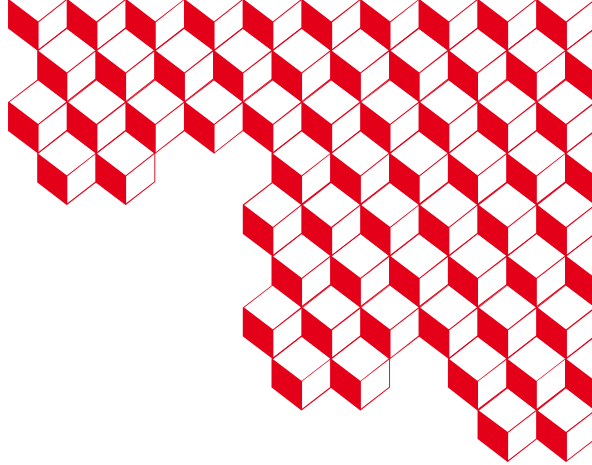


Combining code polymorphism and loop shuffling

Nicolas Belleville, Loïc Masure

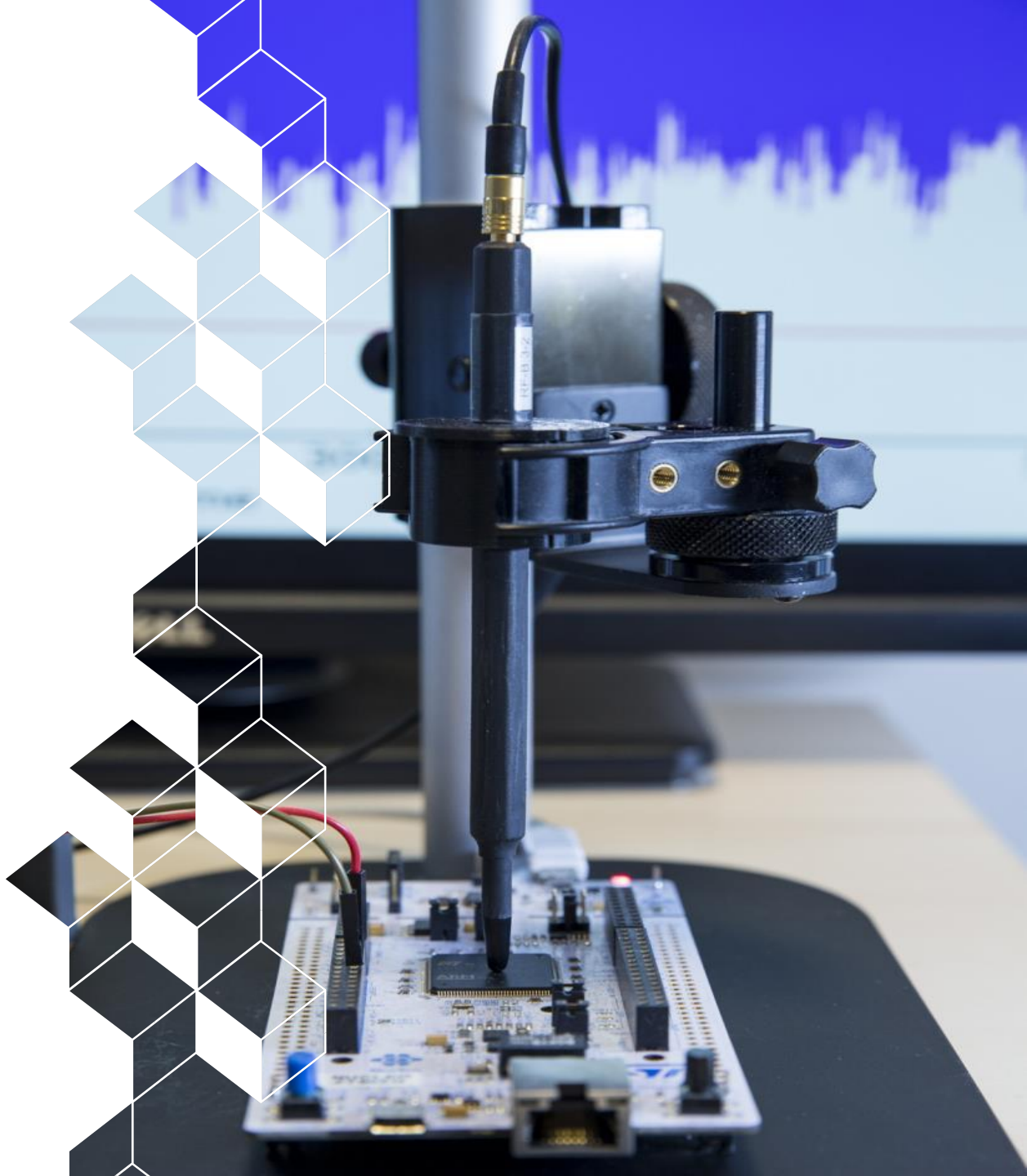
Co-funded by
the European Union

1. Side-channel attacks & countermeasures

Side-channel attacks

Observe a physical quantity to infer information about instructions or data being manipulated by the system.

- Power consumption, electromagnetic emission, acoustic noise, …
- Massive threat for embedded systems security
- Very efficient against cipher implementations



Countermeasures

Masking

- **Split secret data into *shares***, such that at least *d* shares are needed to learn information about the secret (*d+1* is called the masking order).
- Various splitting techniques have been considered in the state of the art.

Most countermeasures **have been studied in isolation** in state of the art.

