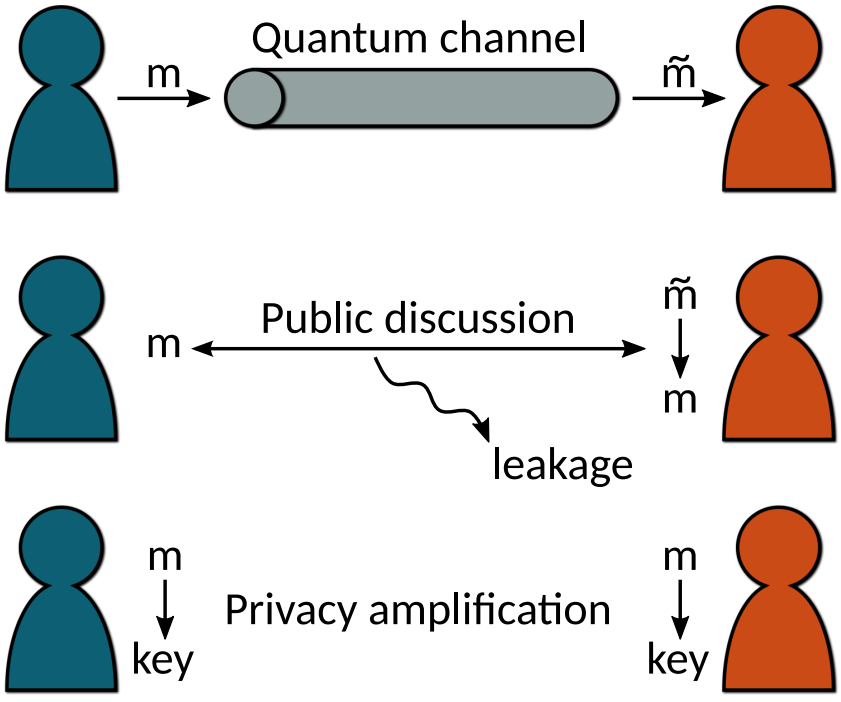
Solution

Apply a technique of **error correction** to the PUF response



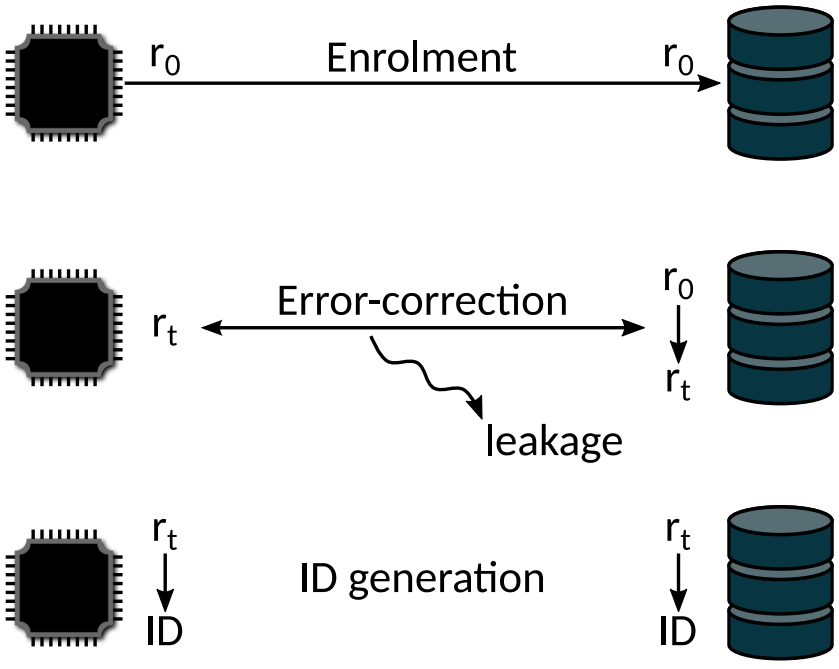
Similarities between two scenarios

Quantum key exchange



$|key| < |m|$

PUF-based ID generation



$|ID| < |r_t|$

One pass

- Perform **parity checks** on **blocks** of the PUF response,
- **Isolate** the errors using **binary search** and correct them,
- Check **current** parity of blocks and **backtrack**,
- **Increase** the block size and **shuffle** the response **randomly**.

One pass

- Perform **parity checks** on **blocks** of the PUF response,
- **Isolate** the errors using **binary search** and correct them,
- Check **current** parity of blocks and **backtrack**,
- **Increase** the block size and **shuffle** the response **randomly**.

Parameters

- Initial block size,
- Number of passes,
- Block size multiplier.

One pass

- Perform **parity checks** on **blocks** of the PUF response,
- **Isolate** the errors using **binary search** and correct them,
- Check **current** parity of blocks and **backtrack**,
- **Increase** the block size and **shuffle** the response **randomly**.

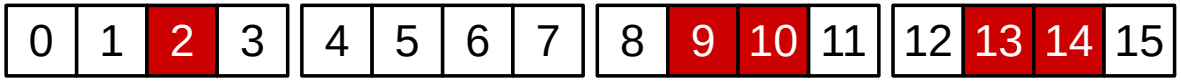
Parameters

- Initial block size,
- Number of passes,
- Block size multiplier.

Information leakage associated with the public discussion

For an n-bit response split into k-bit blocks:

- Parity checks: n/k-bit leakage.
- Binary search: $\log_2(k)$-bit leakage.



Blocks of even relative parity:

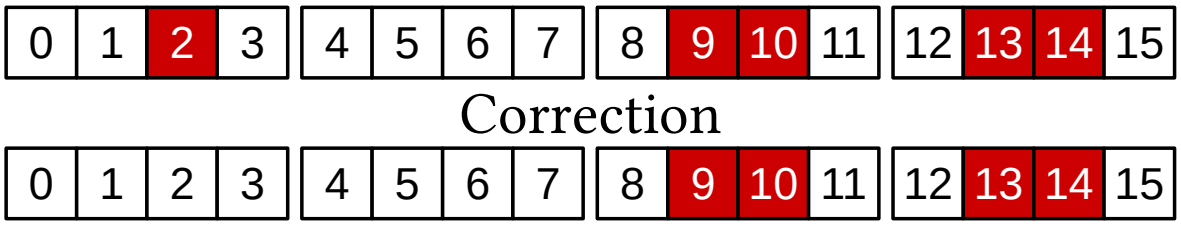
$\varnothing$

Blocks of odd relative parity:

$\varnothing$

$$\text{Relative parity: } P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$$

CASCADE execution

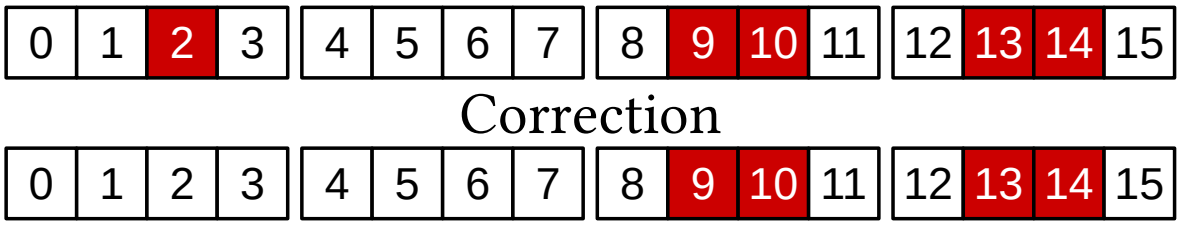


Blocks of even relative parity: $\varnothing$

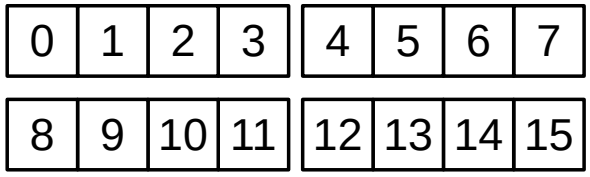
Blocks of odd relative parity: $\varnothing$

$$\text{Relative parity: } P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$$

CASCADE execution



Blocks of even relative parity:

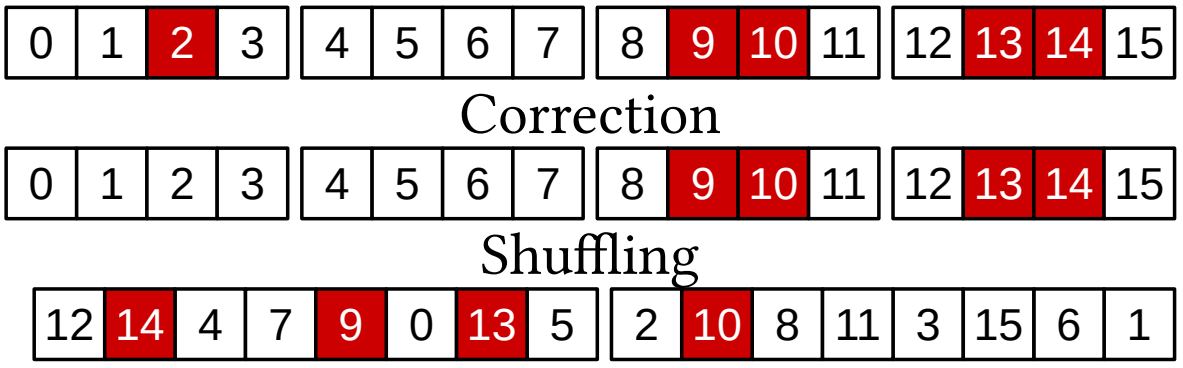


Blocks of odd relative parity:

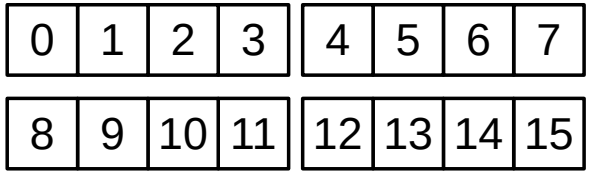
$\varnothing$

$$\text{Relative parity: } P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]] \right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]] \right)}_{\text{Parity of } B_t}$$

CASCADE execution



Blocks of even relative parity:

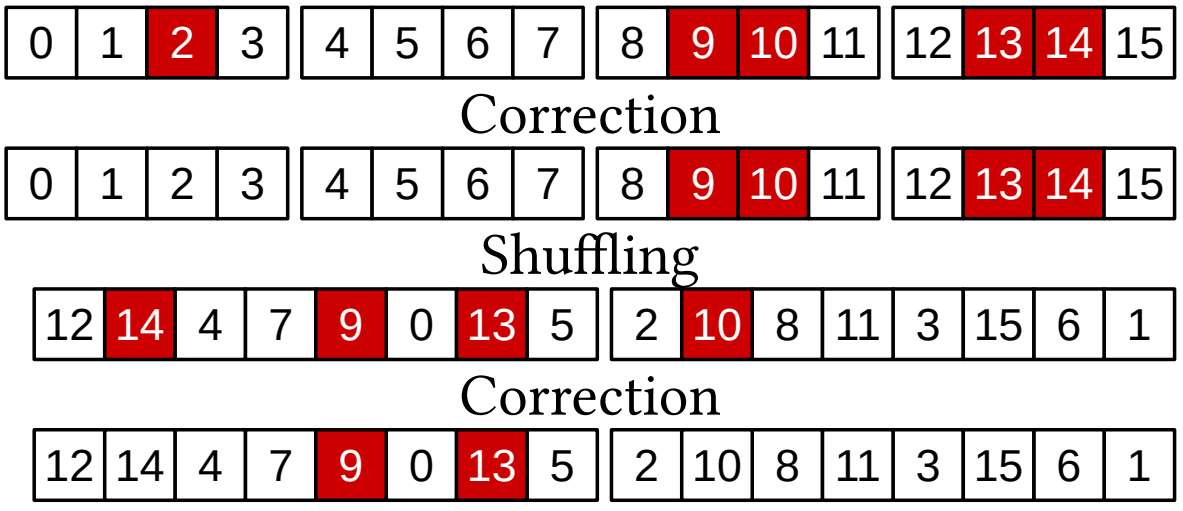


Blocks of odd relative parity:

