A Code-Based ISE to Protect Boolean Masking in Software

Qi Tian[1,2] Hao Cheng[1,3] Chun Guo[1,3] Daniel Page[4]
Meiqin Wang[2,1,3] Weijia Wang[2,1,3]

[1]Shandong University

[2]Quan Cheng Laboratory, Jinan, China

[3]Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Qingdao, China

[4]School of Computer Science, University of Bristol, Bristol, UK

September 17, 2025

Outline

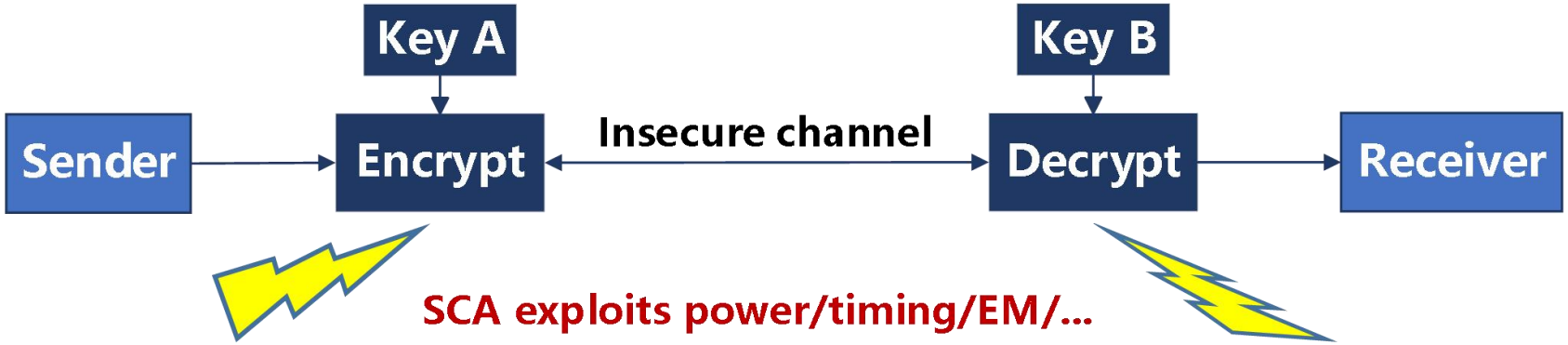
山东大学 SHANDONG UNIVERSITY

Outline

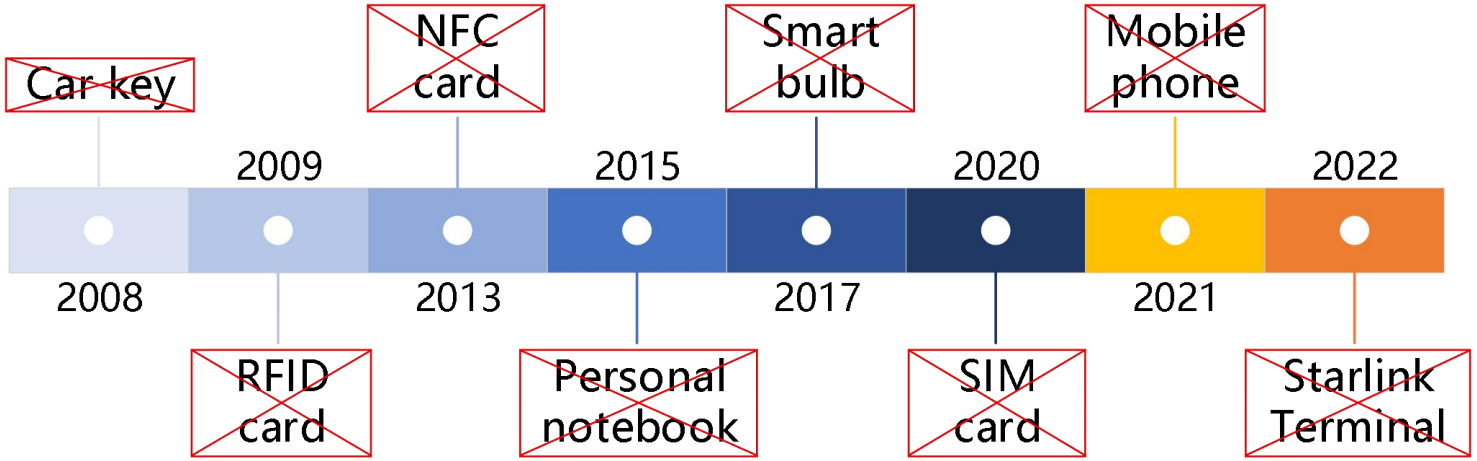
山东大学 SHANDONG UNIVERSITY

Side-channel attack