Combining countermeasures ***may*** improve security

Hiding

- **Lower signal to noise ratio** by lowering signal or increasing noise
- **Shuffling** (loops)
  - "Randomly change the sequence of those operations of a cryptographic algorithm that can be performed in arbitrary order"
- **Code polymorphism**
  - "Capability to regularly change the behaviour of a secured component at runtime without altering its functional properties"

2. (Loop) Shuffling

Shuffling:

> **Randomly change** the sequence of those operations of a cryptographic algorithm that can be **performed in arbitrary order**.”

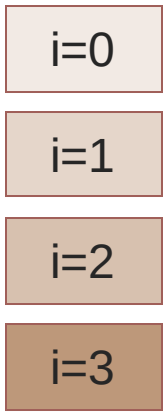
From:

Power Analysis Attacks: Revealing the Secrets of Smart Cards.
S. Mangard et al.
2007

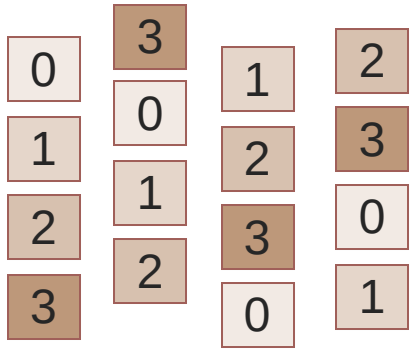
Shuffling in practice

```
for(int i = 0; i < 4; i++)
{
    //process
}
```

Without shuffling:



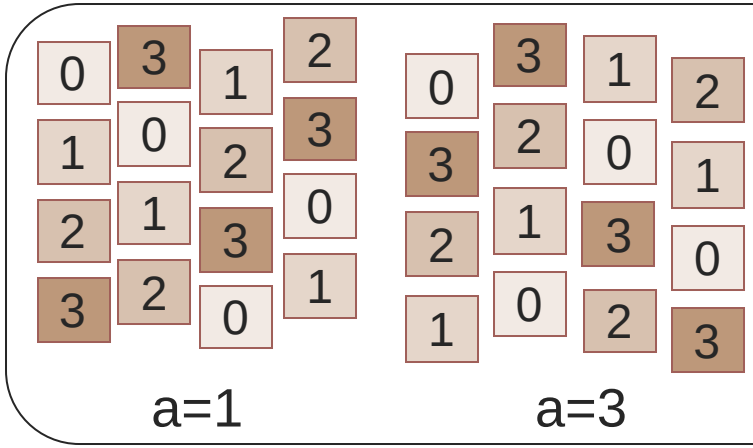
Random start index



$r = rand();$
$i_r = (i + r) \% 4;$

4 permutations

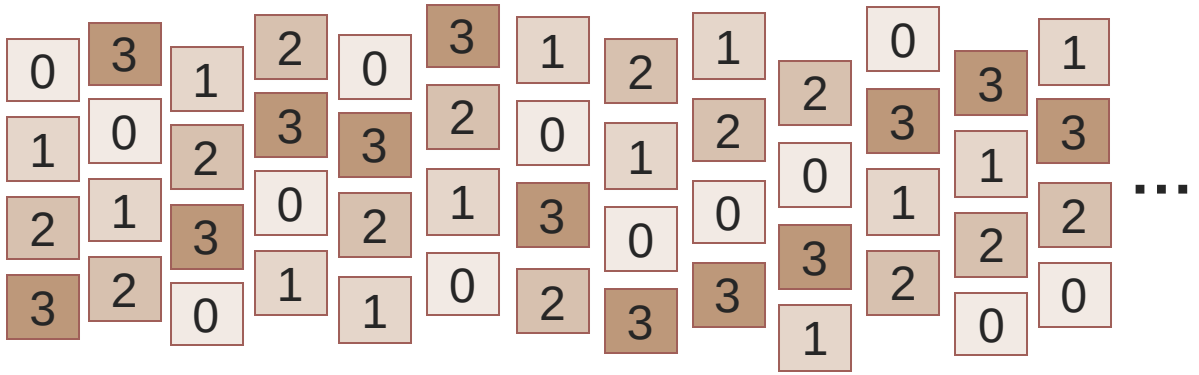
Random walk



$r = rand();$
$a = odd\_rand();$
$i_{ra} = (i * a + r) \% 4;$

8 permutations

Random permutation



Fisher-Yates
(for instance)