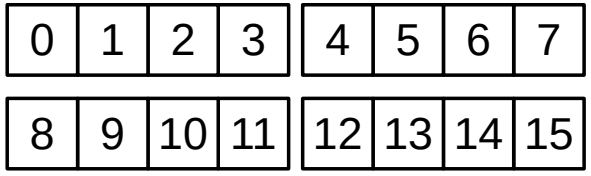
$\varnothing$

$$\text{Relative parity: } P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$$

CASCADE execution



Blocks of even relative parity:

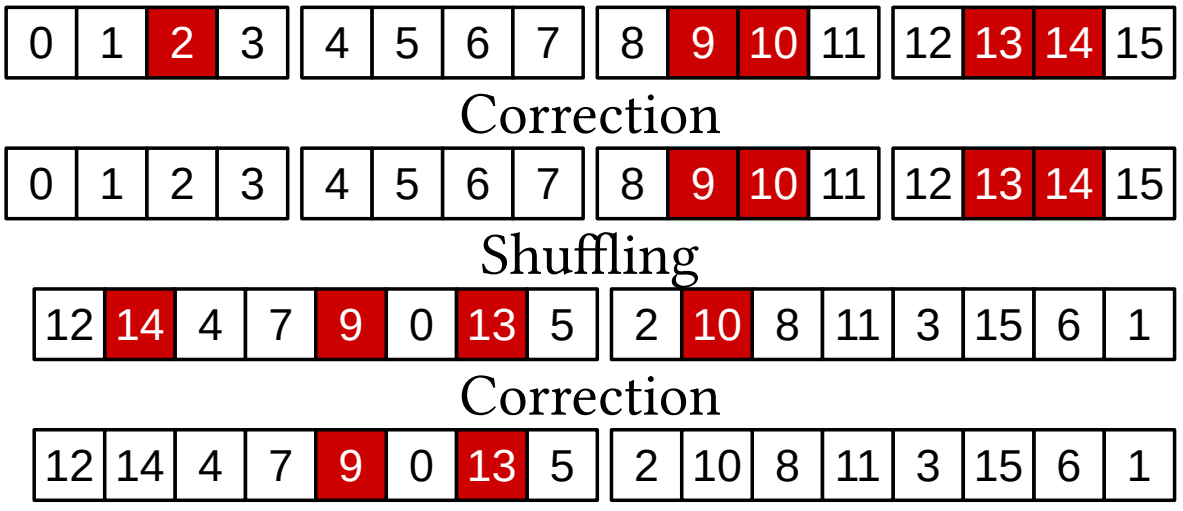


Blocks of odd relative parity:

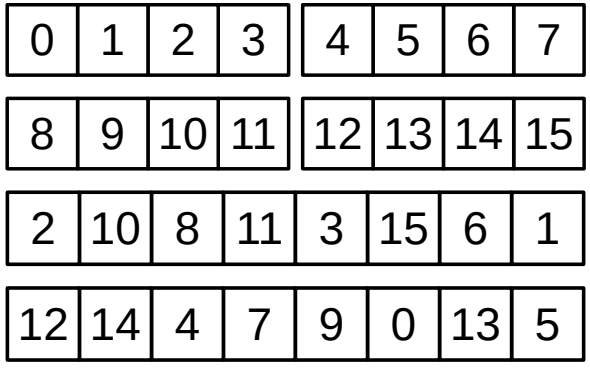
$\varnothing$

Relative parity: $P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$

CASCADE execution



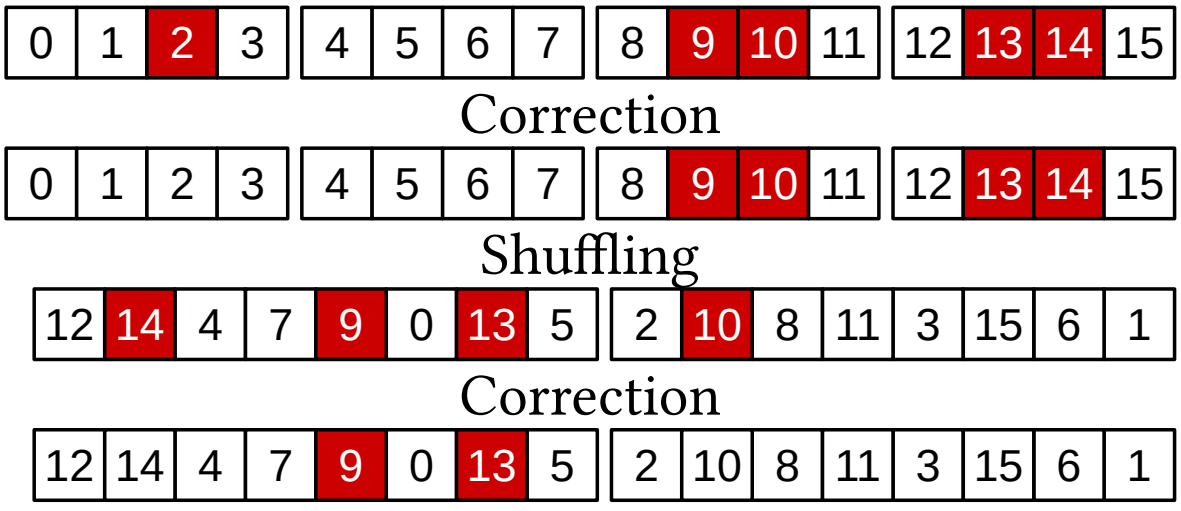
Blocks of even relative parity:



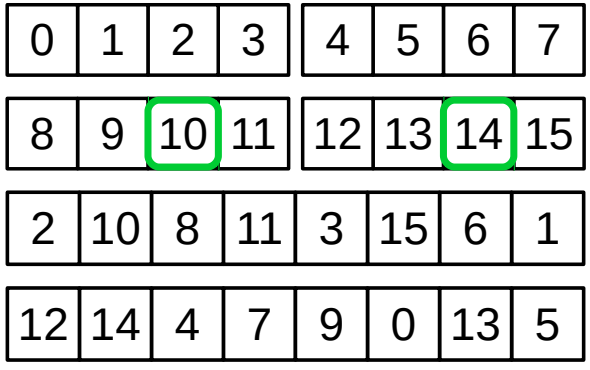
Blocks of odd relative parity:

$\varnothing$

Relative parity: $P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$



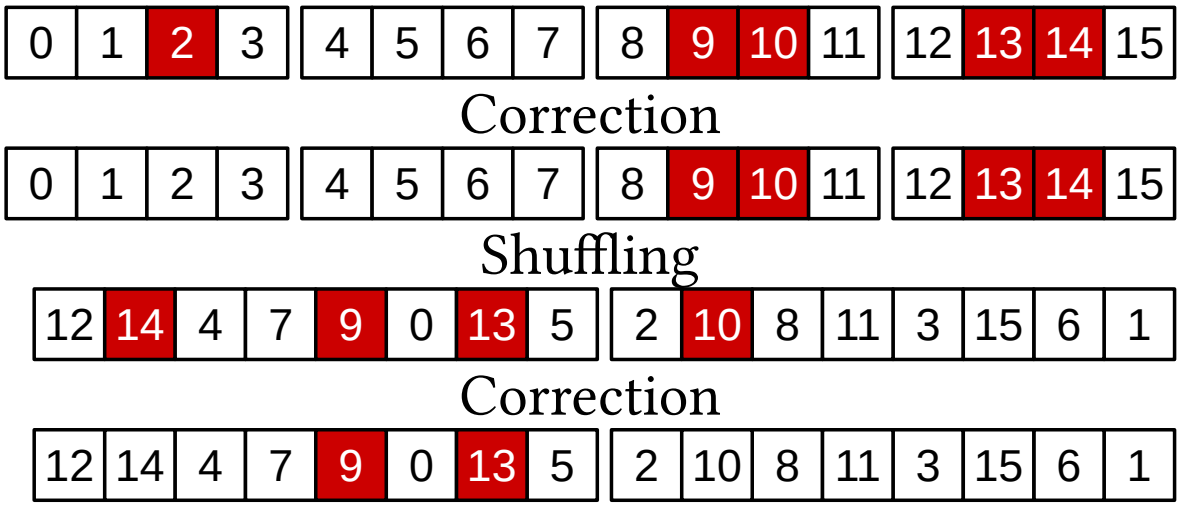
Blocks of even relative parity:



Blocks of odd relative parity:

$\varnothing$

$$\text{Relative parity: } P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$$



Blocks of even relative parity:

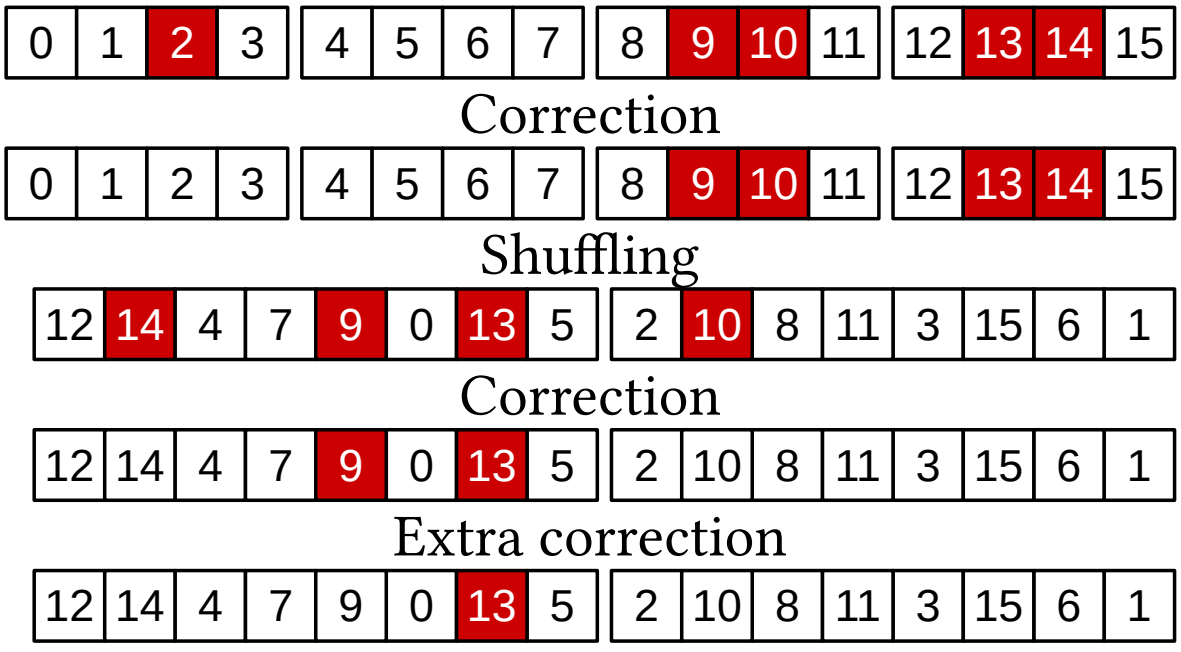
0	1	2	3	4	5	6	7
2	10	8	11	3	15	6	1
12	14	4	7	9	0	13	5