- SCA can obtain sensitive data indirectly by analyzing the physical information generated during device operation



- SCA poses a significant threat to data security in embedded environments



Masking

- A technique that introduces random numbers to obscure the side-channel information generated during the processing of sensitive data
- Boolean masking is a simple and popular masking technique widely used to protect cryptographic algorithm implementations against side-channel attacks

Split the sensitive data into random, independent shares

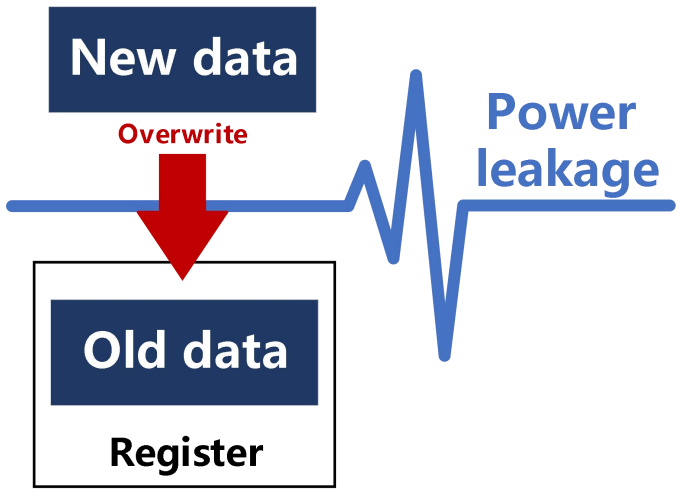
$$x = x_0 \oplus x_1 \oplus \ldots\ldots \oplus x_d$$