24 permutations

3. Code polymorphism

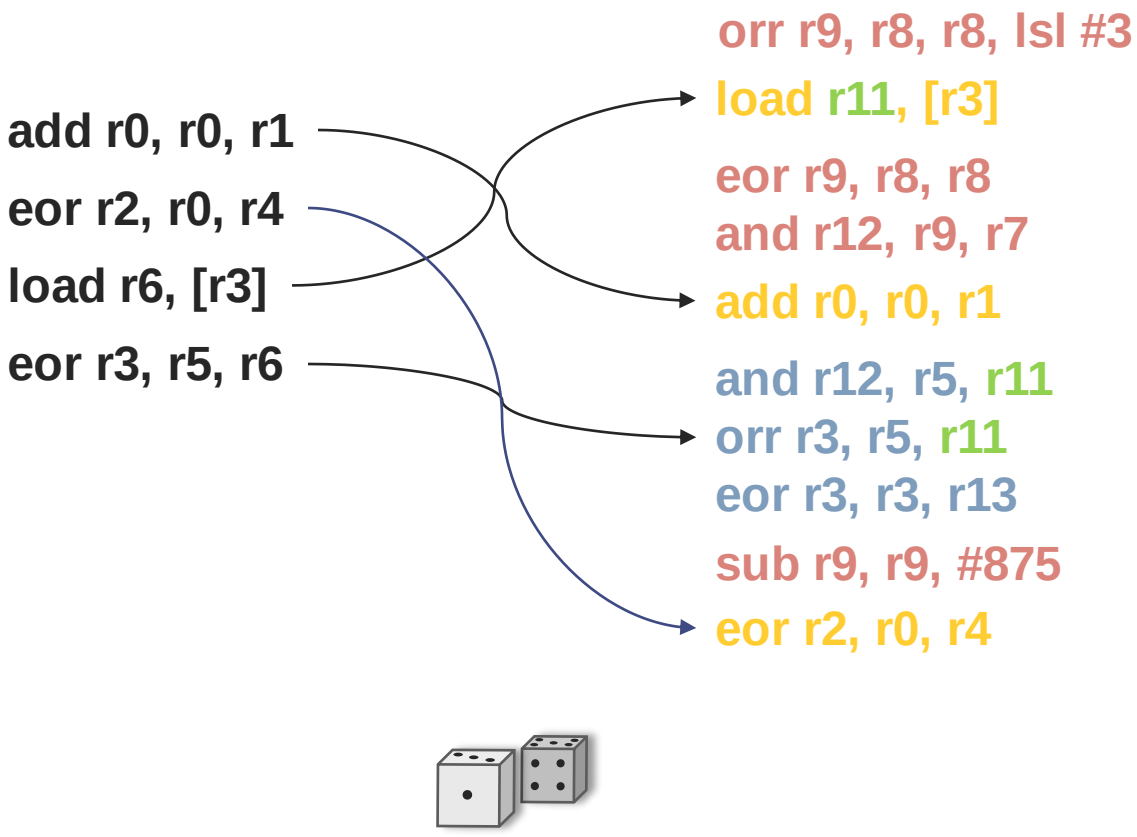
Code polymorphism:

> **“Capability to regularly change the behaviour of a secured component at runtime without altering its functional properties”**

From:

Runtime Code Polymorphism as a Protection Against Side Channel Attacks.
D. Couroussé et al.
WISTP 2016