Blocks of odd relative parity:

8	9	10	11	12	13	14	15

Relative parity: $P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$

CASCADE execution



Blocks of even relative parity:

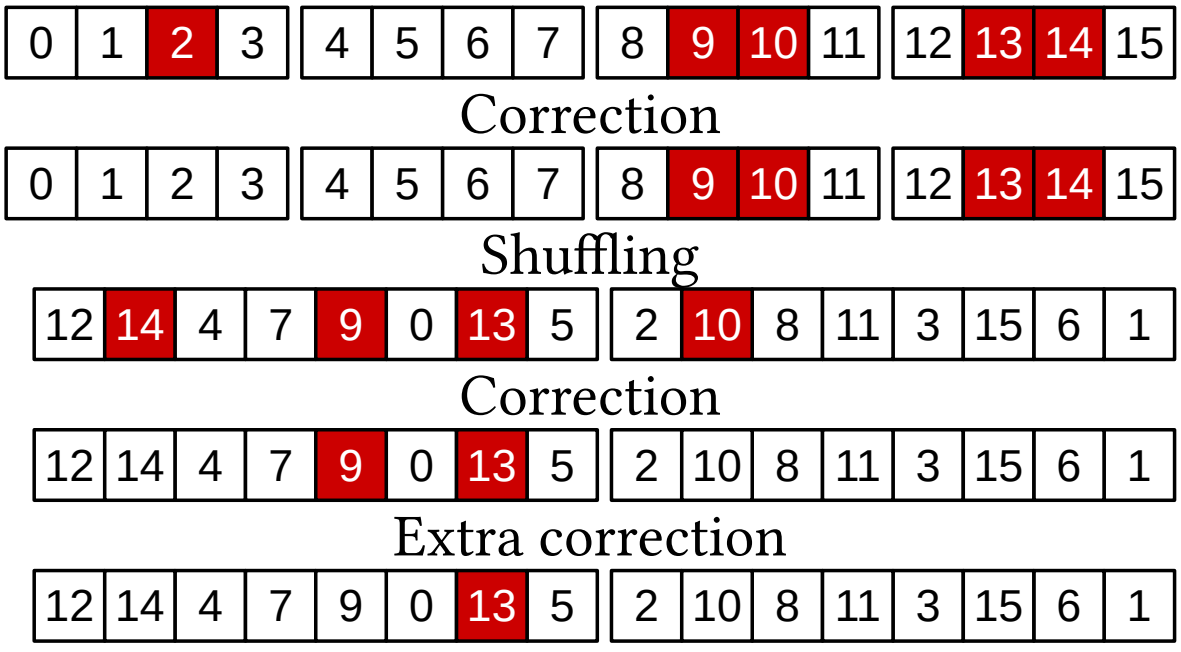
0	1	2	3	4	5	6	7
2	10	8	11	3	15	6	1
12	14	4	7	9	0	13	5

Blocks of odd relative parity:

8	9	10	11	12	13	14	15

Relative parity: $P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$

CASCADE execution



Blocks of even relative parity:

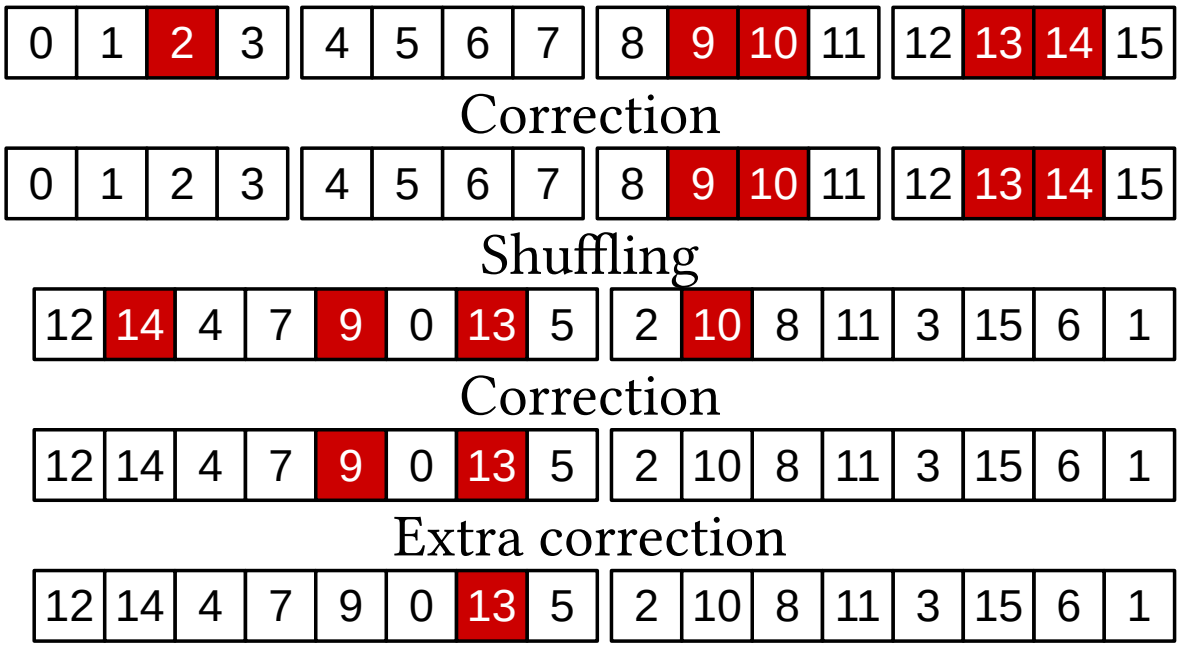
0	1	2	3	4	5	6	7
2	10	8	11	3	15	6	1
12	14	4	7	9	0	13	5

Blocks of odd relative parity:

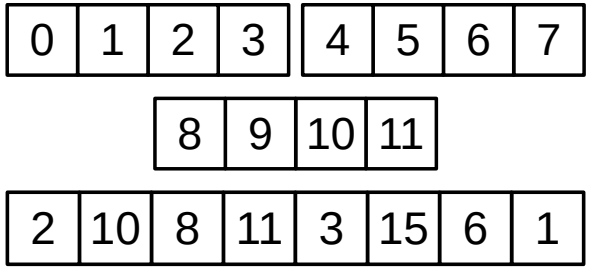
8	9	10	11	12	13	14	15

Relative parity: $P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$

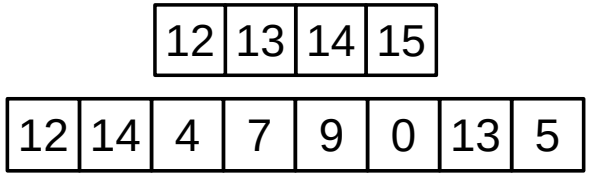
CASCADE execution



Blocks of even relative parity:

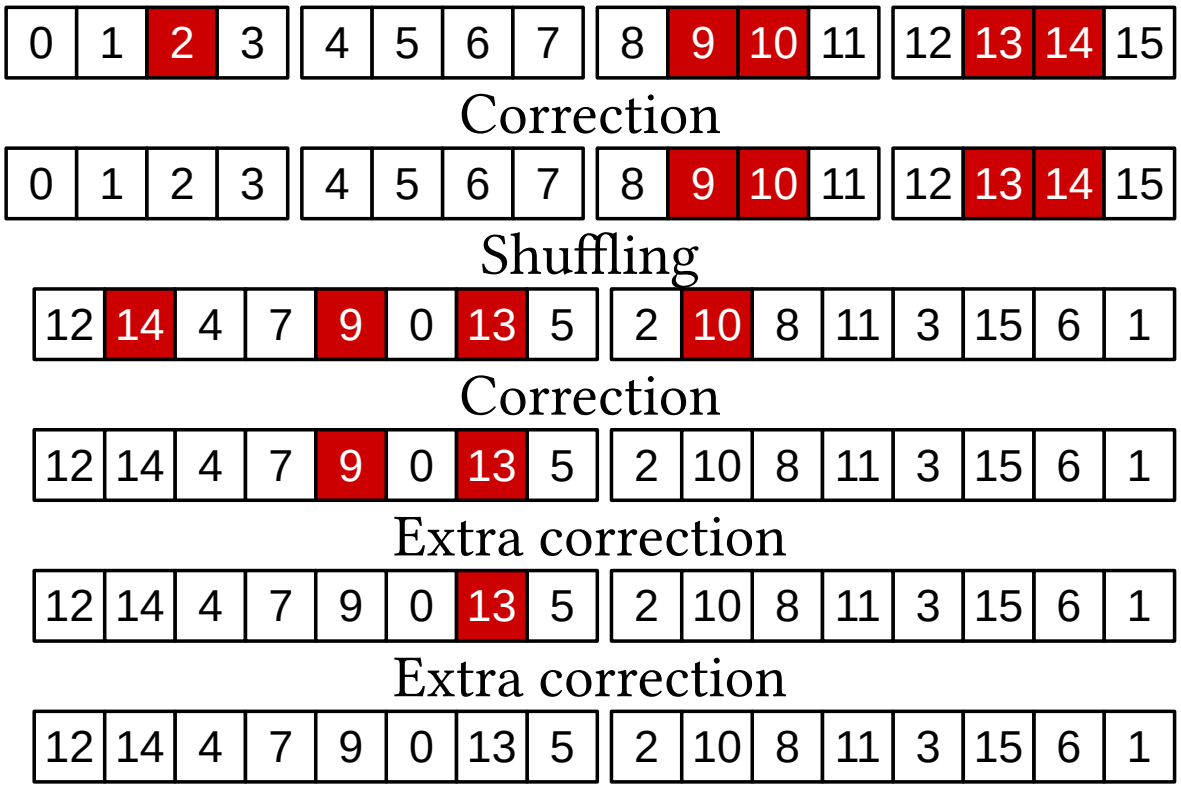


Blocks of odd relative parity:

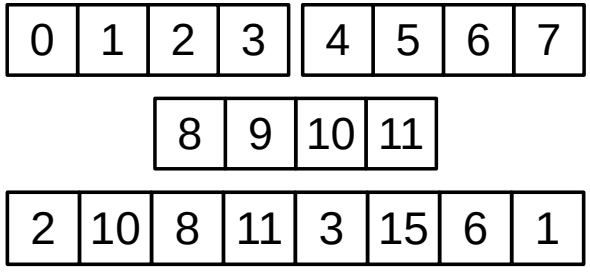


$$\text{Relative parity: } P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$$

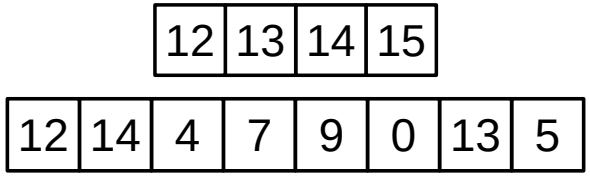
CASCADE execution



Blocks of even relative parity:

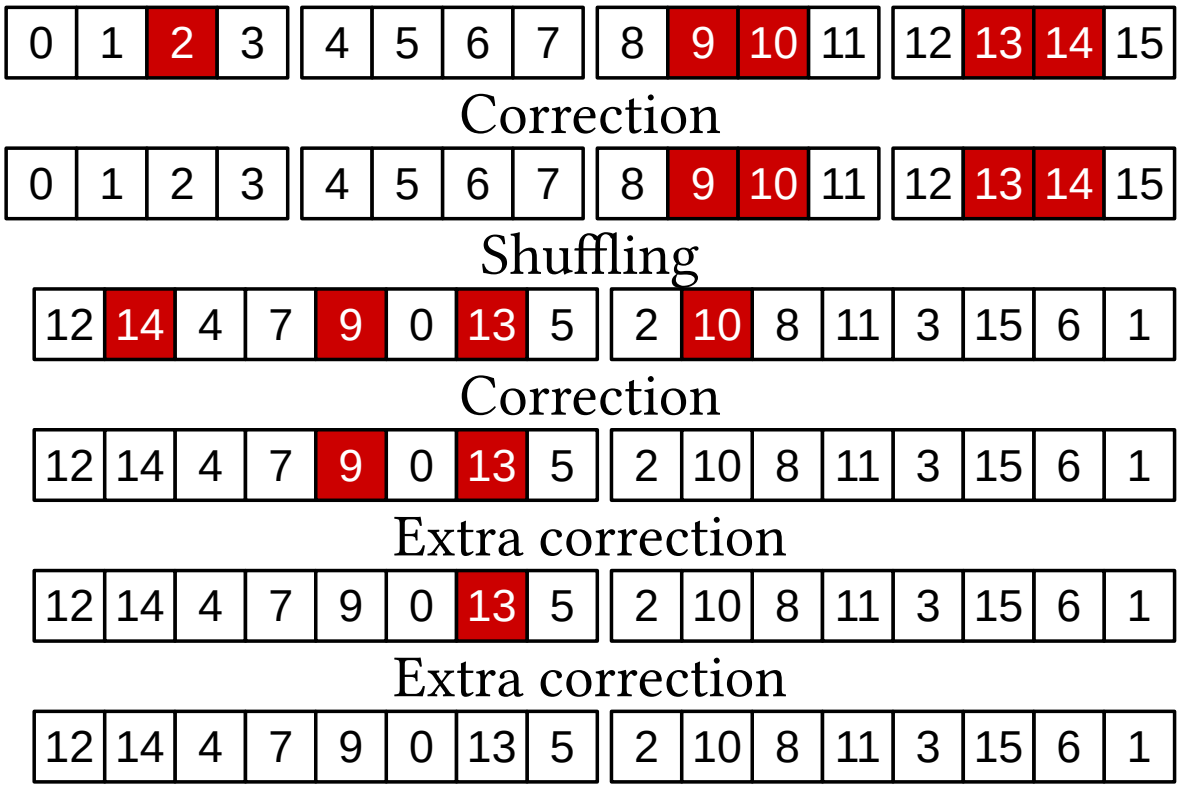


Blocks of odd relative parity:

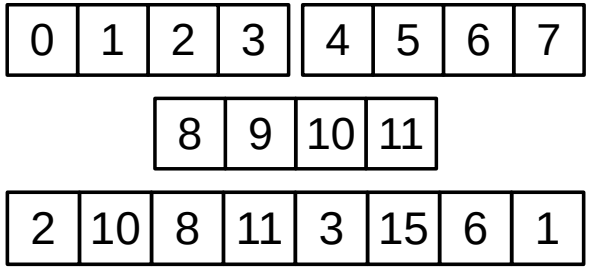


$$\text{Relative parity: } P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$$

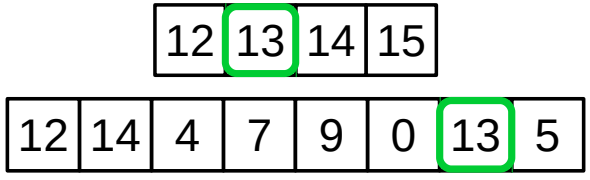
CASCADE execution



Blocks of even relative parity:

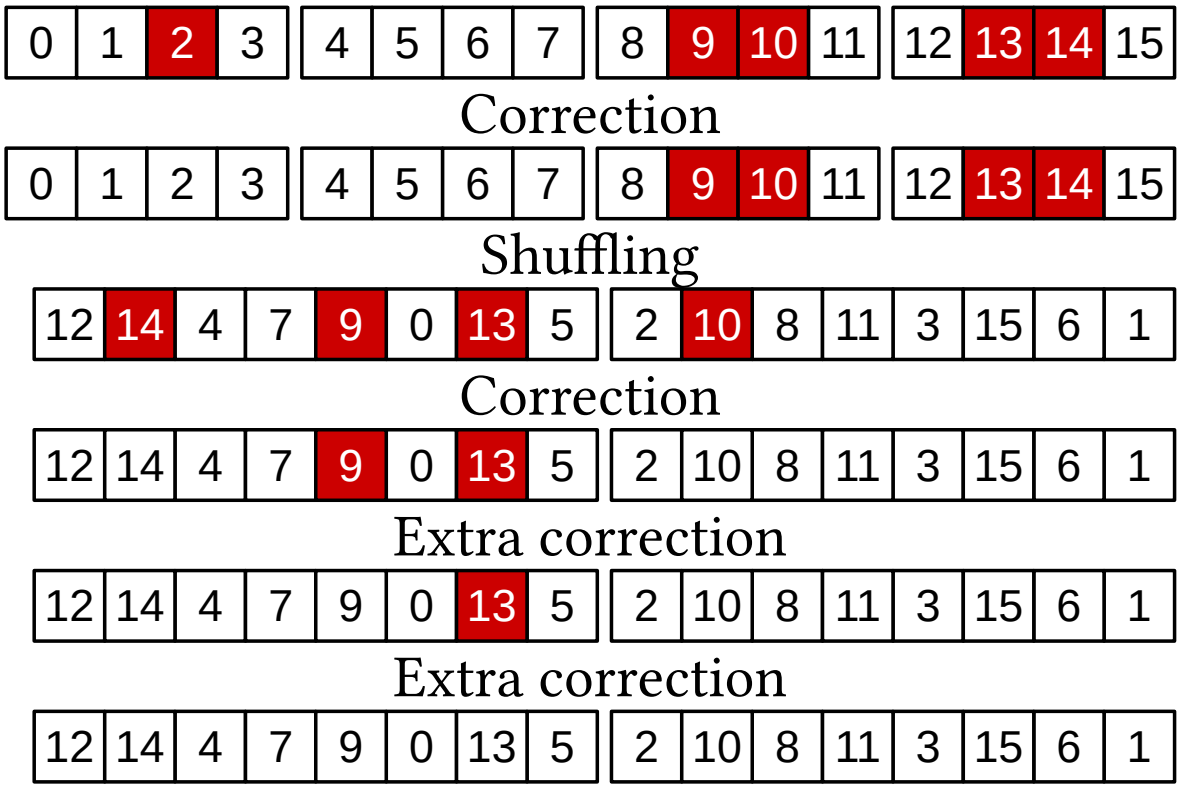


Blocks of odd relative parity:

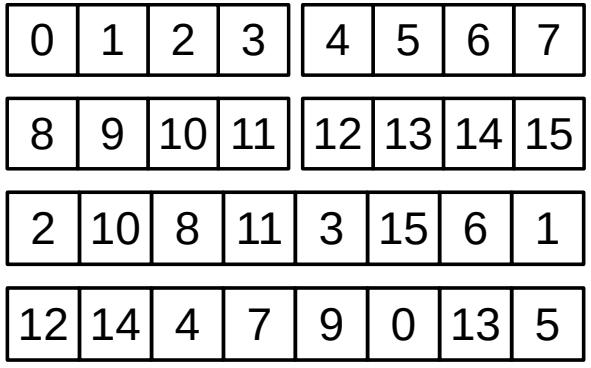


$$\text{Relative parity: } P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$$

CASCADE execution



Blocks of even relative parity:



Blocks of odd relative parity:

$\varnothing$

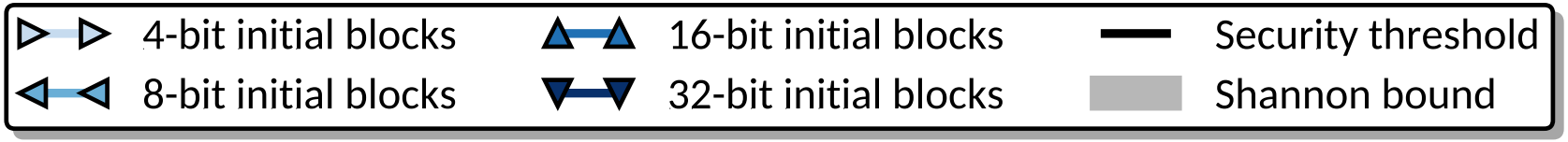
Relative parity: $P_r(B_0, B_t) = \underbrace{\left(\bigoplus_{i=0}^{m-1} r_0[B_0[i]]\right)}_{\text{Parity of } B_0} \oplus \underbrace{\left(\bigoplus_{i=0}^{m-1} r_t[B_t[i]]\right)}_{\text{Parity of } B_t}$

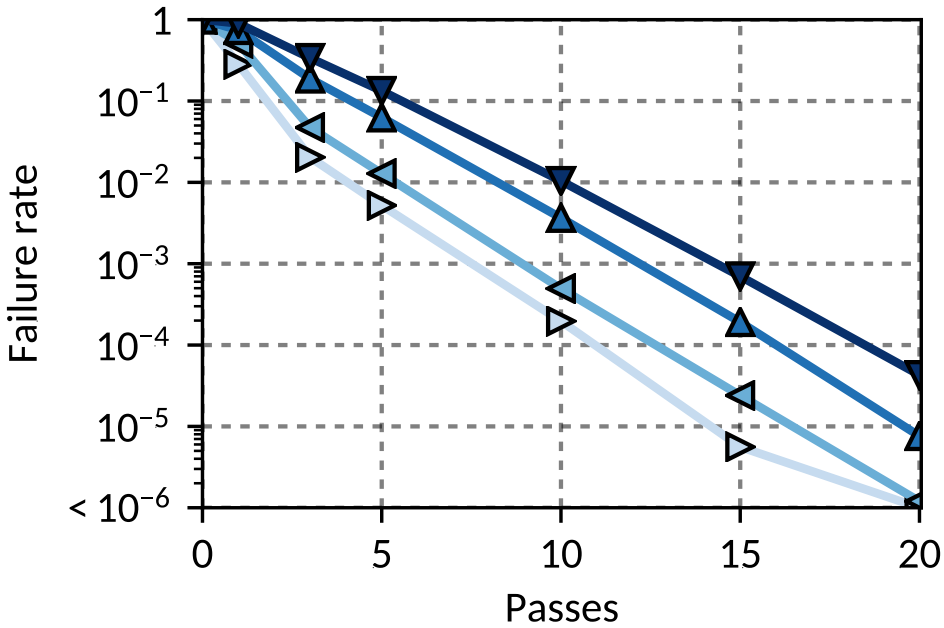
PUFs based on oscillators have a typical error-rate of 2-3% [9] [10].
Keep **128 bits secret** from a 256-bit response with **failure rate < 10^{-6}**.

