Key reconciliation protocol application to error correction in silicon PUF responses

Brice Colombier*, Lilian Bossuet*, David Hély+

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

+Univ. Grenoble Alpes, LCIS
F-26000, Valence, France

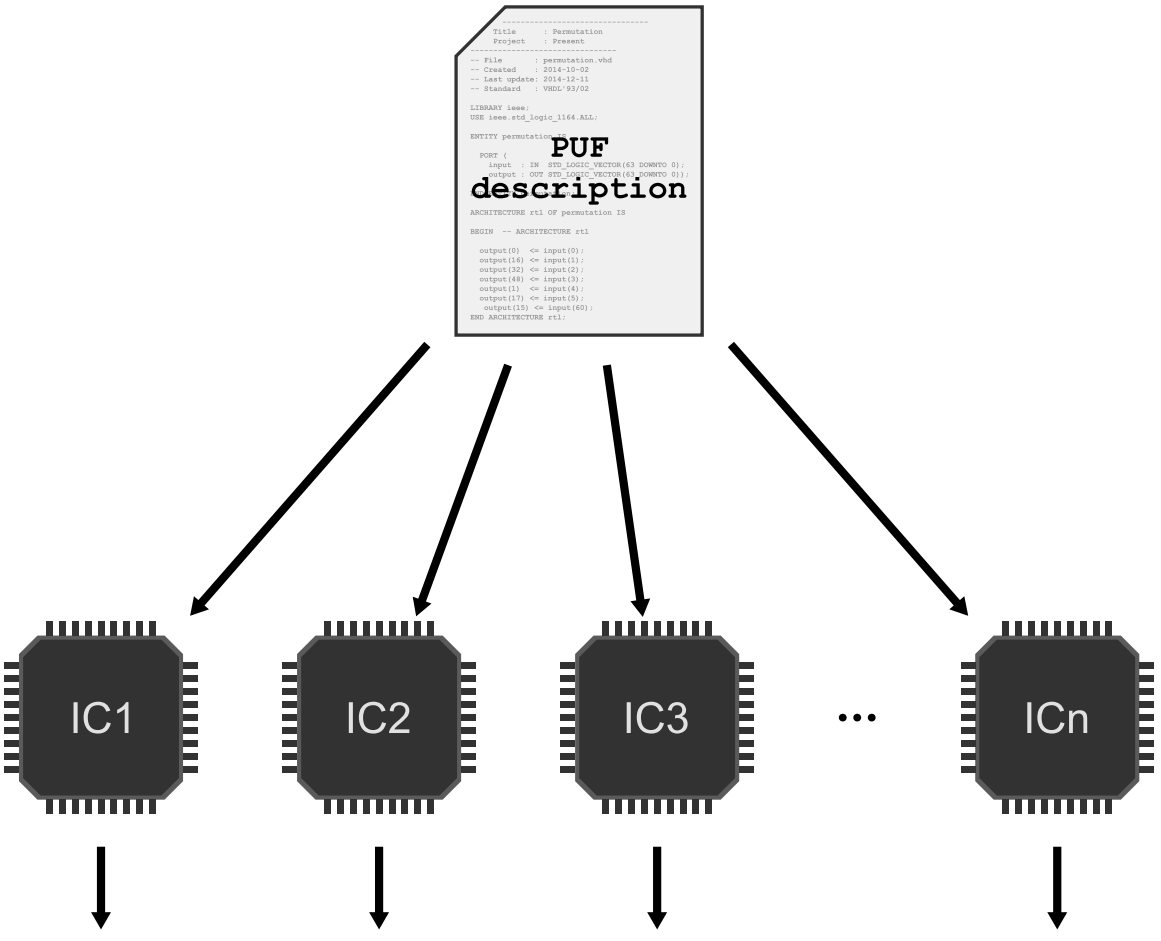
March 18, 2016

LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE

RhôneAlpes Région

SALWARE
French ANR Project

PUFs as identifiers



Principle:

Extract entropy from process variations.

Aim:

Provide a unique, per-device ID, thanks to the **inter-device** uniqueness.

Different responses to the **same** challenge.

The instability problem

Problem:

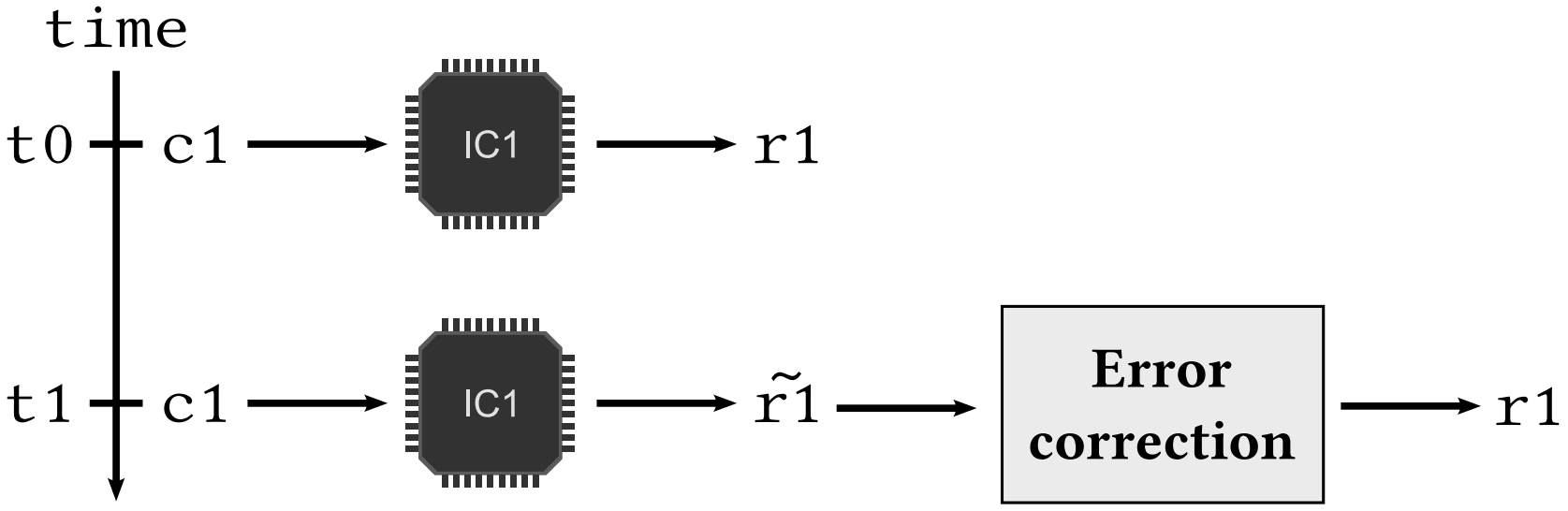
The PUF responses
to the **same** challenge
change over time.

This variation depends on multiple parameters:

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

Assumptions and requirements

Solution: Correct the PUF response.