Code polymorphism: runtime code generation



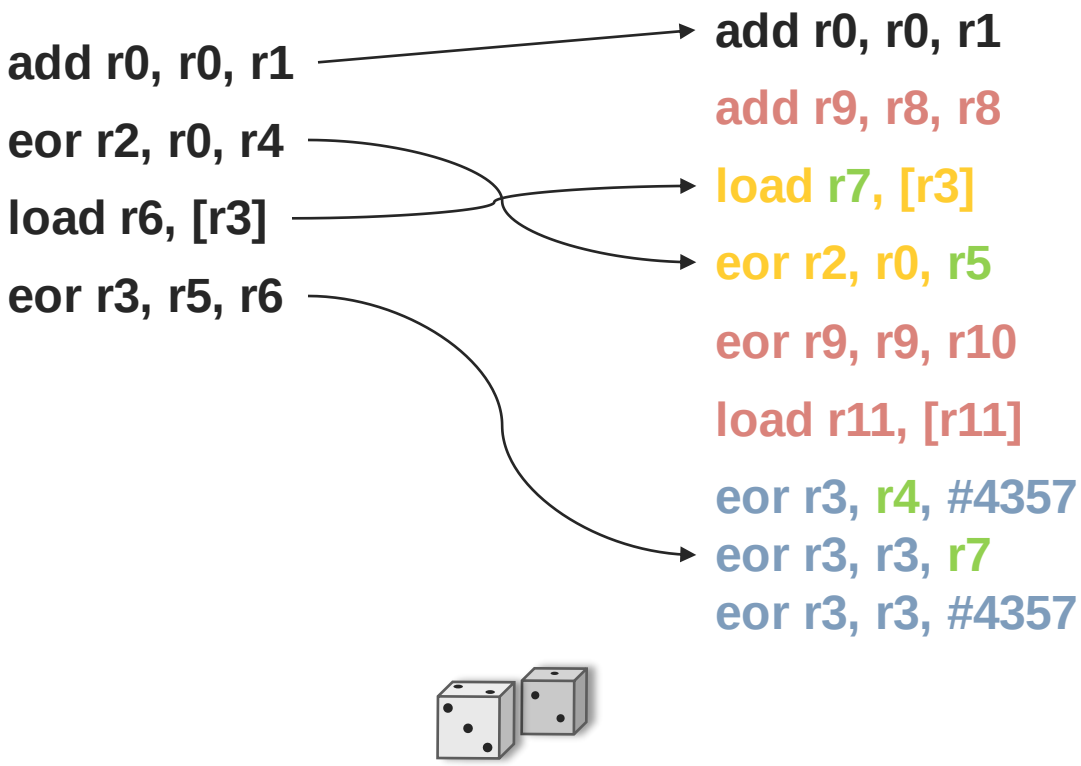
noise instructions

instruction shuffling

semantic variants

register permutation

Code polymorphism: runtime code generation



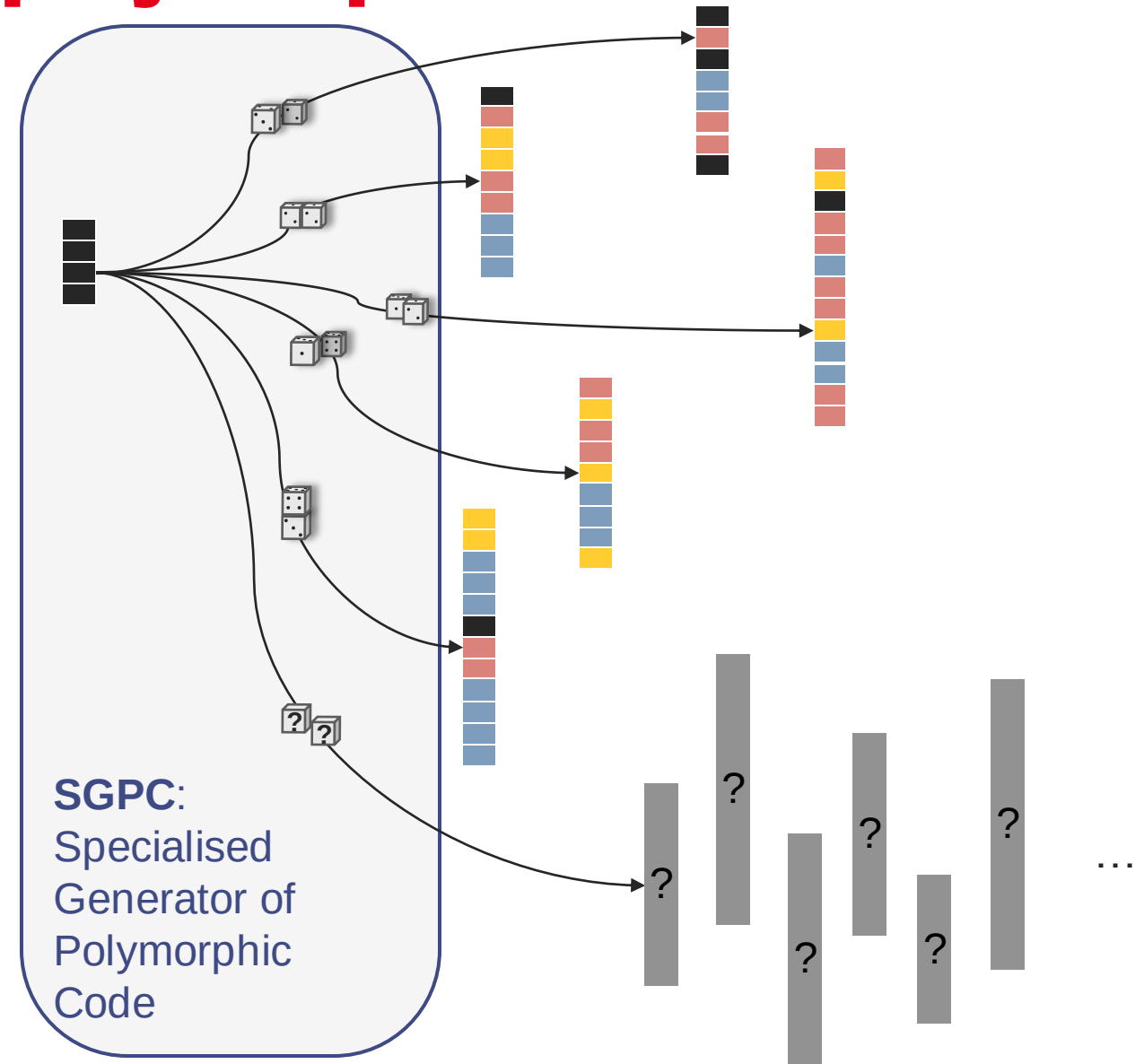
noise instructions

instruction shuffling

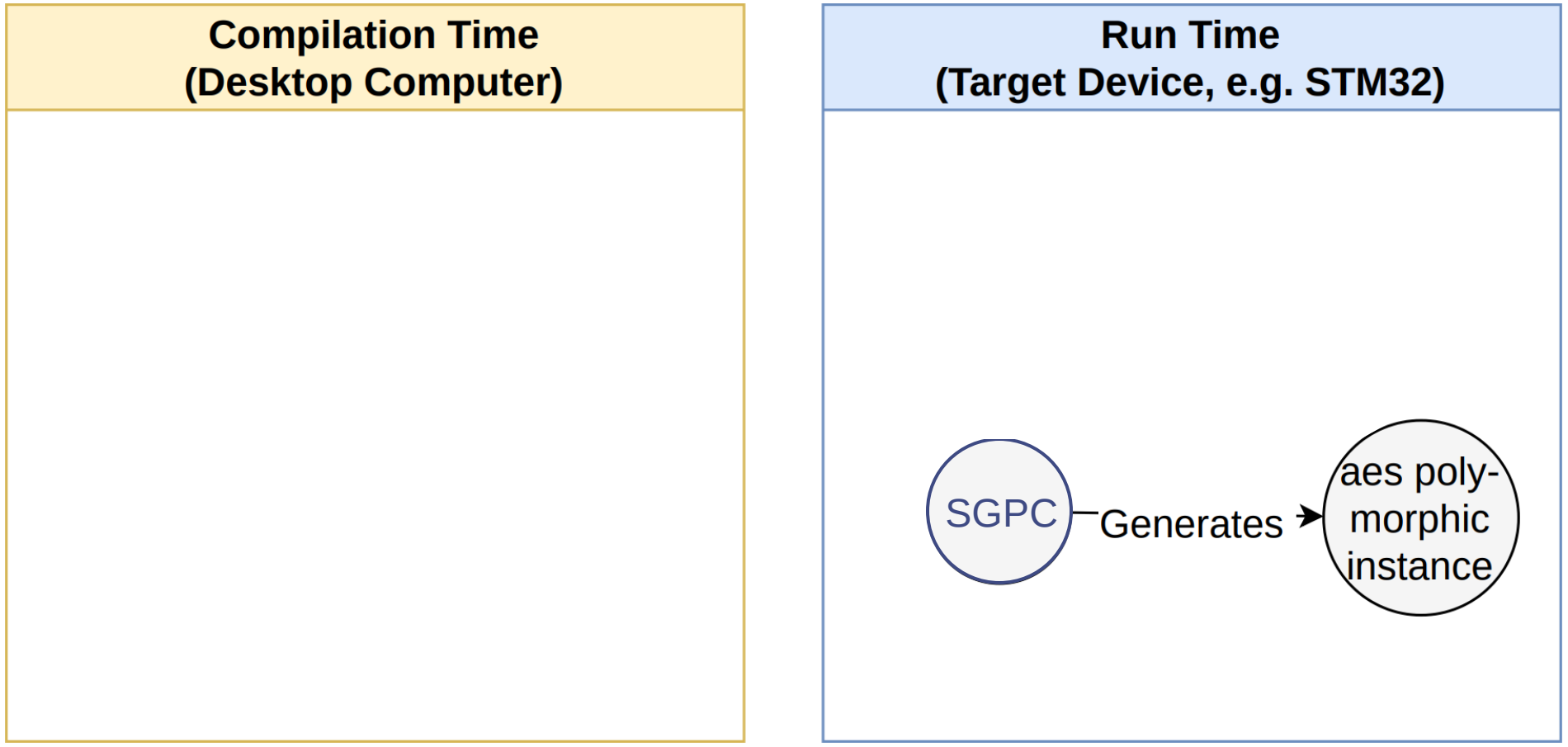
semantic variants

register permutation

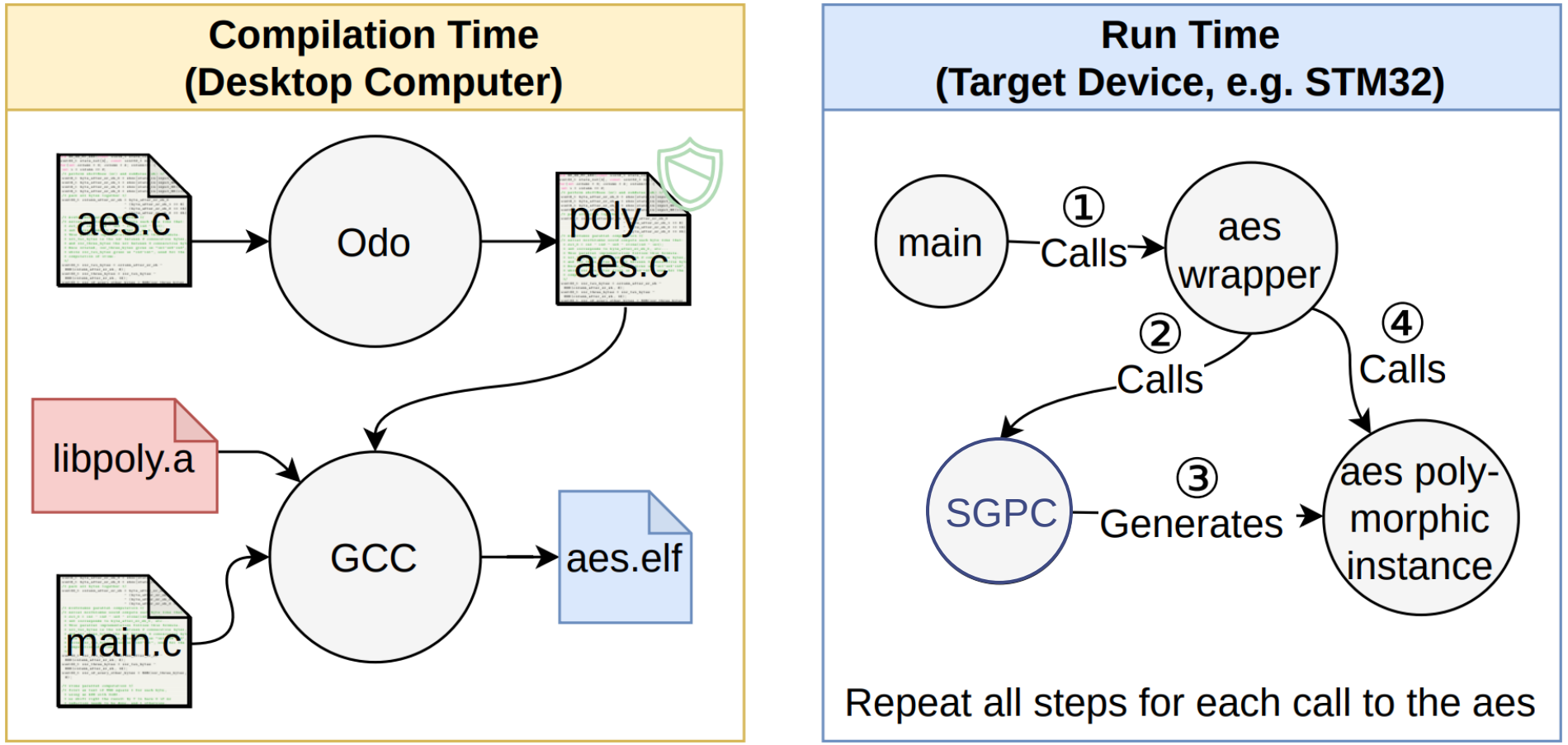
Code polymorphism: runtime code generation



Code polymorphism: flow overview



Code polymorphism: flow overview



Our approach:

4. Combining both countermeasures

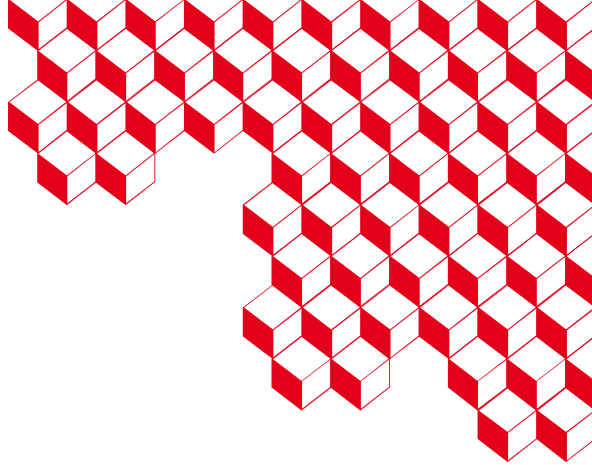
Intuition

Loop shuffling and code polymorphism act on **different granularities**:

- large code chunks for loop shuffling
- assembly instructions for code polymorphism

Intuitively, it *should* be worth combining them.