[9] A. Maiti, J. Casarona, L. McHale, and P. Schaumont. "A large scale characterization of RO-PUF". . *HOST*. 2010, pp. 94–99

[10] A. Cherkaoui, L. Bossuet, and C. Marchand. "Design, Evaluation and Optimization of Physical Unclonable Functions based on Transient Effect Ring Oscillators". *IEEE Transactions on Information Forensics and Security* 11.6 (2016), pp. 1291–1305

PUFs based on oscillators have a typical error-rate of 2-3% [9] [10].
Keep **128 bits secret** from a 256-bit response with **failure rate < 10^{-6}**.

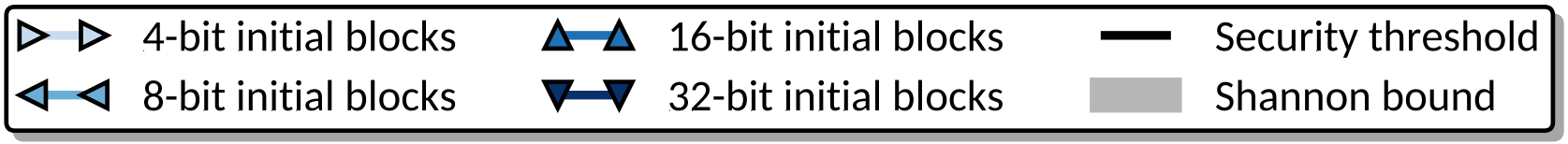


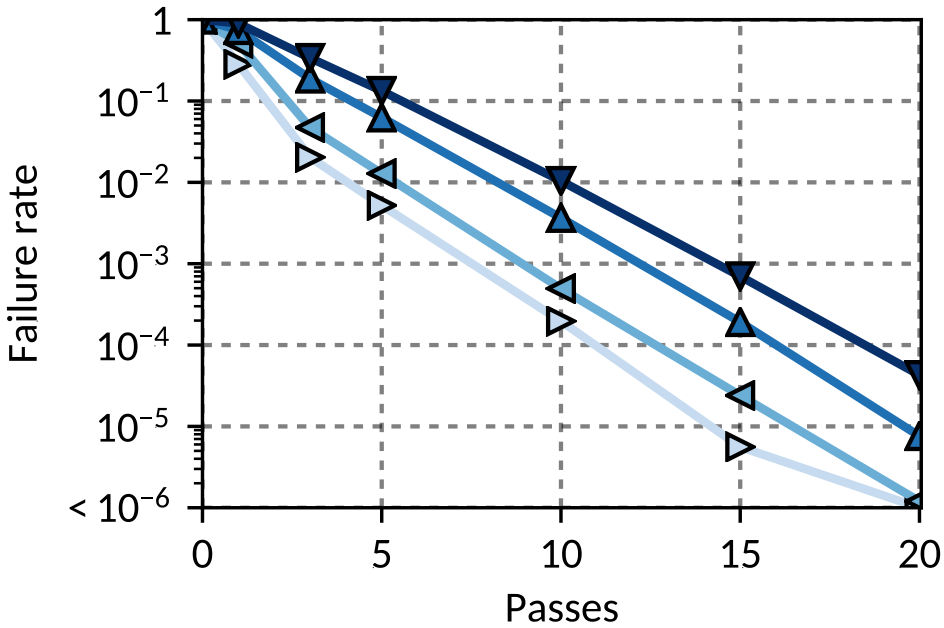


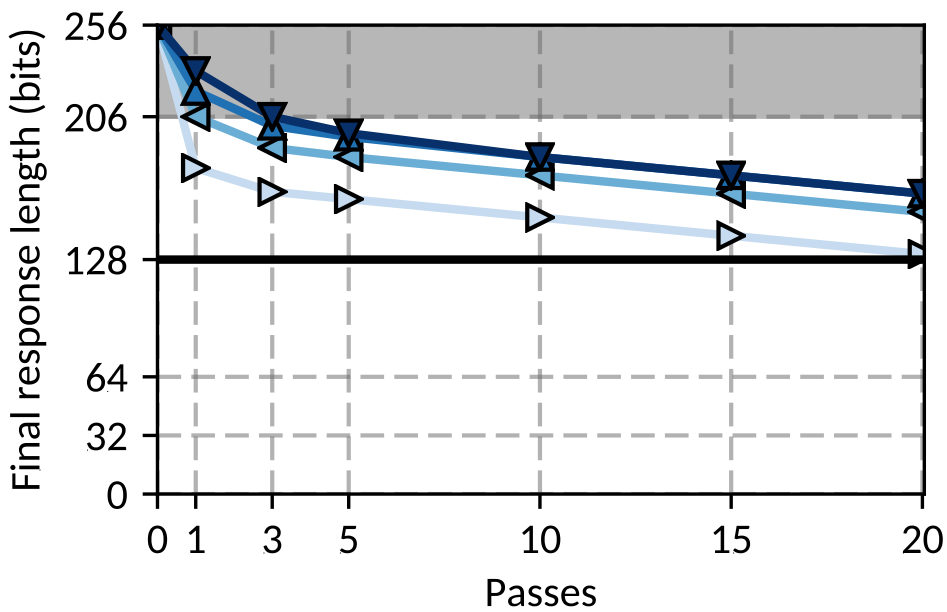
[9] A. Maiti, J. Casarona, L. McHale, and P. Schaumont. "A large scale characterization of RO-PUF". . *HOST*. 2010, pp. 94–99

[10] A. Cherkaoui, L. Bossuet, and C. Marchand. "Design, Evaluation and Optimization of Physical Unclonable Functions based on Transient Effect Ring Oscillators". *IEEE Transactions on Information Forensics and Security* 11.6 (2016), pp. 1291–1305

PUFs based on oscillators have a typical error-rate of 2-3% [9] [10].
Keep **128 bits secret** from a 256-bit response with **failure rate < 10^{-6}**.



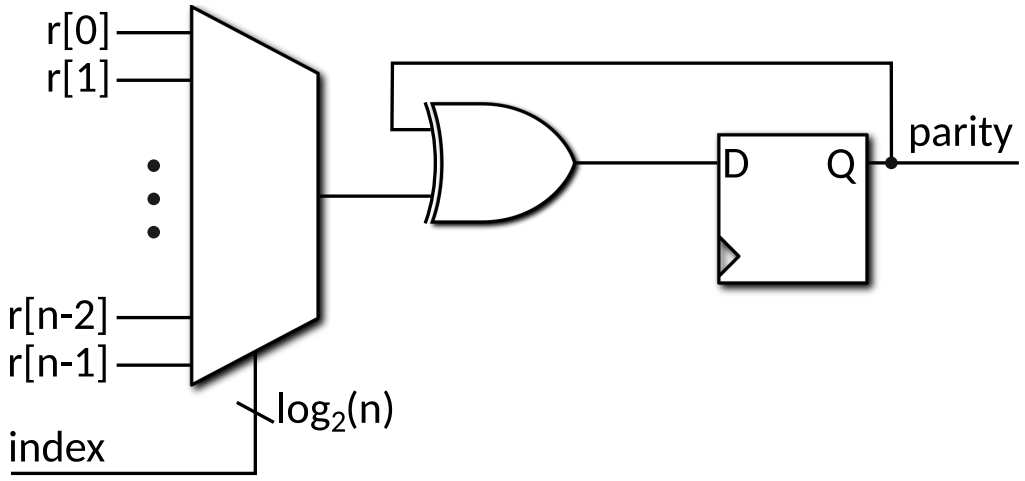




[9] A. Maiti, J. Casarona, L. McHale, and P. Schaumont. "A large scale characterization of RO-PUF". . *HOST*. 2010, pp. 94–99

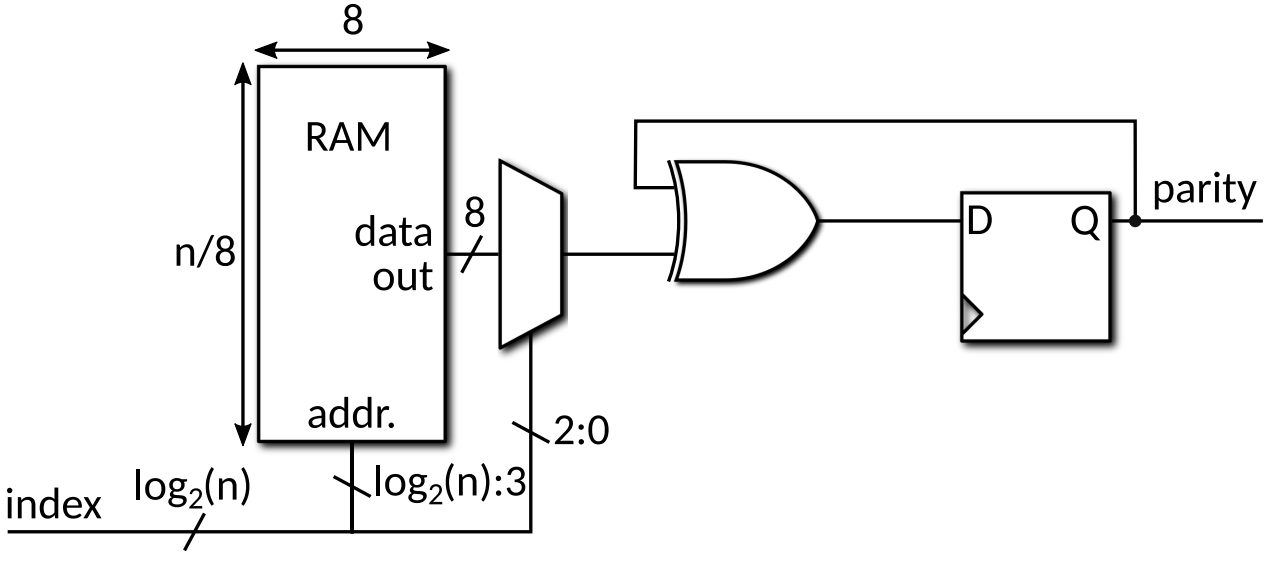
[10] A. Cherkaoui, L. Bossuet, and C. Marchand. "Design, Evaluation and Optimization of Physical Unclonable Functions based on Transient Effect Ring Oscillators". *IEEE Transactions on Information Forensics and Security* 11.6 (2016), pp. 1291–1305

Hardware architectures for the parity computation module



Logic resources:

- Spartan 3: 67 Slices
- Spartan 6: 19 Slices
- 0 RAM bits



Logic resources:

- Spartan 3: 3 Slices
- Spartan 6: 1 Slice
- 256 RAM bits

Classical error-correcting codes for PUFs

Article	Construction and code(s)		Logic resources (Slices) Spartan 3	Logic resources (Slices) Spartan 6	Block RAM bits
[11]	Reed-Muller (4, 7)			179	0
[12]	Reed-Muller (2, 6)		164		192
[13]	Concatenated: Repetition and Reed Muller		168		0
[14]	Differential Sequence Coding and Viterbi		75	27	10752
This work: CASCADE protocol		logic only	67	19	0
		with RAM	3	1	256

[11] M. Hiller et al. "Low-Area Reed Decoding in a Generalized Concatenated Code Construction for PUFs". *ISVLSI*. 2015, pp. 143–148

[12] R. Maes, P. Tuyls, and I. Verbauwhede. "Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs". *CHES*. 2009, pp. 332–347

[13] C. Bösch et al. "Efficient Helper Data Key Extractor on FPGAs". *CHES*. 2008, pp. 181–197

[14] M. Hiller, M. Yu, and G. Sigl. "Cherry-Picking Reliable PUF Bits With Differential Sequence Coding". *IEEE Trans. Information Forensics and Security* 11.9 (2016), pp. 2065–2076

Key reconciliation protocol for PUF responses: conclusion

First usage of a **key reconciliation protocol** for PUF responses:

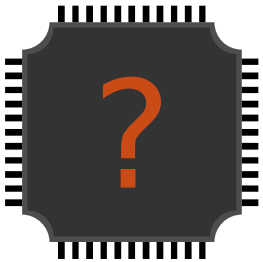
- ✔ **most lightweight** error-correction solution of state-of-the-art,
- ✔ can reach **very low** failure rates (down to 10^{-8}),
- ✔ leakage is **limited** and **easy** to estimate,
- ✔ **parameterizable** and can be changed **on the fly**.