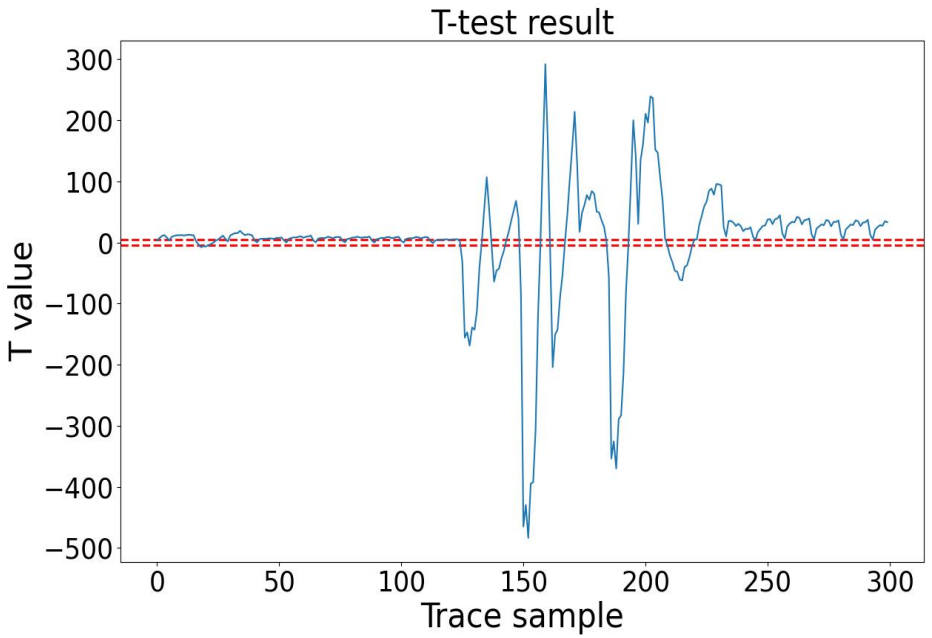
a sharing

$d + 1$ shares

Challenges in Software Implementation of Boolean Masking

- In terms of efficiency
  - Requires substantial random numbers, along with significant computational and storage overhead
- In terms of security
  - The low-noise environment of processors undermine Boolean masking protections
  - Presence of transitional leakage





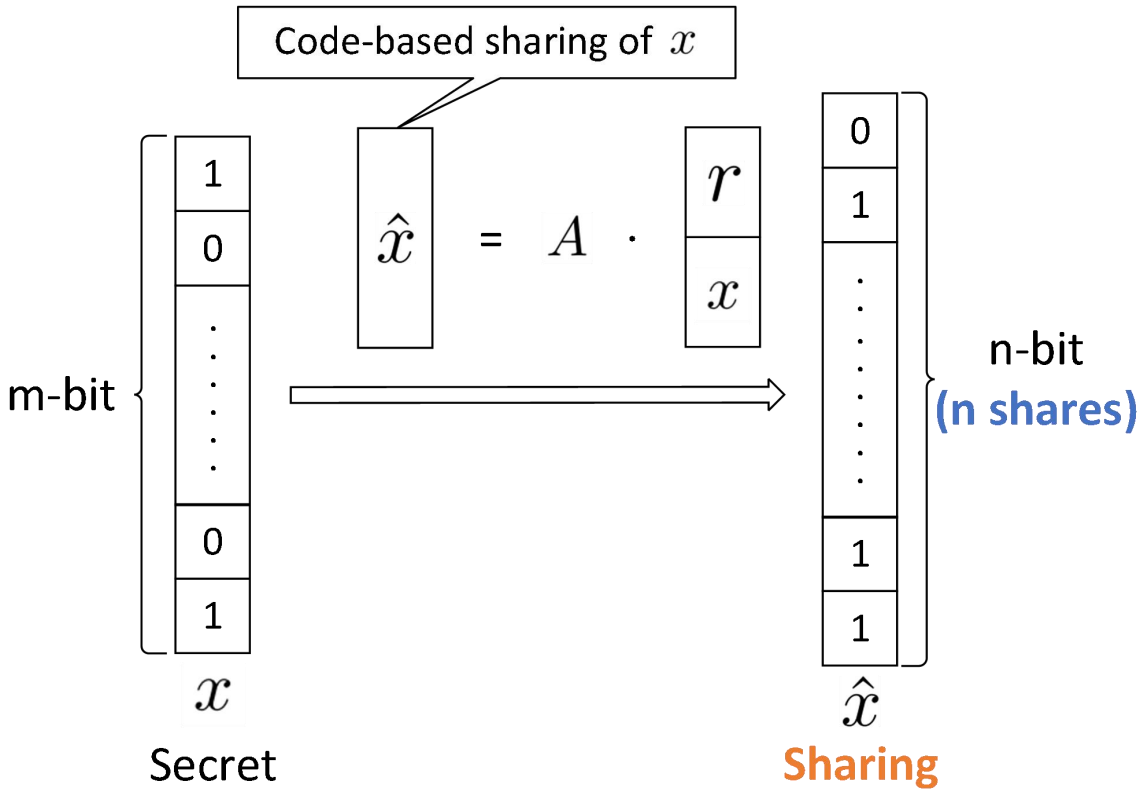
T-test analysis of first-order ISW multiplication on the Ibex