Research question



How combining shuffling and code polymorphism impacts resulting security?

5. Results

Experimental setup

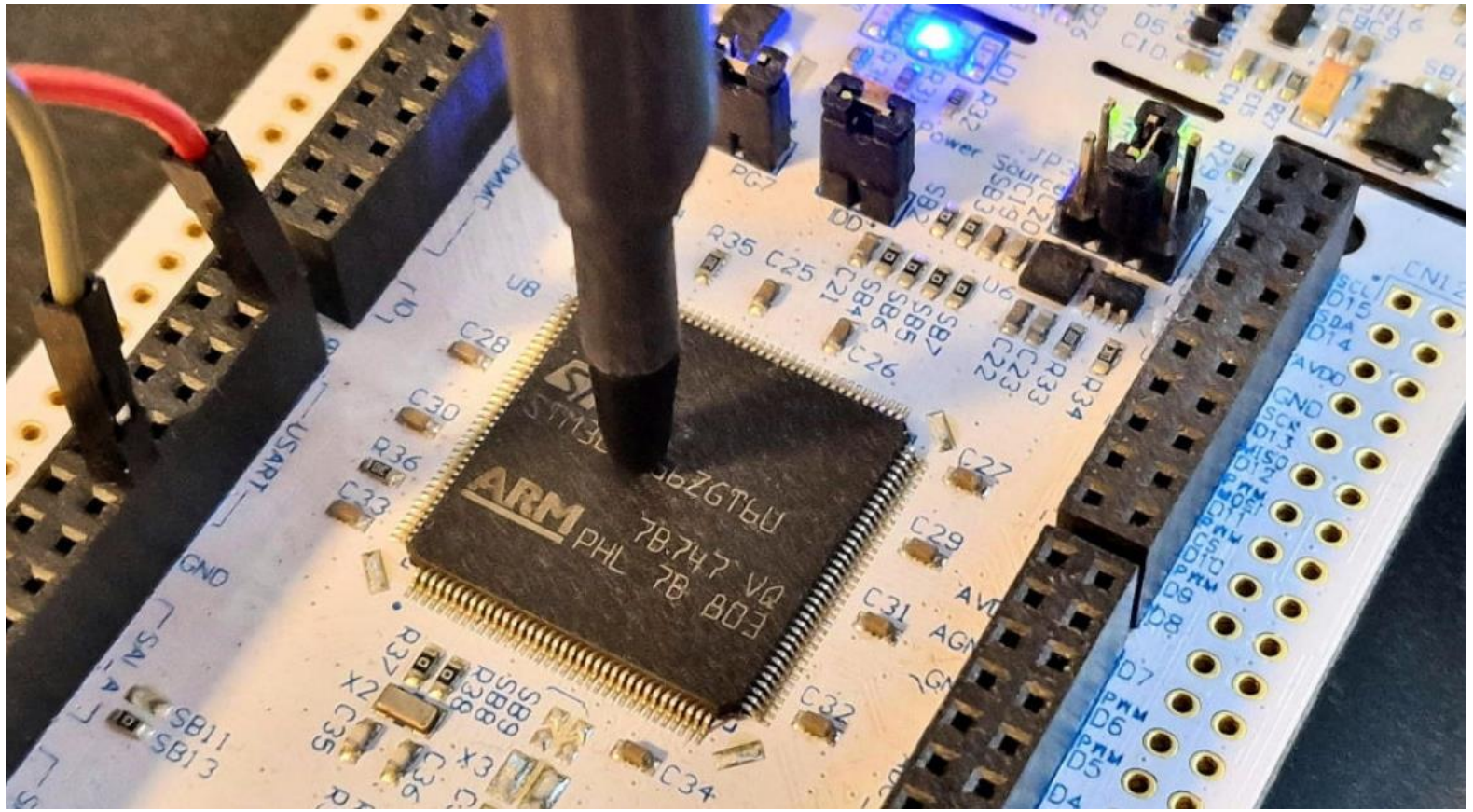
- AES use case with custom implementation

- STM32F7 with ARM Cortex M7, 166.66MHz
- Picoscope 5244B, 500Msamples/s

  ➔ 3 samples per clock cycle
- Langer RF-B 3-2 EM probe
- Langer preamplifier PA 303