Associated publication:

B. Colombier, L. Bossuet, D. Hély, and V. Fischer. "Key Reconciliation Protocols for Error Correction of Silicon PUF Responses". *IEEE Transactions on Information Forensics and Security* 12.8 (2017), pp. 1988–2002

Integration and demonstrator

A **secure remote activation scheme** that is:

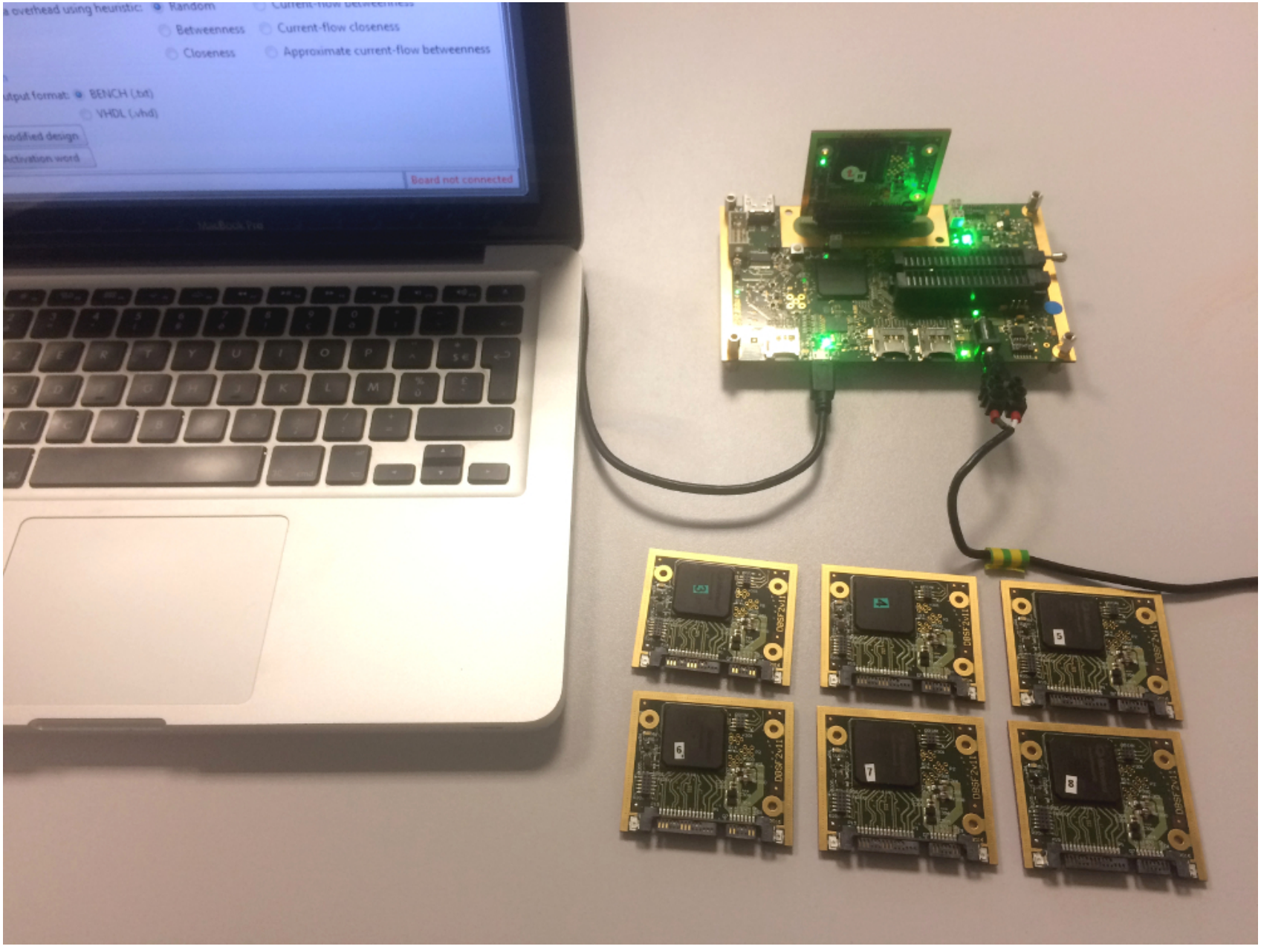


- **Easy to use** by legitimate parties,
  - ✔ seamless **integration** into the standard design flow,
  - ✔ low impact on the **performances** of the IP core,
  - ✔ usable for any IP core (**universal**),
- **Hard to circumvent** for an adversary,
  - ✔ **impossibility** to use illegal copies,
  - ✔ **instance**-specific identifier,
  - ✔ **security** guaranteed by a cipher [15].

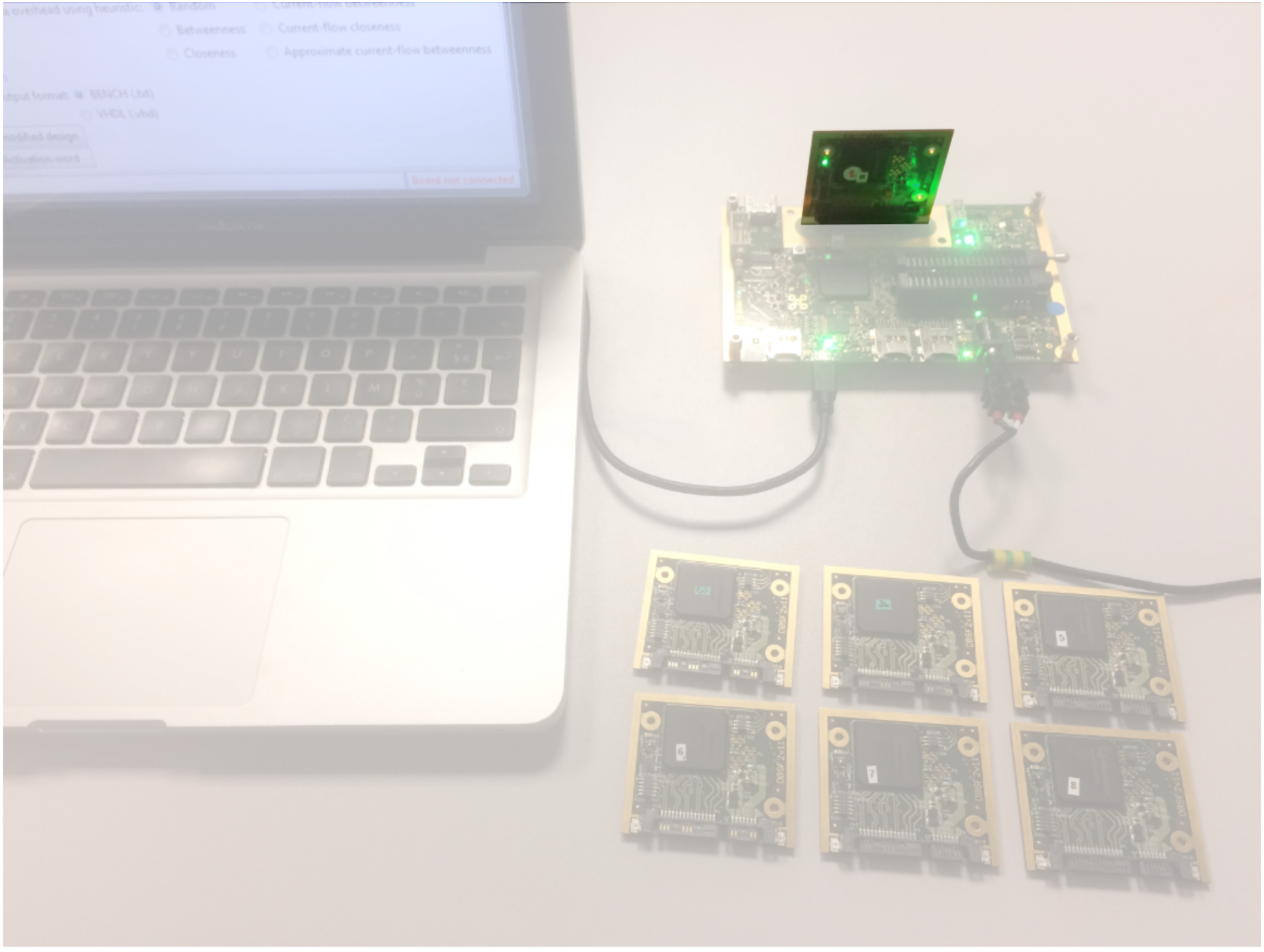
---

[15] C. Marchand, L. Bossuet, and K. Gaj. "Area-oriented comparison of lightweight block ciphers implemented in hardware for the activation mechanism in the anti-counterfeiting schemes". *International Journal of Circuit Theory and Applications* 45.2 (2017), pp. 274–291