Outline

山东大学 SHANDONG UNIVERSITY

Code-based Masking

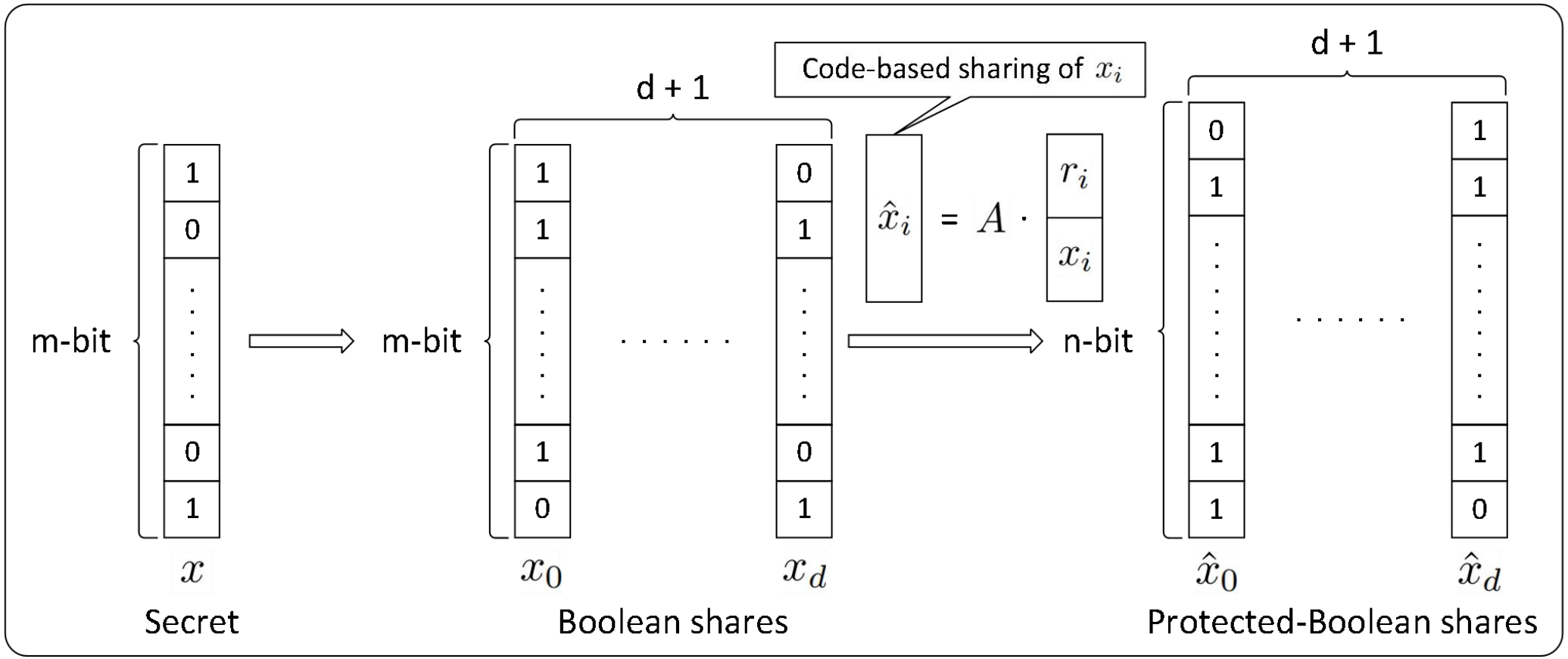
- The more complex algebraic structure of code-based masking makes it more robust to transitional leakage, and it can reduce information leakage under low-noise conditions