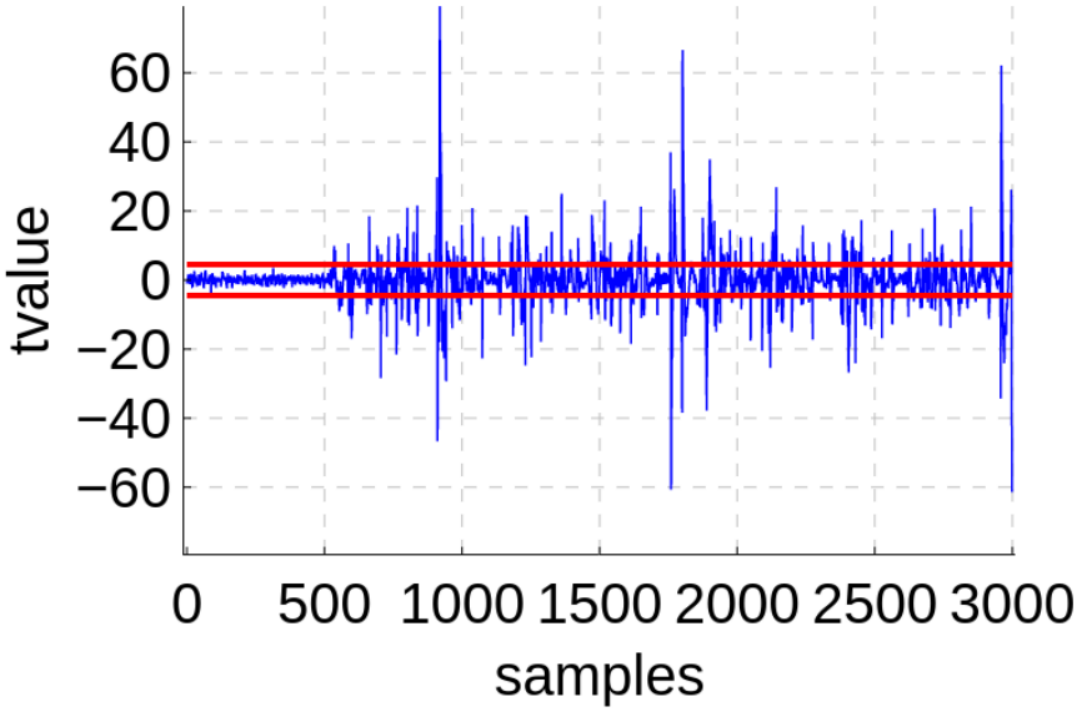
- Probe placement checked via 50k vs 50k non-specific ttest on unprotected AES:
  - Maximum tvalue: **223.90**

  ➔ Strong leakage

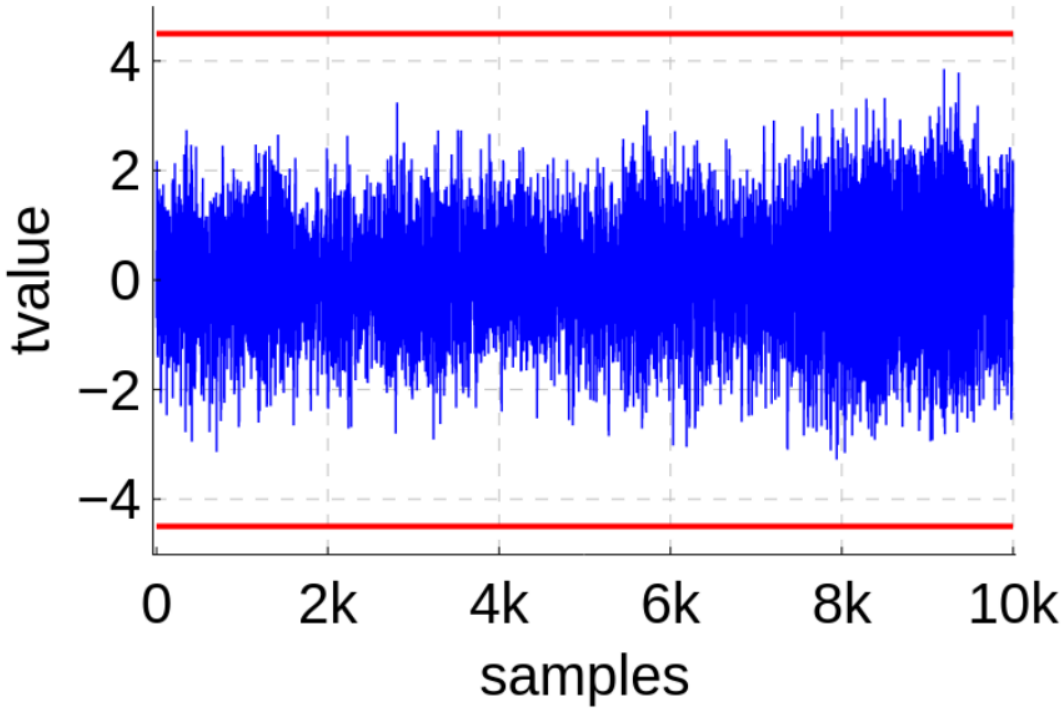


Permutation leakage

Non-specific ttest on shuffling's permutation variable



(a) Loop shuffling only.



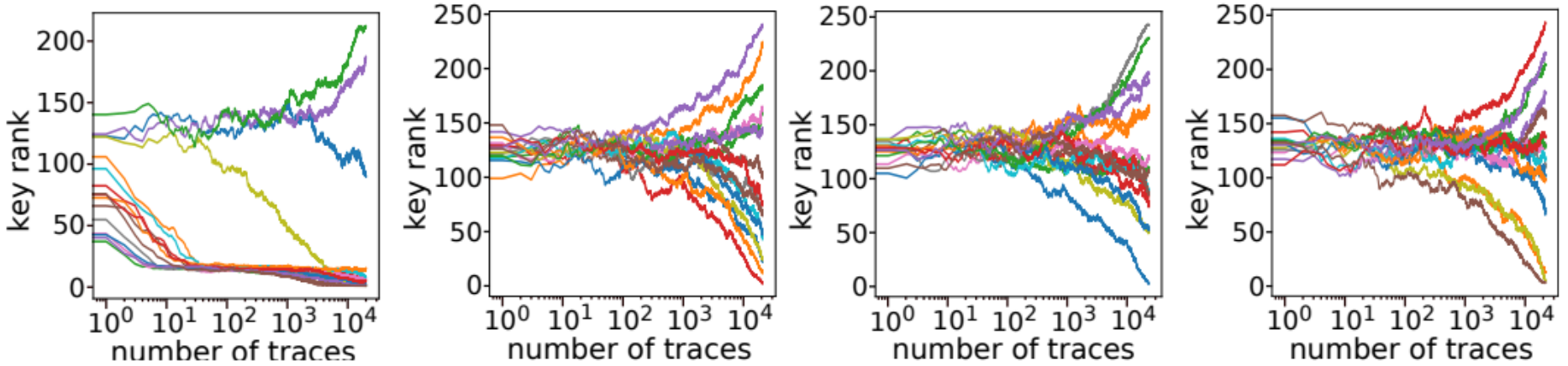
(b) With code polymorphism.