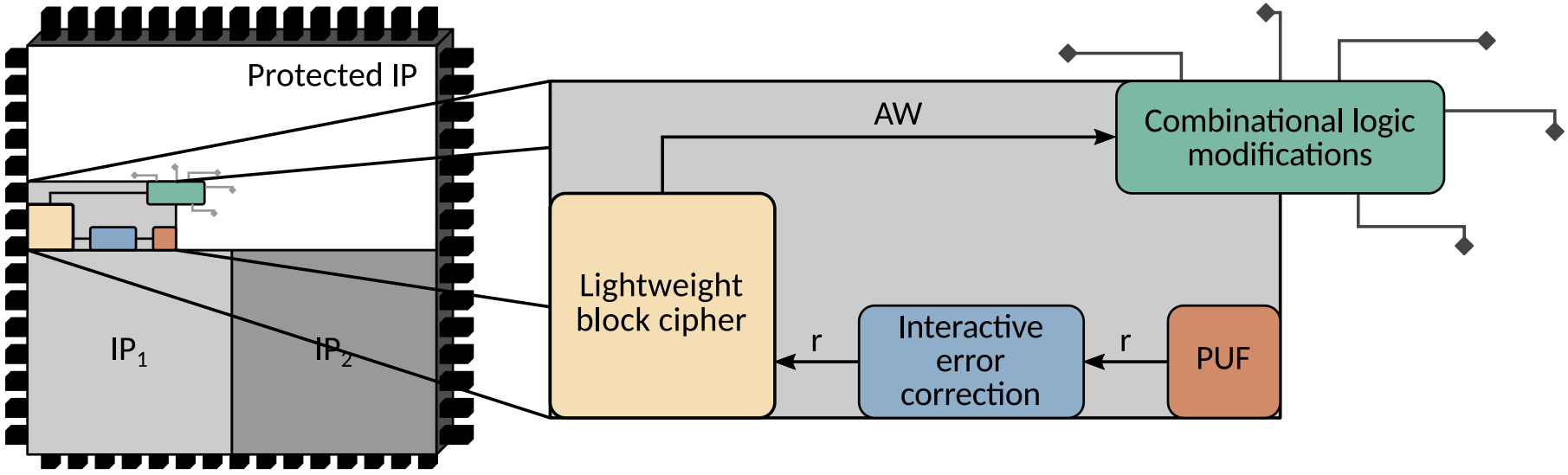
Hardware/software infrastructure



Hardware/software infrastructure

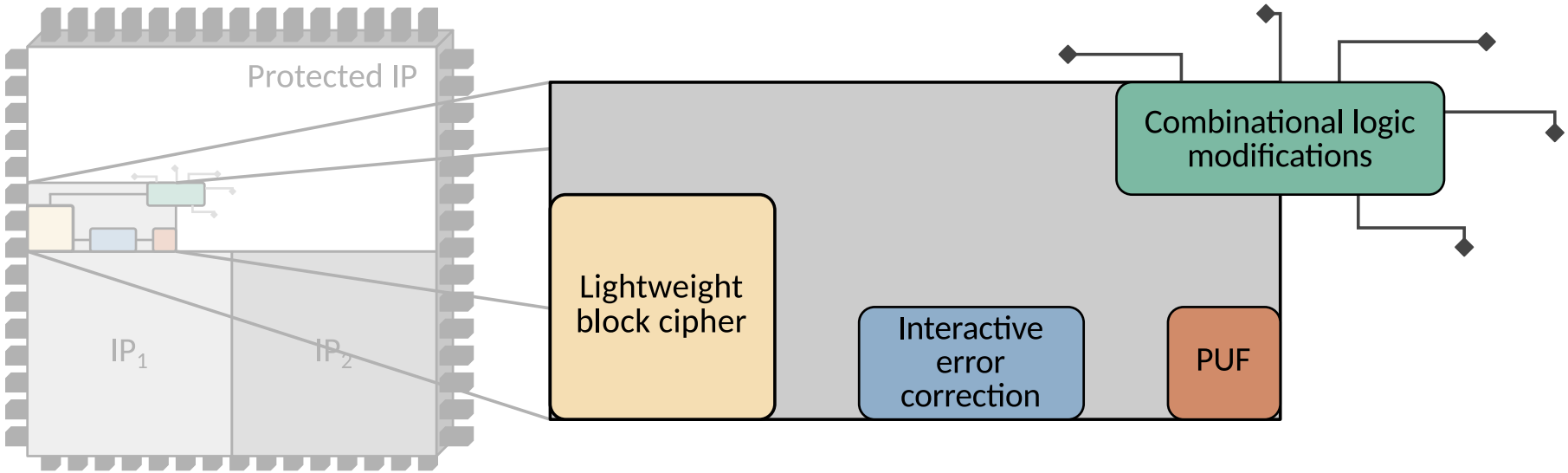


IP protection module overview



	Intel Cyclone V		Microsemi SF2	
	6-LUTs	DFFs	4-LUTs	DFFs
TERO-PUF	4841	160	2258	158
Response shift register	0	128	0	128
Communication	321	2560	2664	2478
MUX indexes 128x7:7	301	0	595	0
IP protection module	∼**500**	∼**350**	∼**700**	∼**350**
MUX response bits 128:1	67	0	85	0
AW storage	0	128	0	128
`CASCADE` module	1	1	1	1
Controller	104	90	101	69
Cipher [15]	∼300	∼200	∼500	∼200

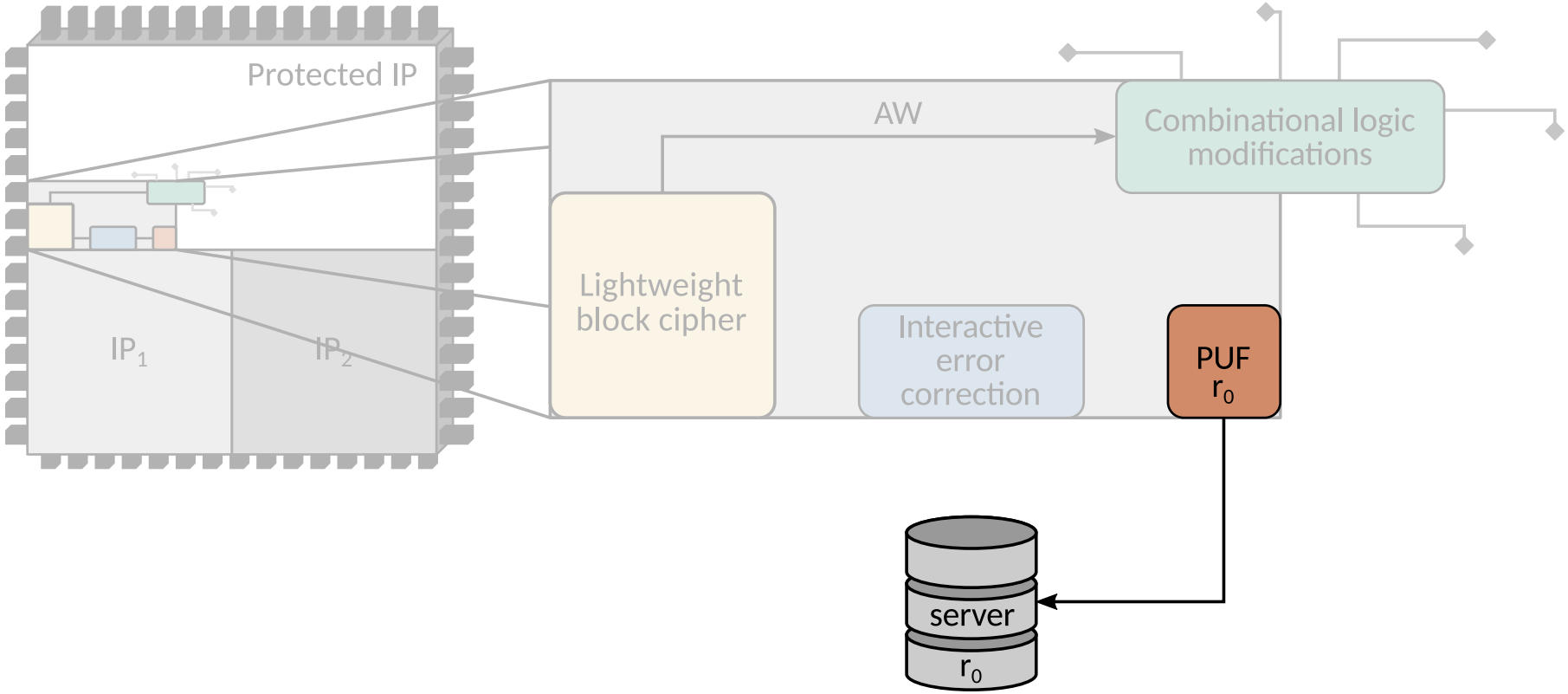
Typical use-case at design time



Typical use-case at design time

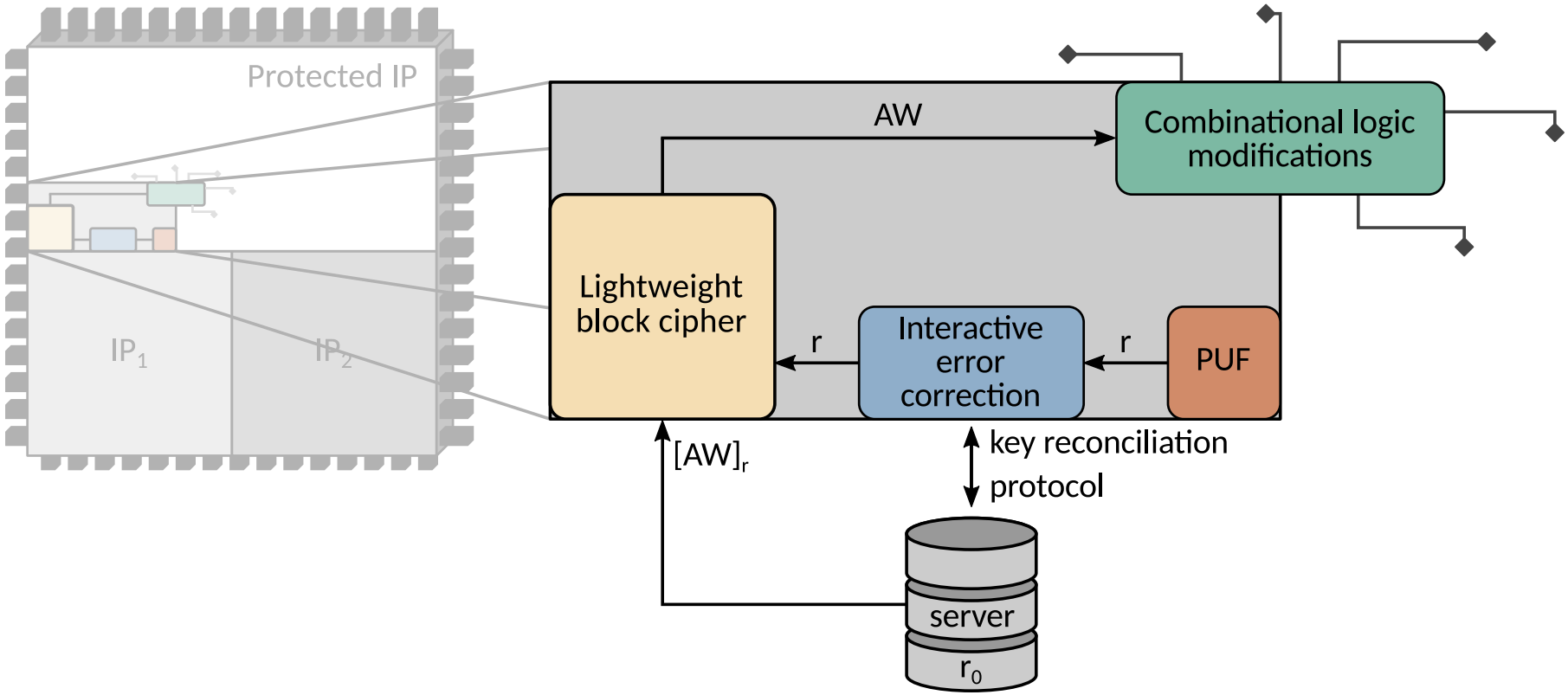
- **Modify** the combinational design to make it activable,
- **Wrap** the design with the additional components: block cipher, PUF, CASCADE module, etc.
- **Store** the **associated** activation word.