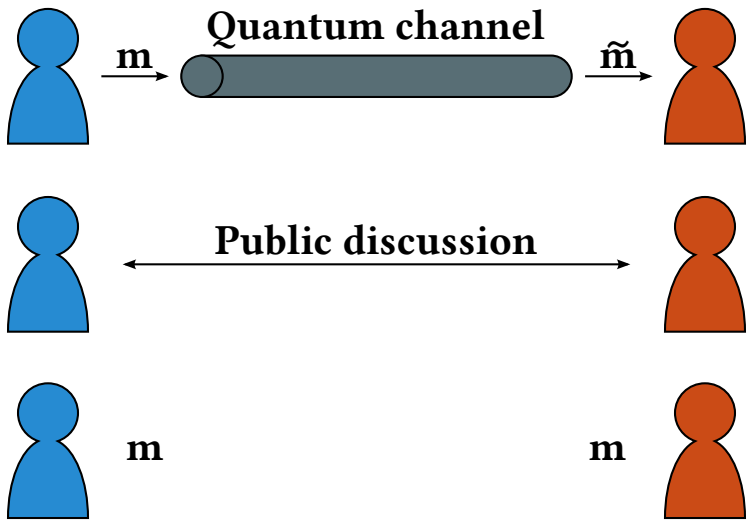


Requirements for the error correction module:

- Low area,
- High correction probability,
- Limited leakage.

Information reconciliation protocols

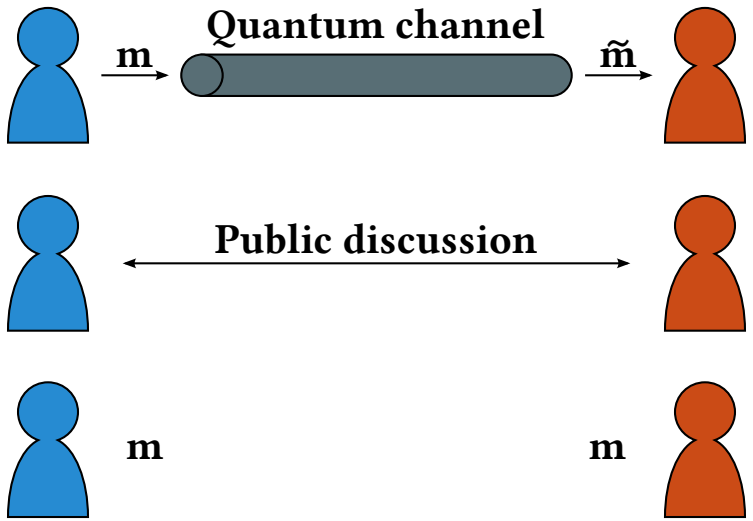
CASCADE introduced in 1993 by Brassard and Salvail [1].



[1] G. Brassard and L. Salvail, *Secret-Key Reconciliation by Public Discussion*, **EUROCRYPT**, 1993.

Information reconciliation protocols

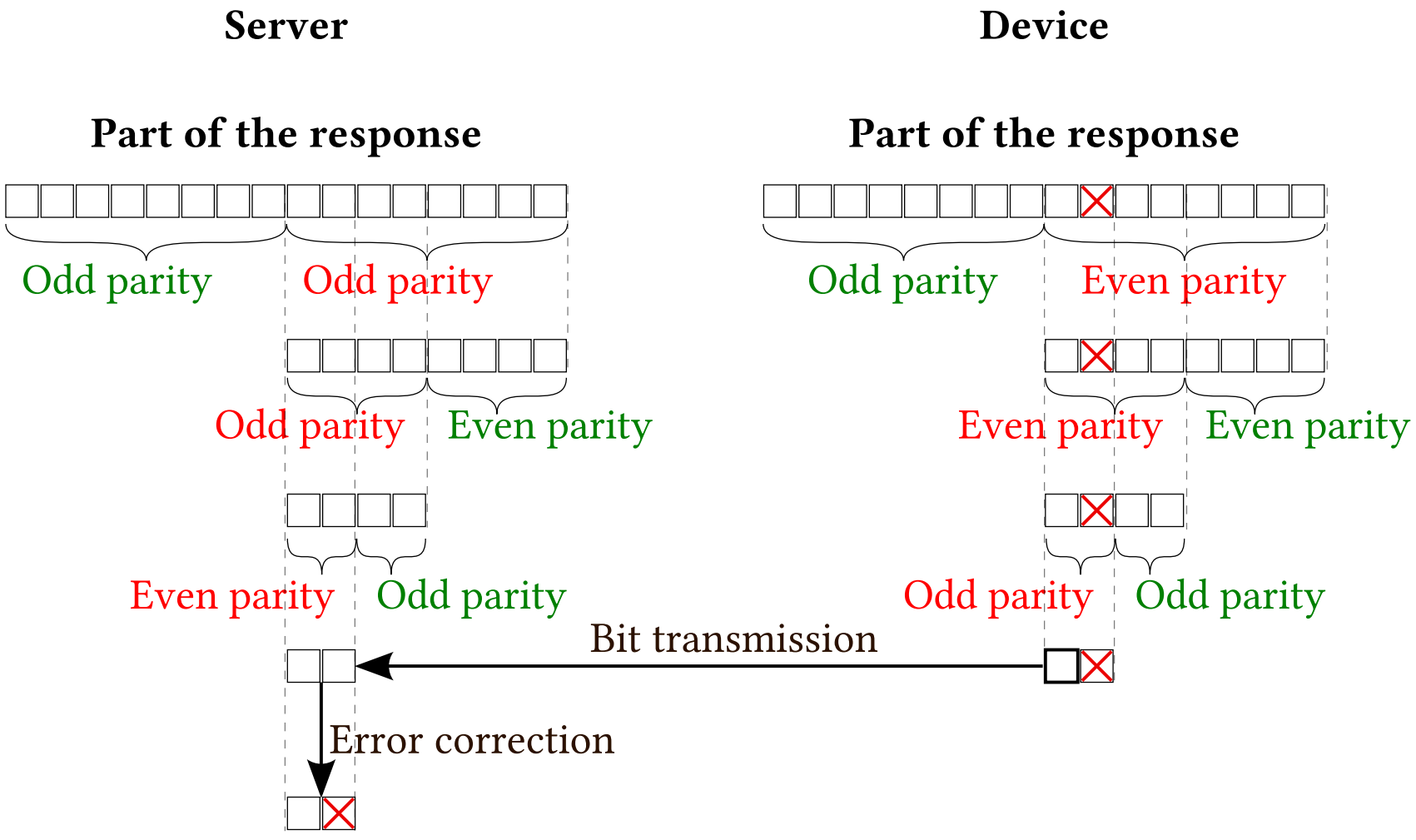
CASCADE introduced in 1993 by Brassard and Salvail [1].



This could be used to correct errors
in slightly different PUF responses

[1] G. Brassard and L. Salvail, *Secret-Key Reconciliation by Public Discussion*, **EUROCRYPT**, 1993.

CONFIRM: dichotomy-based error correction



Allows to correct **one error** per pass.