$$\hat{x} = Enc(x) = A \cdot (r \,||\, x)$$

$$x = Dnc(\hat{x}) = [A^{-1} \cdot \hat{x}]_m$$

Using code-based masking to protect Boolean masking shares



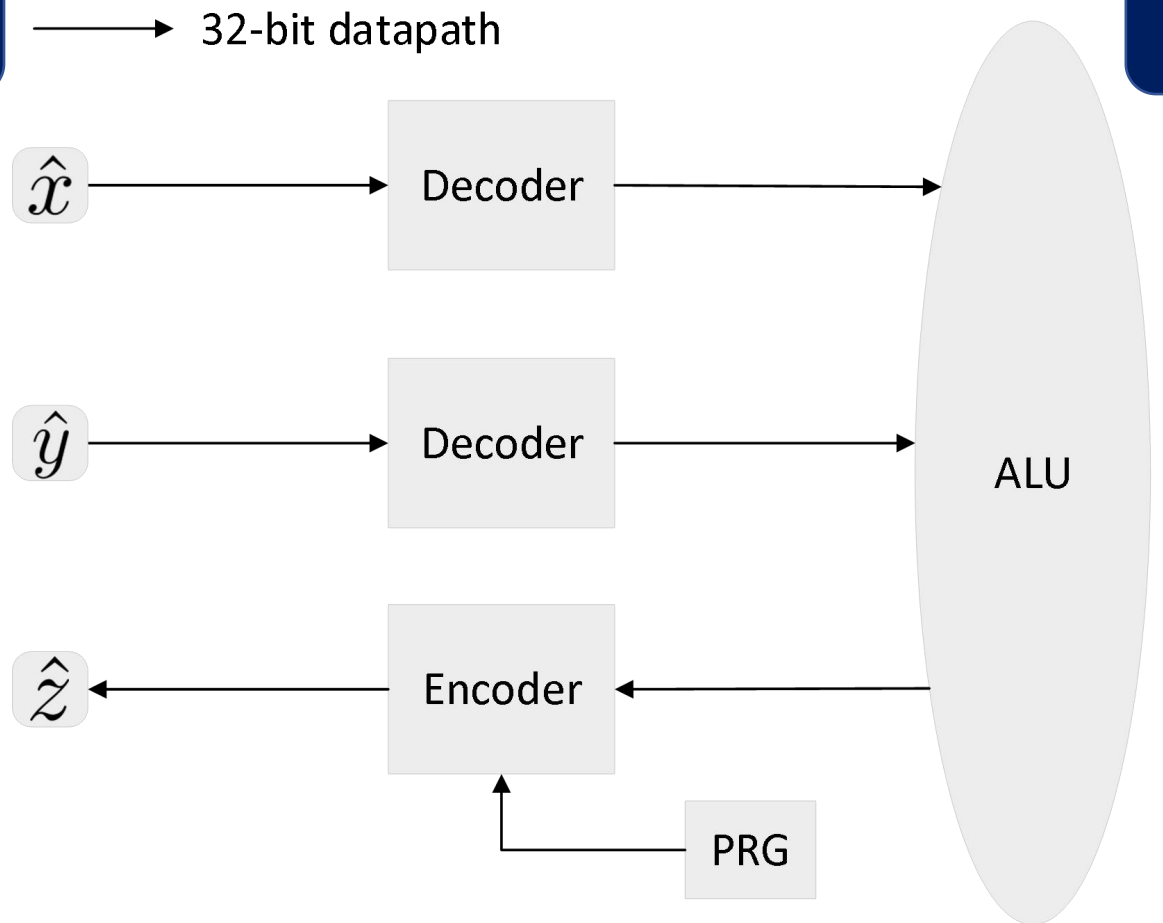
$$x = x_0 \oplus x_1 \oplus \ldots\ldots \oplus x_d = Dec(\hat{x}_0) \oplus Dec(\hat{x}_1) \oplus \ldots\ldots \oplus Dec(\hat{x}_d)$$