Typical use-case at enrollment time



Typical use-case at enrollment time

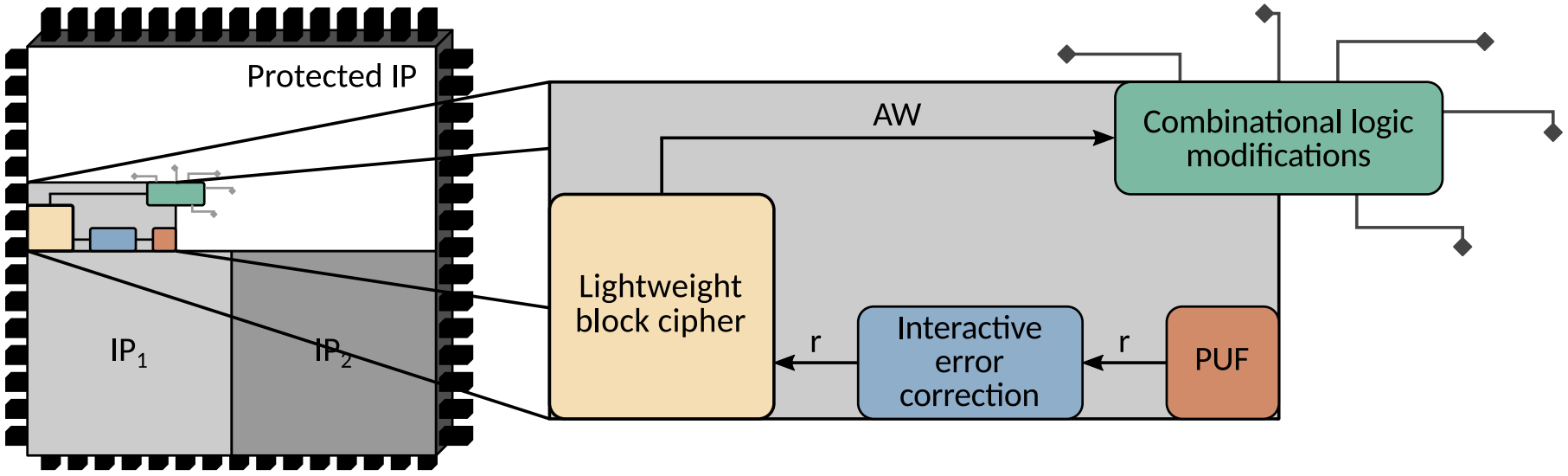
- Get the PUF **reference response**,
- *Optional*: **Characterize** the PUF in operational conditions to **estimate the expected error-rate**,
- **Store** the reference response on the server.



Typical use-case at activation time

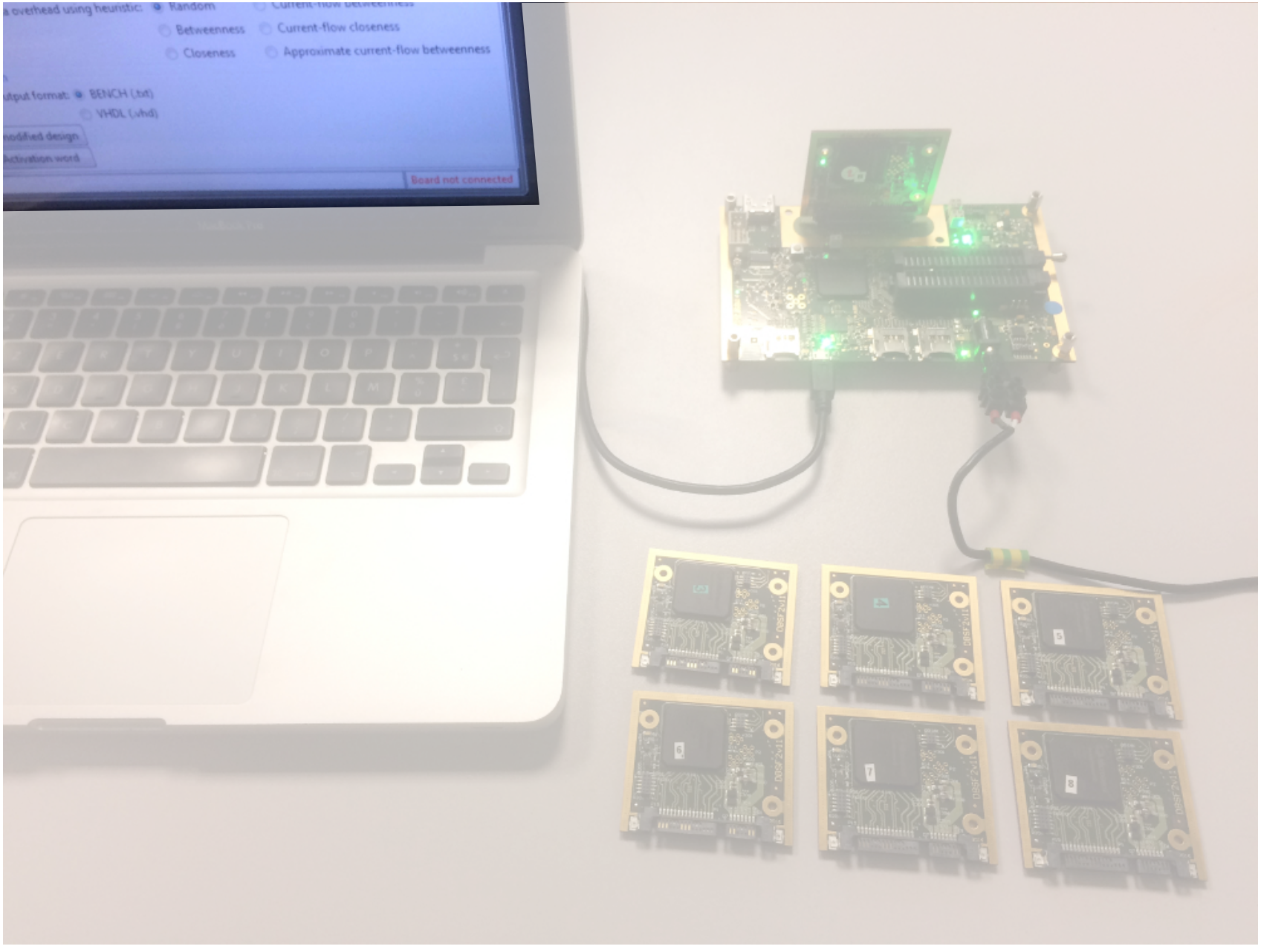
- **Re-generate** a PUF response on the device,
- Perform **reconciliation** to correct the PUF response on the **server side**,
- **Encrypt** the activation word with the response and send it.

IP protection module overview

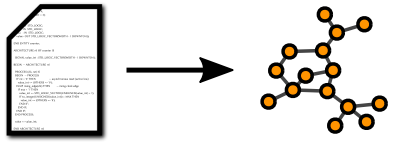


https://gitlab.com/SALWARE/salware_hector_mb
https://gitlab.com/SALWARE/salware_db_sf2

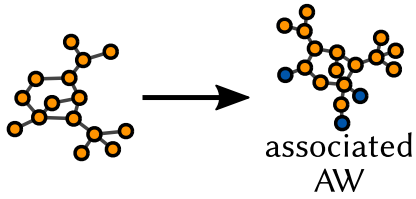
Hardware/software infrastructure



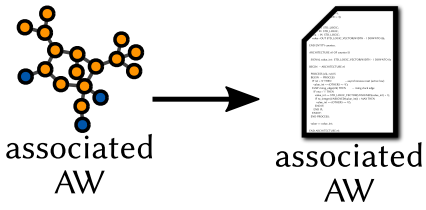
Parse the netlist (multiple formats) into a graph



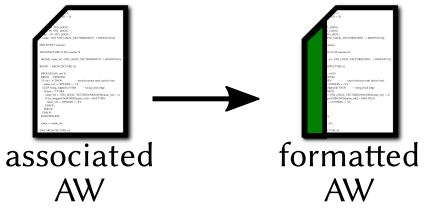
Modify the graph for logic locking/masking



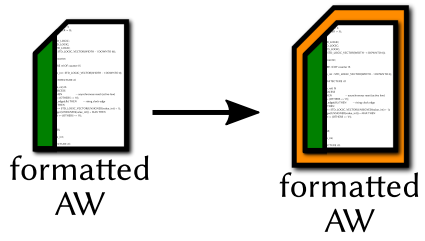
Convert back into a netlist



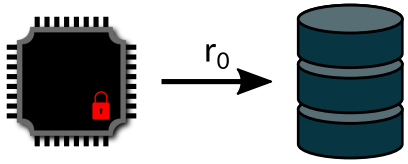
Add the AW decoder



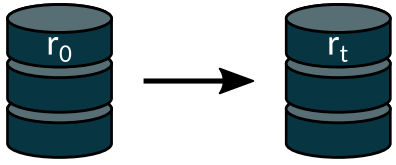
Wrap with the extra components



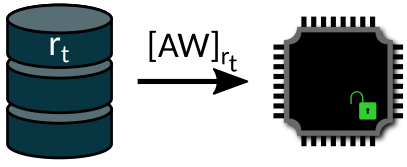
Get the PUF reference response and store it



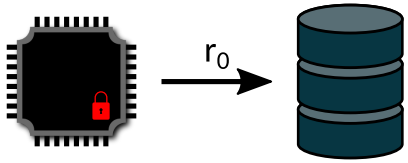
Perform the reconciliation



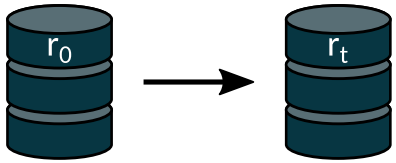
Encrypt the AW and send it



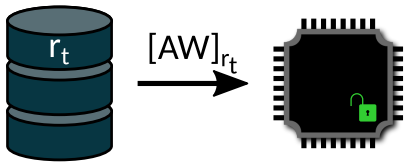
Get the PUF reference response and store it



Perform the reconciliation



Encrypt the AW and send it



`https://gitlab.com/SALWARE/salware_app`

Associated publications:

B. Colombier et al. “Hardware Demo: Complete Activation Scheme for IP Design Protection”. *HOST*. 2017

B. Colombier et al. “Hardware Demo: Complete Activation Scheme for IP Design Protection”. *FPL*. 2017

B. Colombier, L. Bossuet, and D. Hély. “A comprehensive hardware/software infrastructure for IP cores design protection”. *FPT*. 2017

Conclusion and perspectives

Summary of contributions:

- Two methods of logic modifications for IP protection:
  - **Logic locking**: lightweight and scalable to very large netlists,
  - **Centrality-based logic masking**: better trade-off between computational complexity and masking efficiency,
- First use of the **CASCADE** key reconciliation protocol with **PUFs**, **10x more lightweight** than existing ECCs,
- **Full integration** and demonstrator.

Publications:

- 4 journals (2x IEEE TIFS, MICPRO, IET-CDT),
- 4 international conferences (2x ISVLSI, FCCM, FPT).

Possible research perspectives:

- **Fine-grained** licensing approaches (evaluation, premium, etc),
- Security **evaluation** of hardware IP protections,
- **Trusted** computing: (un)trusted mode $\longleftrightarrow$ (un)locked IP core,
- Solutions for **analog** IP cores (25% of counterfeited ICs).

Possible industrial perspectives:

SALWARE
French ANR Project

Business
- Monetization,
- Pay-per-use,
- Royalties.

Software
- Modified design flow,
- Database,
- Key management.

Hardware
- Activation,
- ID,
- Security.

algodone

Possible industrial perspectives:

Business

- Monetization,
- Pay-per-use,
- Royalties.

Software

- Modified design flow,
- Database,
- Key management.

Hardware

- Activation,
- ID,
- Security.

algodone

— Questions ? —