Relative parity

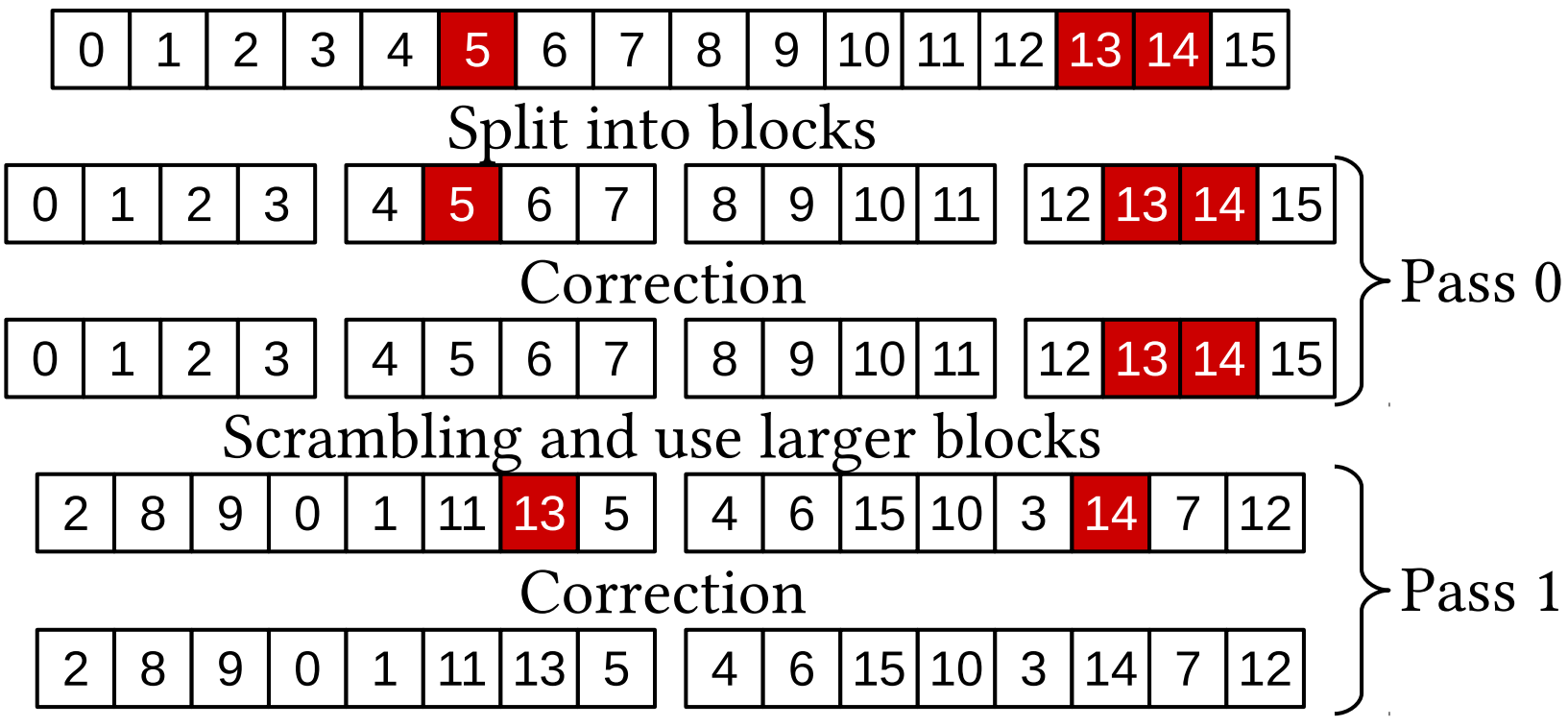
The relative parity of two vectors V_1 and V_2 is given by:

$$P_r(V_1, V_2) = \underbrace{\left(\bigoplus_{i=1}^{|V_1|} V_1\right)}_{\text{Parity of } V_1} \oplus \underbrace{\left(\bigoplus_{i=1}^{|V_2|} V_2\right)}_{\text{Parity of } V_2}$$

If they are **identical** or differ in an **even** number of errors:
→ their relative parity is 0.
If they differ in an **odd** number of errors:
→ their relative parity is 1.

BINARY: using CONFIRM to correct multiple errors

BINARY method:



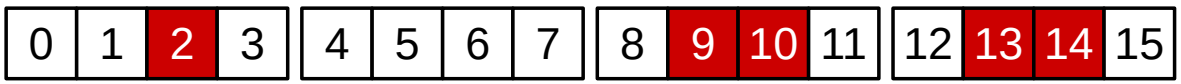
Backtracking

After a pass, all the blocks have an **even** relative parity.
→ if an error is corrected on a bit from this block in a subsequent pass, then its relative parity becomes **odd** again.
→ one more error from this block can be corrected.

Required: two lists storing blocks according to their relative parity.

Correcting an error at index i makes blocks containing index i move from one list to the other (**their relative parity changed**).
Error correction is carried out until there are **no more blocks of odd parity**.

The CASCADE protocol



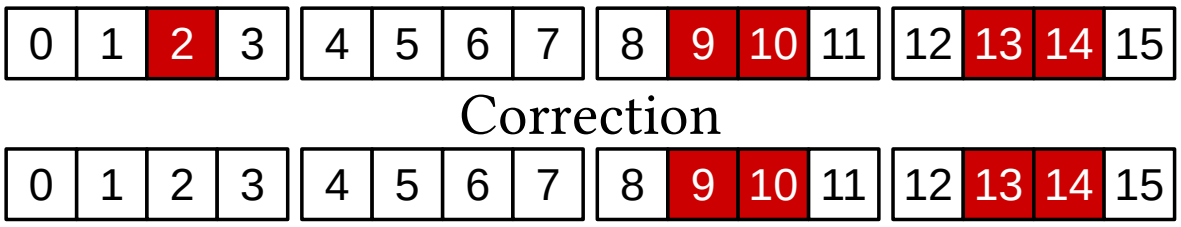
Blocks of even relative parity:

$\varnothing$

Blocks of odd relative parity:

$\varnothing$

The CASCADE protocol



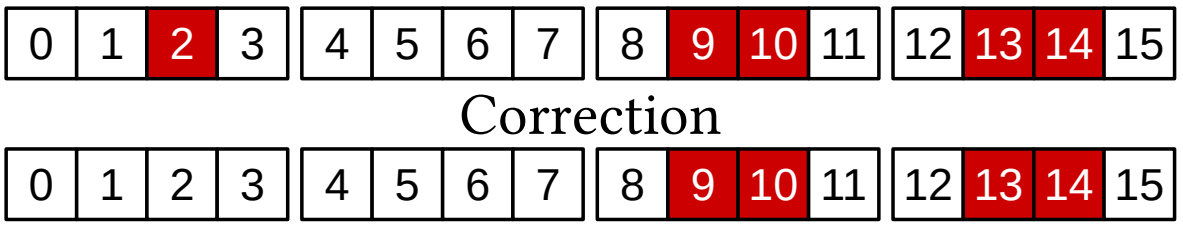
Blocks of even relative parity:

$\varnothing$

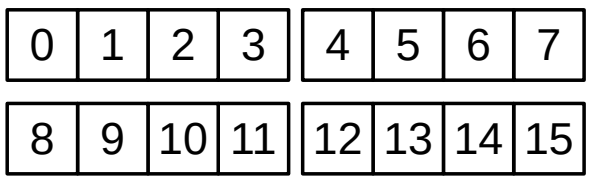
Blocks of odd relative parity:

$\varnothing$

The CASCADE protocol



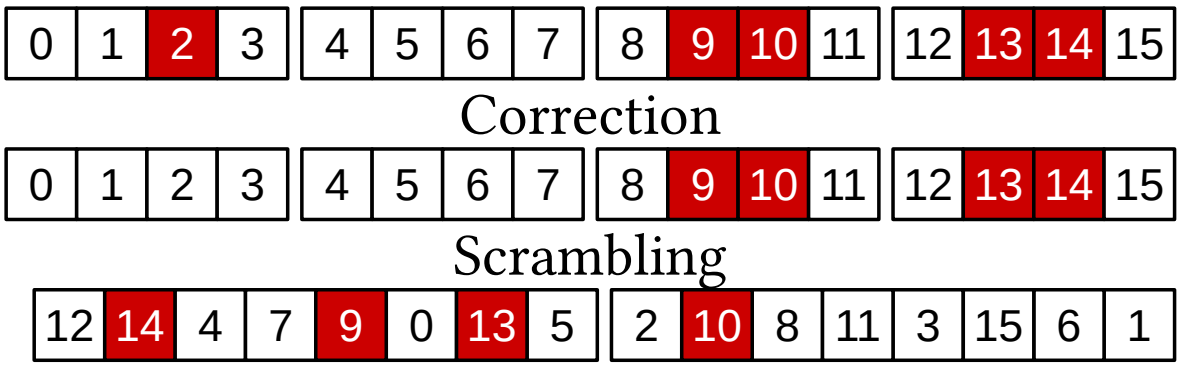
Blocks of even relative parity:



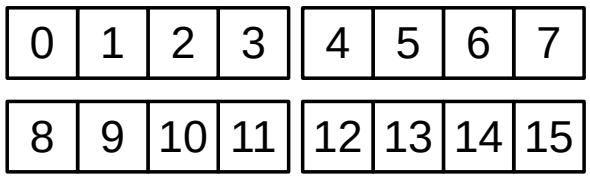
Blocks of odd relative parity:

$\varnothing$

The CASCADE protocol



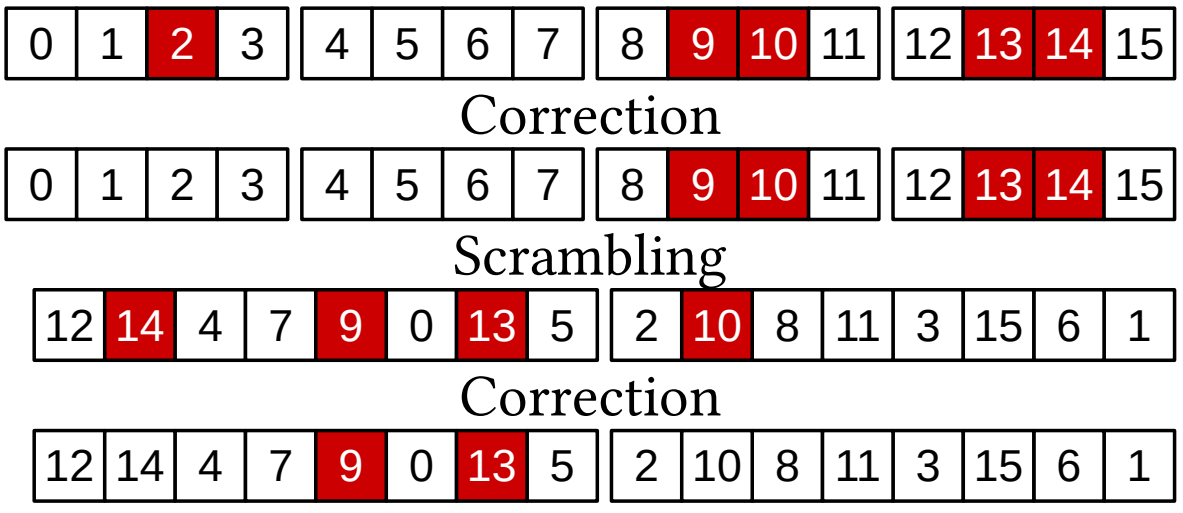
Blocks of even relative parity:



Blocks of odd relative parity:

$\varnothing$

The CASCADE protocol



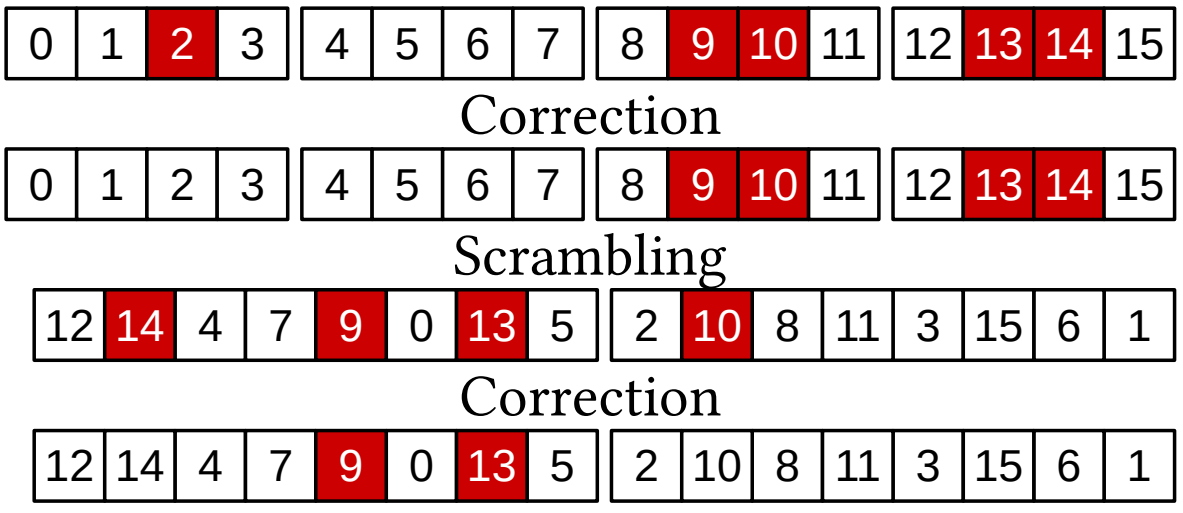
Blocks of even relative parity:

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

Blocks of odd relative parity:

$\varnothing$

The CASCADE protocol



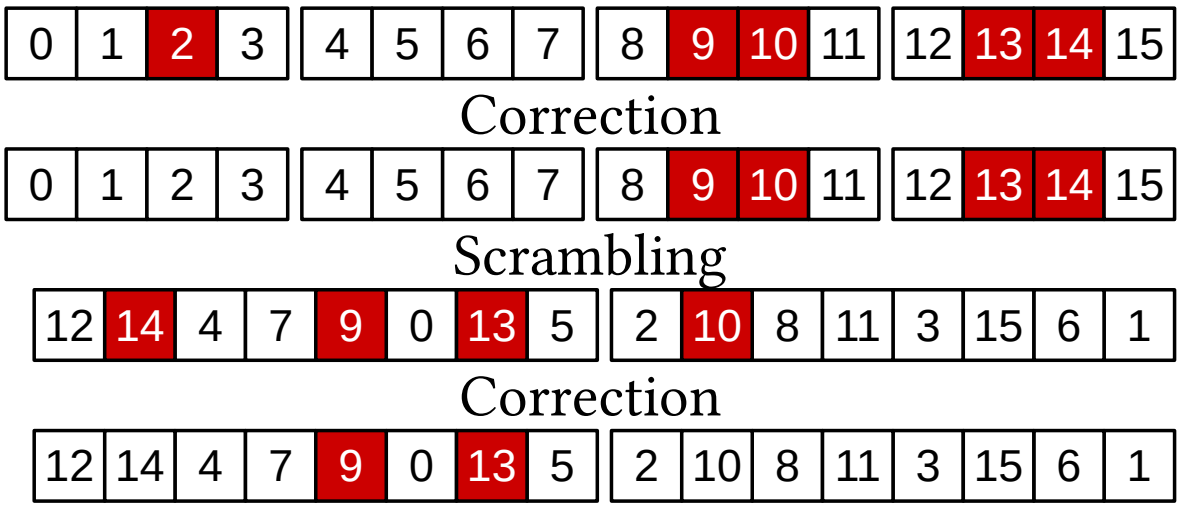
Blocks of even relative parity:

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

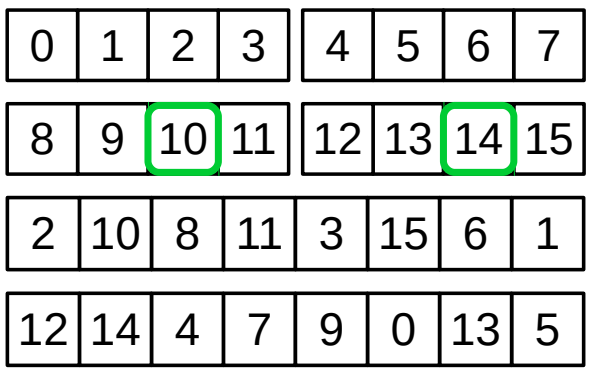
Blocks of odd relative parity:

$\varnothing$

The CASCADE protocol



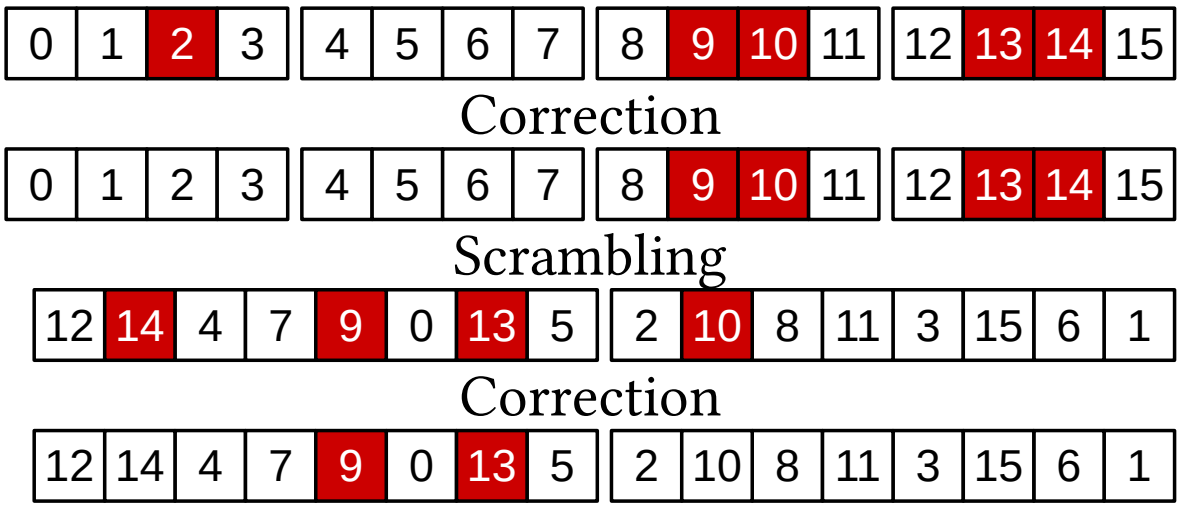
Blocks of even relative parity:



Blocks of odd relative parity:

$\varnothing$

The CASCADE protocol



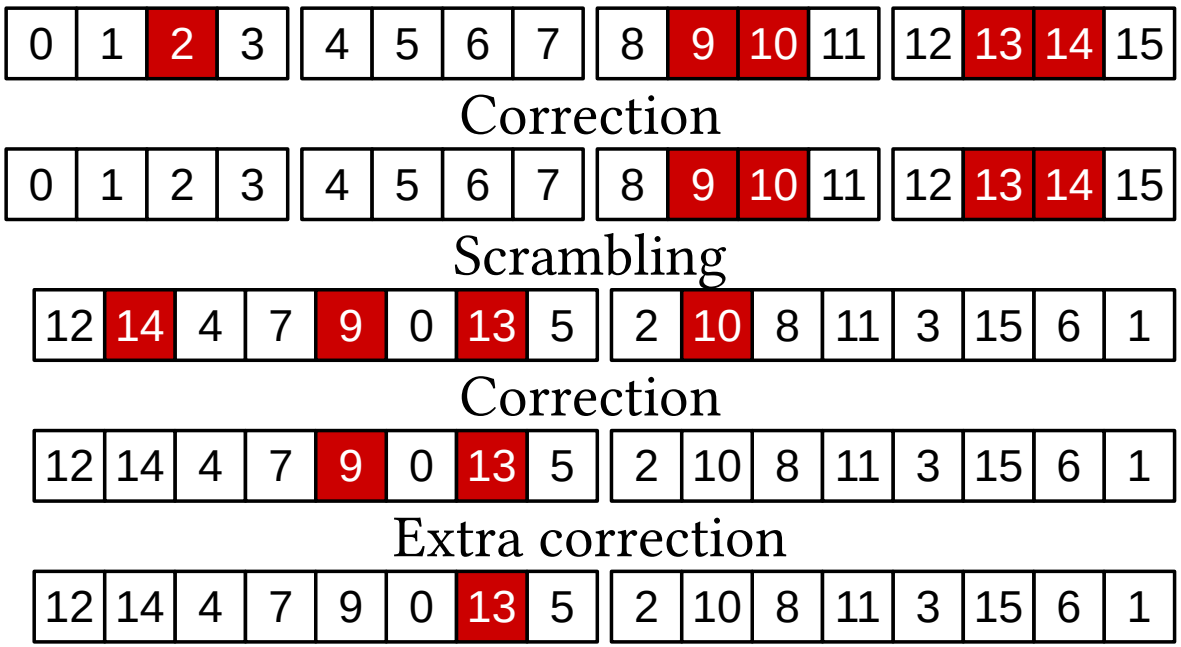
Blocks of even relative parity:

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

The CASCADE protocol



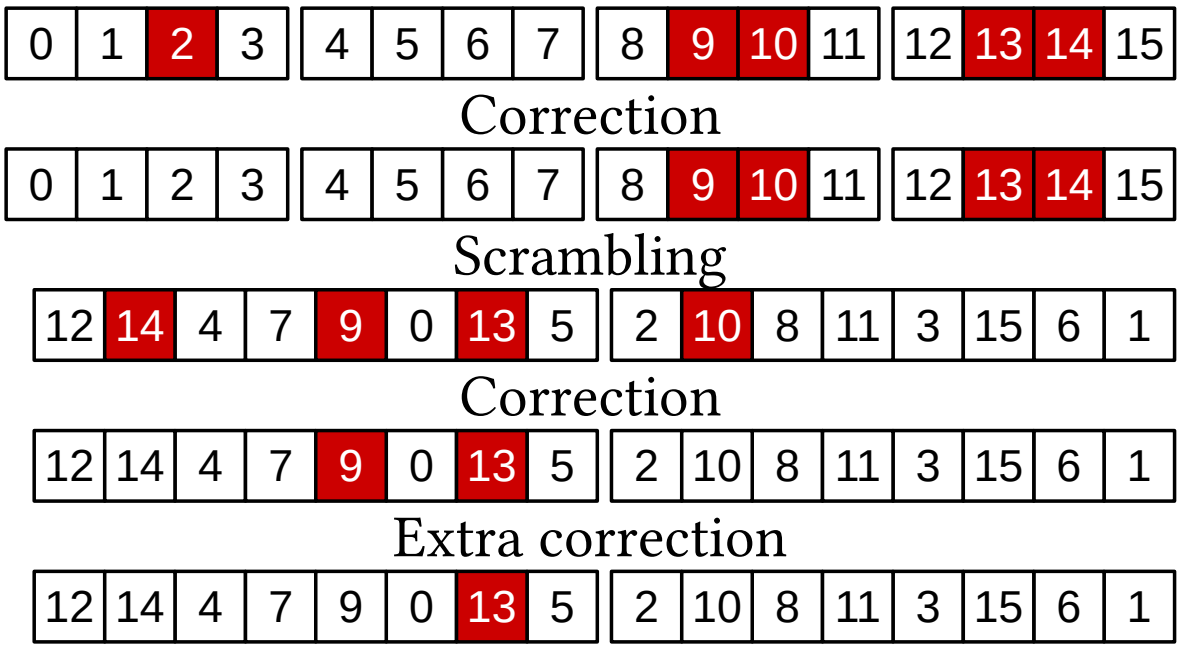
Blocks of even relative parity:

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

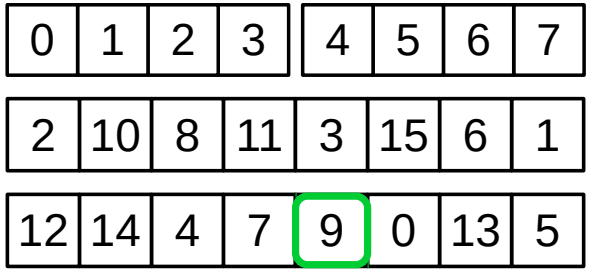
Blocks of odd relative parity:

8	9	10	11	12	13	14	15

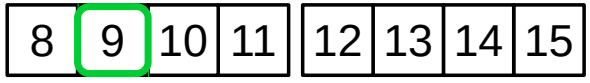
The CASCADE protocol



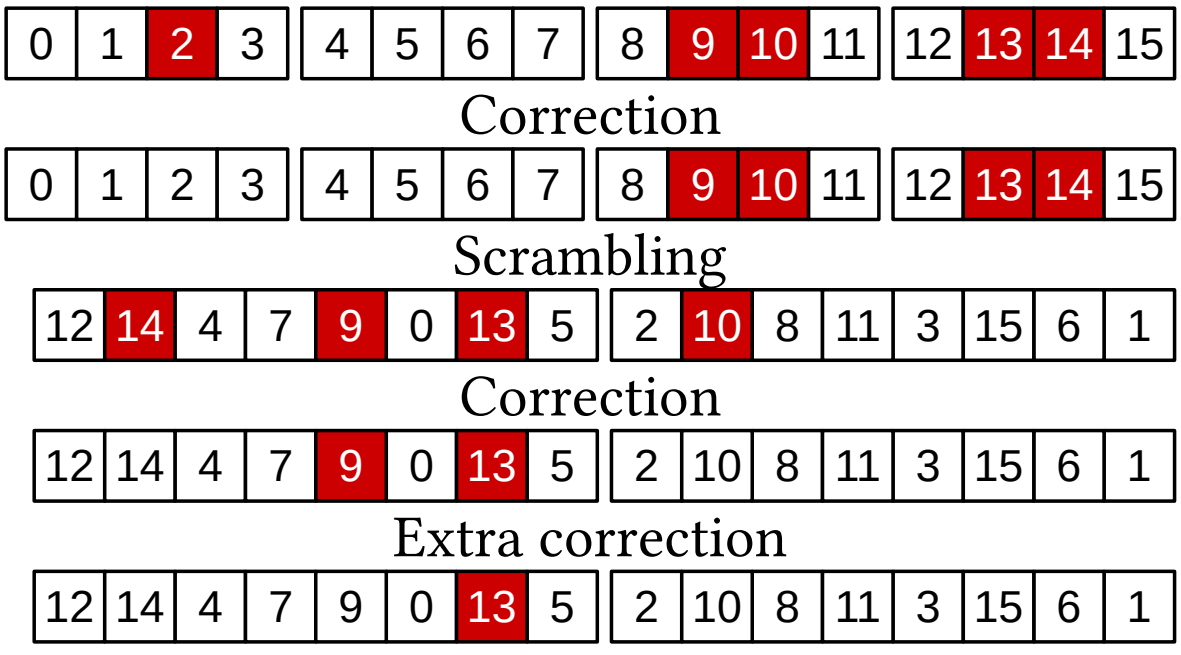
Blocks of even relative parity:



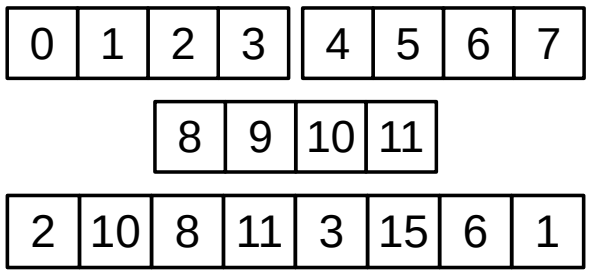
Blocks of odd relative parity:



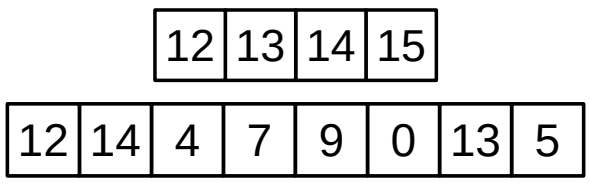
The CASCADE protocol



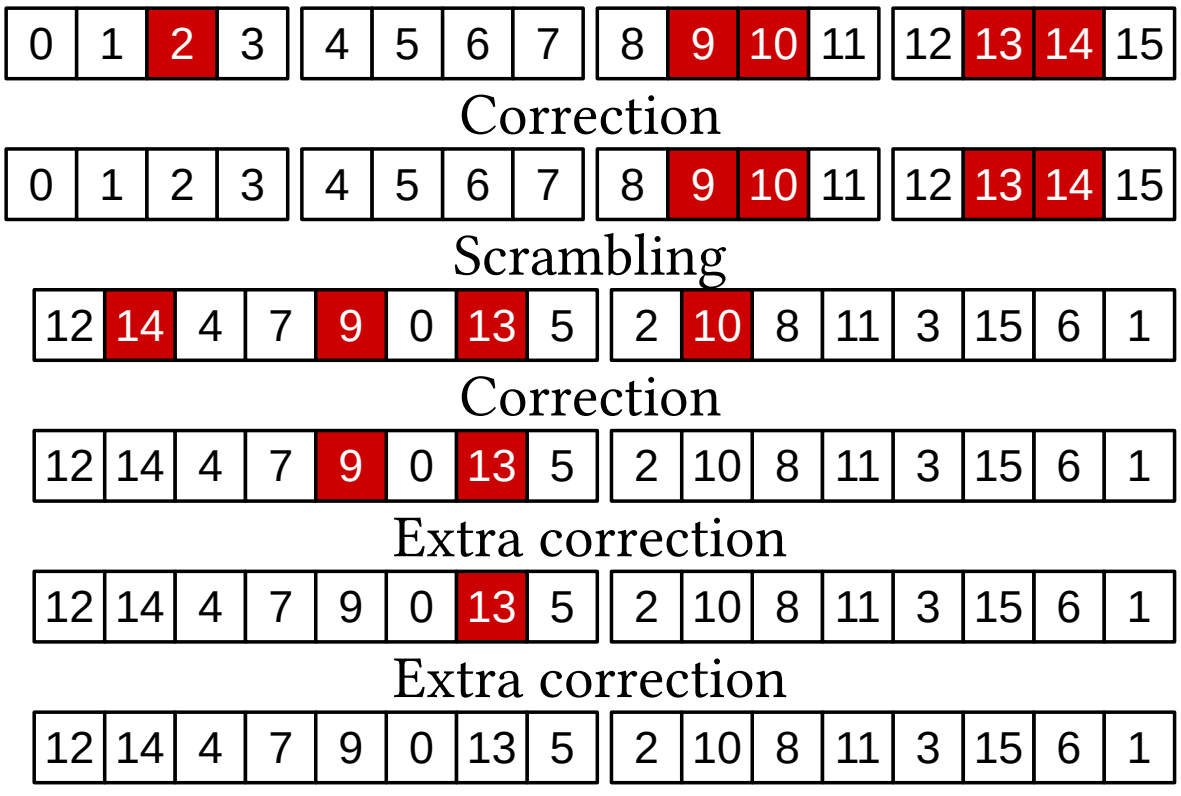
Blocks of even relative parity:



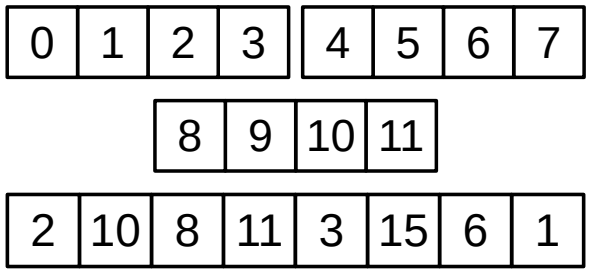
Blocks of odd relative parity:



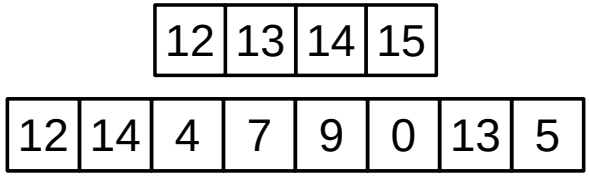
The CASCADE protocol



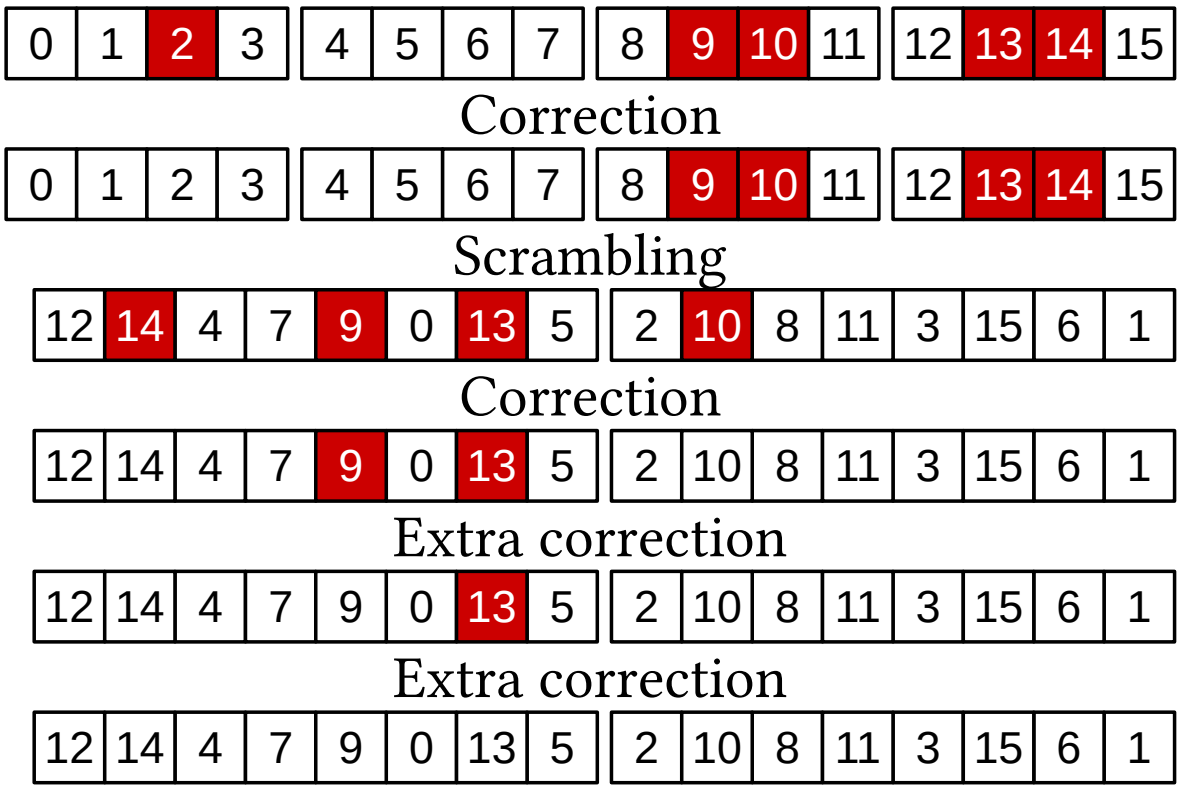
Blocks of even relative parity:



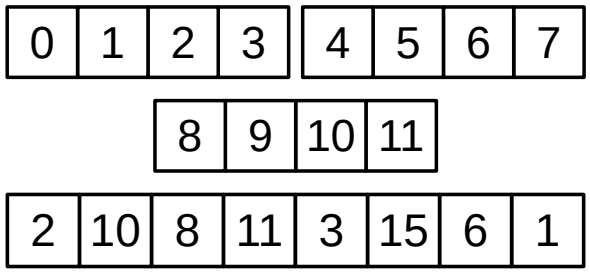
Blocks of odd relative parity:



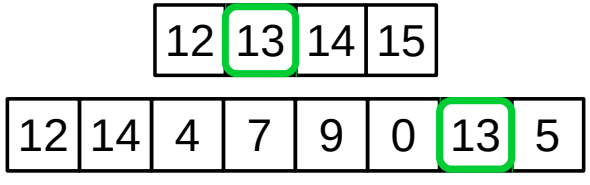
The CASCADE protocol



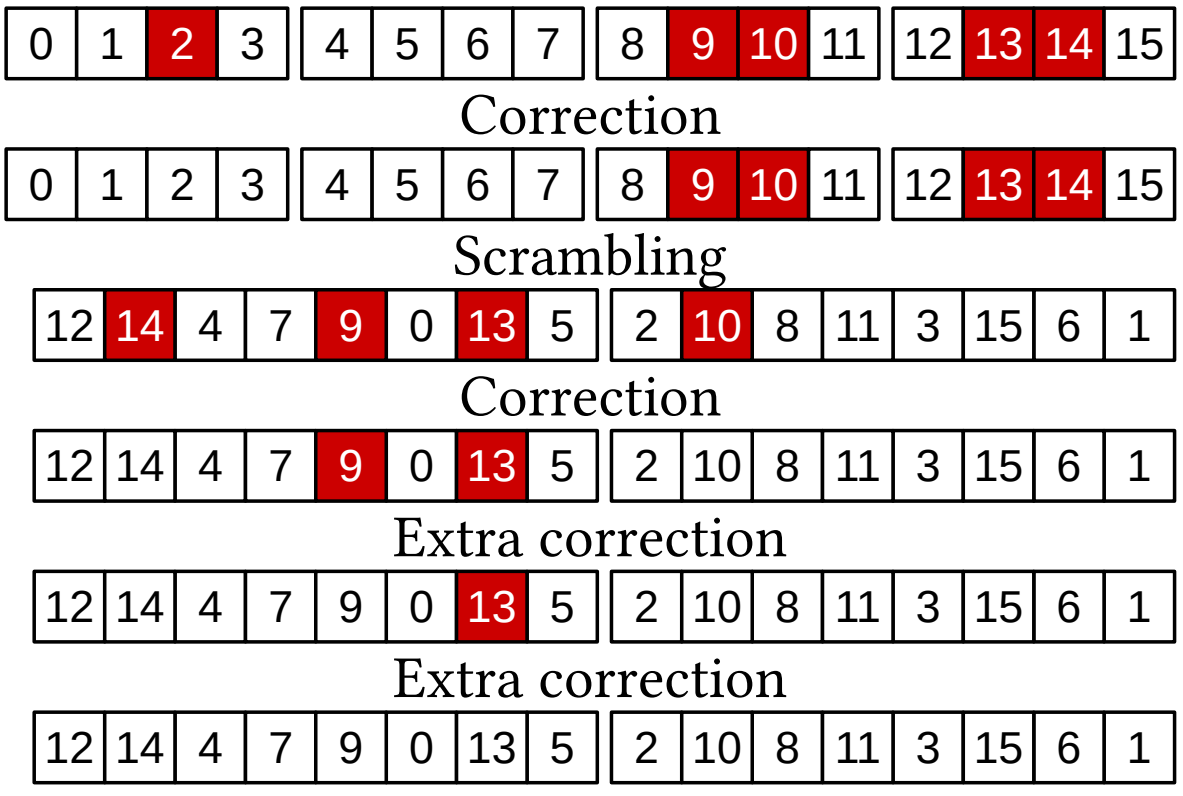
Blocks of even relative parity:



Blocks of odd relative parity:



The CASCADE protocol



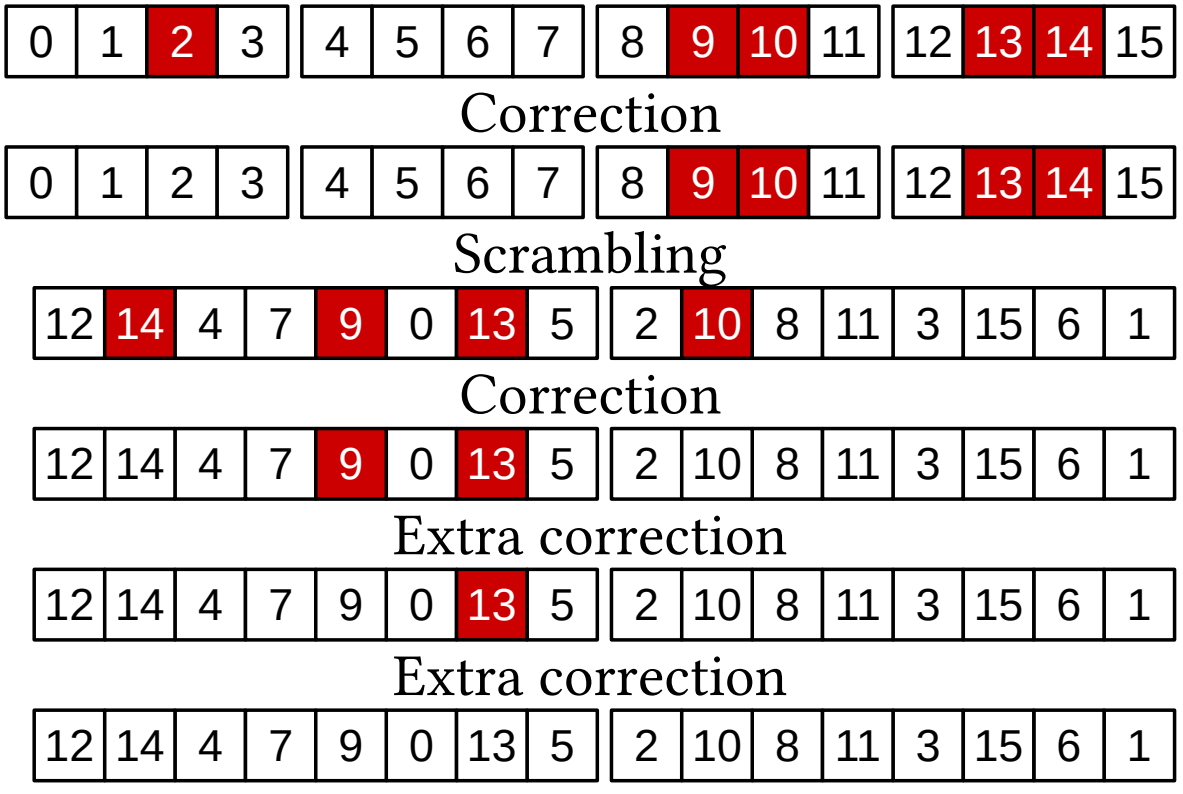
Blocks of even relative parity:

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

Blocks of odd relative parity:

$\varnothing$

The CASCADE protocol



Blocks of even relative parity:

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

Blocks of odd relative parity:

$\varnothing$

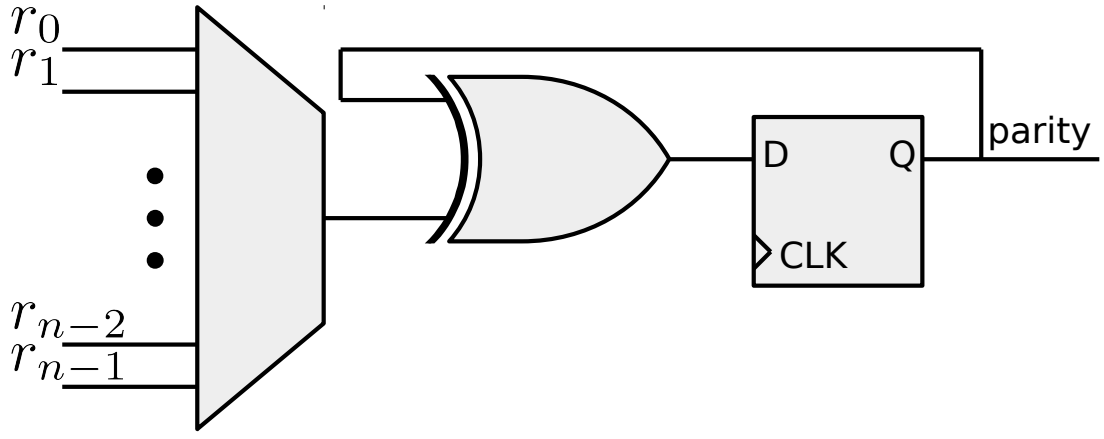
For the **same number of passes**, the CASCADE protocol allows to correct **more errors** than BINARY.

Implementation

Only **parity computations** are embedded on the device.
All other computations can be done **on the server**.

Requirements:

- A multiplexer,
- One XOR gate,
- One D flip-flop.



Example: 256-bit response

Xilinx Spartan 6: 19 Slices

State-of-the-art error correcting codes implementation:

Reed-Muller (4, 7): 179 Slices
BCH (255, 21, 55): >60 Slices

Conclusion

Compared to existing error correcting codes implementations:

- similar latency,
- can reach the same very low failure rates (10^{-6}),
- leakage is easy to estimate and control,
- at least 3x fewer on-chip logic resources.

Interesting option for error-correction of PUF responses.

Conclusion

Compared to existing error correcting codes implementations:

- similar latency,
- can reach the same very low failure rates (10^{-6}),
- leakage is easy to estimate and control,
- at least 3x fewer on-chip logic resources.

Interesting option for error-correction of PUF responses.

— Questions? —