In our case, $\boldsymbol{m = 16, n = 32}$

Using code-based masking to protect Boolean masking shares

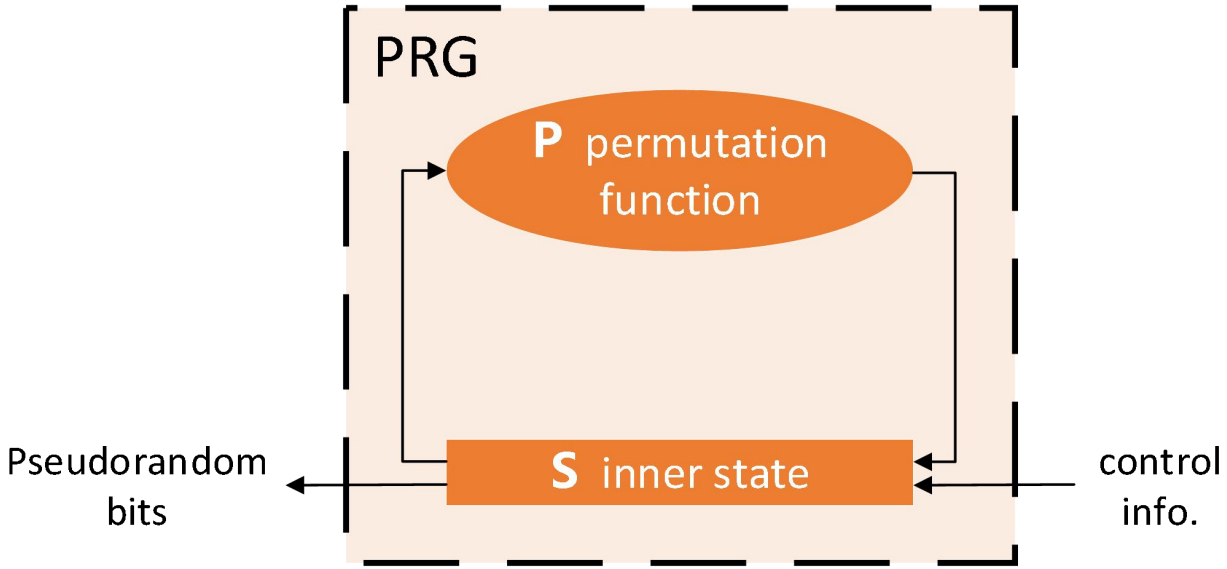
Preventing transitional leakage

Increasing Noise Level



Pseudorandom generator (PRG)

- Leakage-resilient PRG



- The permutation function P executes once per clock cycle
- Security in the continuous leakage model: even if attacker obtains multiple outputs and leakage information, the next execution can still generate r-bit random number

Instruction design

- “cbm.xxx”
- R-type and I-type
- Decoders are used by default
- es (inst[31:30])

es	Be encoded	Be refreshed
00	⊗	⊗
01	⊗	✓
10	✓	⊗
11	✓	✓

- “cbm.xor t0, t0, x0, 0”
- “xor” can replace “cbm.xor”

Computation instructions

	31 30	29 28 27 26 25	24 23 22 21 20	19 18 17	16 15 14	13 12 11 10 9 8 7 6 5 4 3 2 1	0
cbm.and rd, rs1, rs2, es :	es	00000	rs2	rs1	000	rd	00010	11
cbm.or rd, rs1, rs2, es :	es	00000	rs2	rs1	001	rd	00010	11
cbm.xor rd, rs1, rs2, es :	es	00000	rs2	rs1	010	rd	00010	11
cbm.sll rd, rs1, rs2, es :	es	00000	rs2	rs1	011	rd	00010	11
cbm.srl rd, rs1, rs2, es :	es	00000	rs2	rs1	100	rd	00010	11
cbm.andi rd, rs1, imm, es :	es	imm		rs1	000	rd	01010	11
cbm.ori rd, rs1, imm, es :	es	imm		rs1	001	rd	01010	11
cbm.xori rd, rs1, imm, es :	es	imm		rs1	010	rd	01010	11
cbm.slli rd, rs1, imm, es :	es	imm		rs1	011	rd	01010	11
cbm.srli rd, rs1, imm, es :	es	imm		rs1	100	rd	01010	11