Resistance to CPA with integration and deep learning attack

CPA with integration (up to 500k traces)

Configuration	Target key byte															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Unprotected	3k	6k	1k	2k	3k	2k	4k	1k	2k	3k	2k	4k	4k	3k	4k	2k
Loop shuffling	28k	32k	35k	18k	38k	28k	42k	45k	20k	18k	35k	20k	35k	40k	50k	45k
Code polymorphism	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
All countermeasures	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

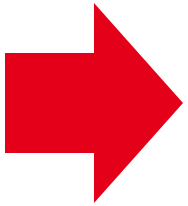
Deep learning attack (80k training, 20k validation)



(a) Unprotected (b) Loop shuffling (c) Code polymorphism (d) All countermeasures

So what?

- Code polymorphism efficiently hides permutation leakage
- Loop shuffling alone is not enough
- Deep learning attack struggles much more than anticipated
- Both attack fail on code polymorphism alone ➔ impossible to conclude that combining countermeasures gives better security than code polymorphism alone at this point.



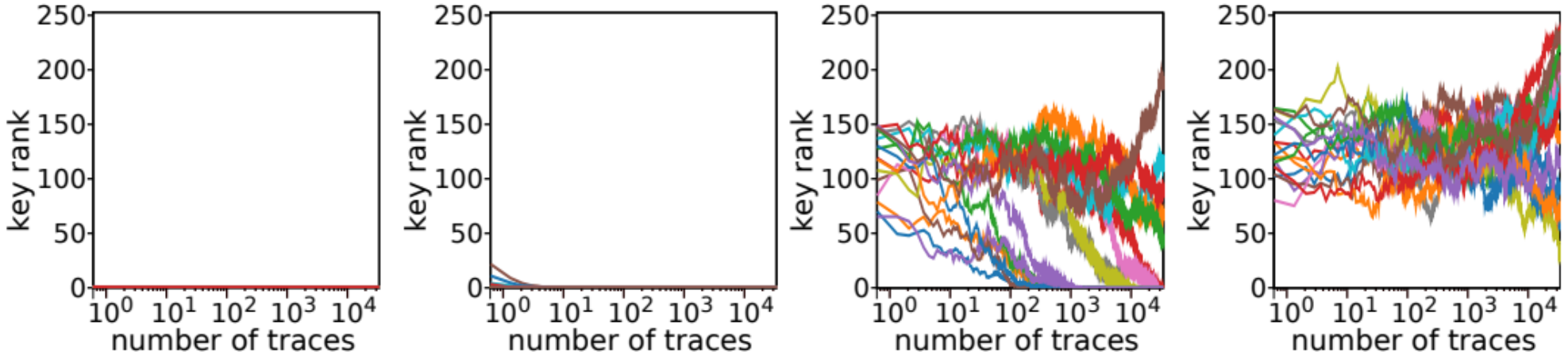
Move on with simulated traces (using a Hamming weight leakage model)
Goal: **absence of noise** to ease attack and ease comparison

Resistance to CPA and deep learning attack on simulated traces

CPA (up to 1M traces)

Config.	Target key byte															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Unprotect.	8	9	9	8	8	8	8	11	8	8	8	10	12	7	8	8
Loop shuf.	1165	921	707	713	1030	1186	1268	615	1104	1060	838	1047	891	604	234	1280
Code poly.	41k	41k	47k	61k	29k	25k	35k	55k	39k	42k	25k	85k	23k	24k	25k	50k
All	595k	215k	752k	369k	>1M	767k	998k	744k	>1M	592k	703k	747k	848k	588k	763k	735k

Deep learning attack (80k training, 20k validation)



(a) Unprotected (b) Loop shuffling (c) Code polymorphism (d) All countermeasures

6. Conclusion

Conclusion & Future works

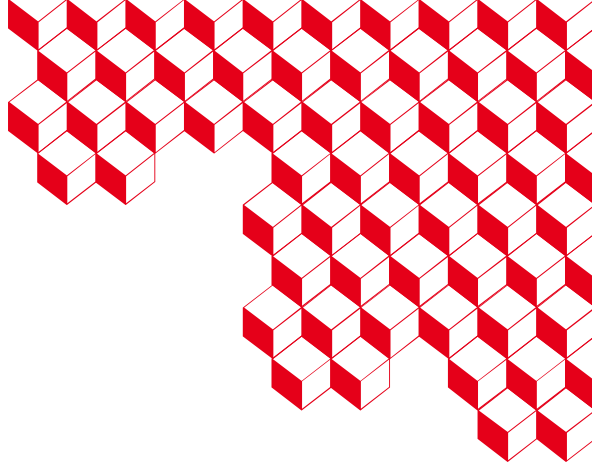
Conclusion

- Combining loop shuffling and code polymorphism induces a significant increase of security!

Future works

- Combine with masking, in particular to protect masked table computation.
- Stack other hiding counter-measures with different granularity (e.g. clock jitter)

	Shuffles large code regions	Desynchronises instructions	Modifies leakage shape
Loop shuffling	✓	✗	✗
Code polymorphism	✗	✓	✓
This paper: Loop shuffling with code polymorphism	✓	✓	✓



Thanks!
Any question?