Instruction design

PRG management instructions

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cbm.prg imm :	00000							imm					00000							00000						000				00000										10110										11		
cbm.s2r rd, imm :	00000							imm					00000							00000						001				rd										10110										11		
cbm.r2s rs1, imm :	00000							imm					00000							rs1						010				00000										10110										11		

- Keccak-p[100]
- “cbm.prg”: control the PRG state

imm	Operation
00	resetting PRG with the current seed
01	manually generating a new random number
10	disabling the automatic generation of random numbers
11	enabling the automatic generation of random numbers

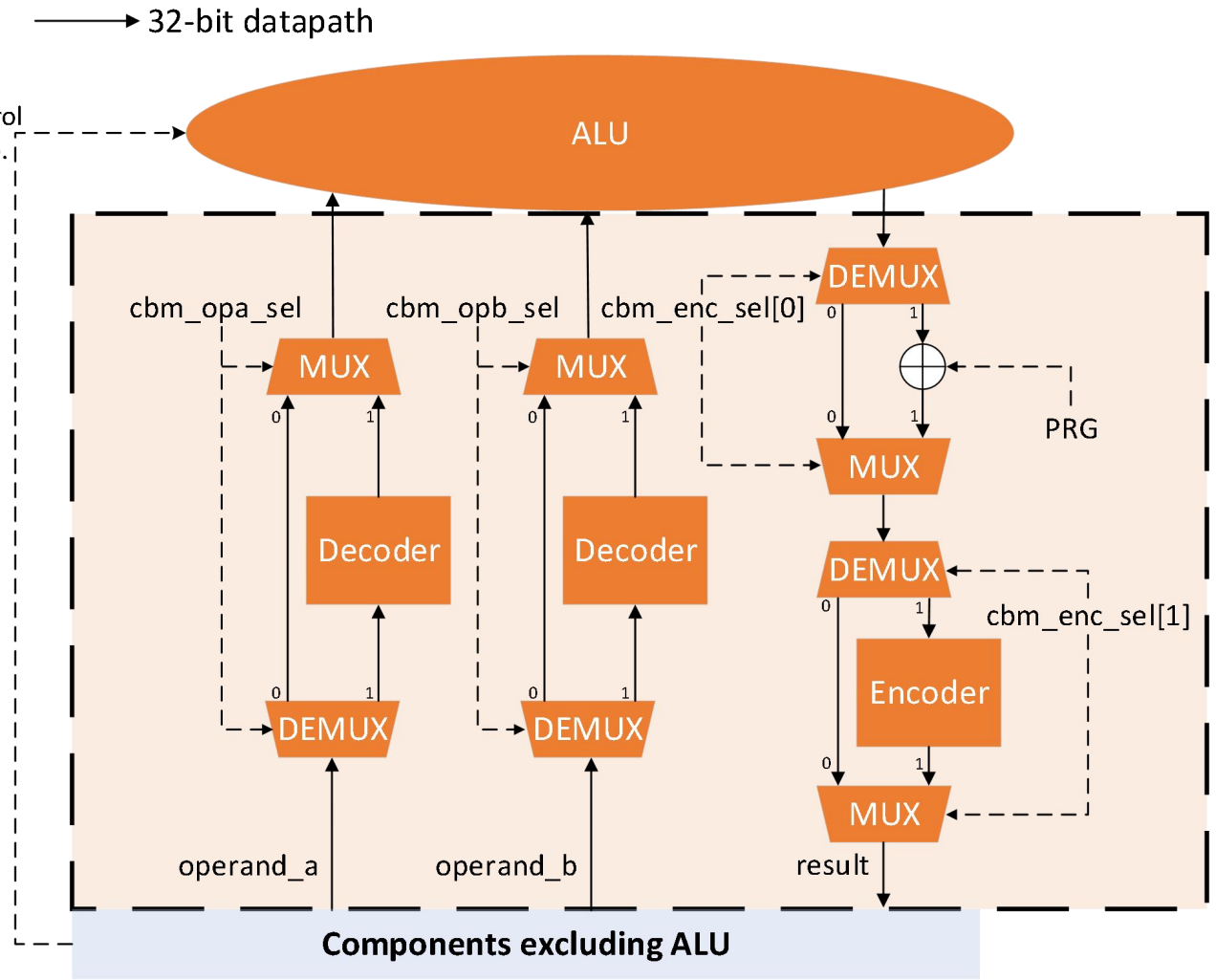
Security Proof

- **Requirement 1**
  - For each gate in the circuit connecting the general-purpose registers to the decoder, at most one input depends on the bits that appear in the general-purpose registers.
- **Requirement 2**
  - **a (Strict)**: Related sensitive data, such as shares of the same secret, etc. must not be accessed within two consecutive instructions
  - **b (Loose)**: Related sensitive data, such as shares of the same secret, etc. must not be accessed within two consecutive CBM instructions
- **Requirement 3**
  - The SNR of combinational circuits should be notably lower than that of registers and memory

Implementation details

- **Base core**: Ibex
- **Base SoC**: Ibex Demo System
- **Base ISA**: RV32IMC
- **PRG**: Keccak-p[100]

Signal	Depends on
cbm_opa_sel	Whether it is a cbm instruction
cbm_opb_sel	Whether it is a cbm instruction
cbm_enc_sel[0]	ex[0] (inst[30])
cbm_enc_sel[1]	ex[1] (inst[31])



Implementation details

- Decoder/Encoder Matrix

$$A = A^{-1} = \begin{pmatrix} 2 & 3 & 4 & 12 & 5 & 10 & 8 & 15 \\ 3 & 2 & 12 & 4 & 10 & 5 & 15 & 8 \\ 4 & 12 & 2 & 3 & 8 & 15 & 5 & 10 \\ 12 & 4 & 3 & 2 & 15 & 8 & 10 & 5 \\ 5 & 10 & 8 & 15 & 2 & 3 & 4 & 12 \\ 10 & 5 & 15 & 8 & 3 & 2 & 12 & 4 \\ 8 & 15 & 5 & 10 & 4 & 12 & 2 & 3 \\ 15 & 8 & 10 & 5 & 12 & 4 & 3 & 2 \end{pmatrix}$$

- The implementation of matrix multiplication
  - This can be regarded as the Shortest Linear Program (SLP)
  - Area-optimized version: 182 XORs, 289.85MHz
  - Frequency-optimized version: 251 XORs, 340.64MHz
- Register Gating
  - It can address the problems of wire switching in the multiplexer tree, unintended reads, and glitchy signals by placing an AND gate after each register

Outline

1 Background

2 Design and Implementation

3 **Evaluation**

4 Future works

山东大学 SHANDONG UNIVERSITY

Hardware overhead and Frequency

Core	FPGA		ASIC
	Registers	LUTs	GE
Base core	2362 (1.00×)	3951 (1.00×)	30627 (1.00×)
Base core + #1	2530 (1.07×)	4795 (1.21×)	33057 (1.08×)
Base core + #2	2530 (1.07×)	4865 (1.23×)	33772 (1.10×)

Core	**FPGA Freq.** [MHz]	**ASIC Freq.** [MHz]
Base core	72.03 (1.00×)	347.22 (1.00×)
Base core + #1	62.43 (0.87×)	289.85 (0.83×)
Base core + #2	64.11 (0.89×)	340.64 (0.98×)

#1 CBM (area-optimized version)
#2 CBM (frequency-optimized version)

Formal verification

- **Coco** as the verification tool
- Verify **first-order ISW multiplication**
- Hardware fulfills **Requirement 1** and **3**
  - **Requirement 1:** at most one input depends on the bits in the register
  - **Requirement 3:** the SNR of combinational circuits should be lower than that of registers and memory
- Software fulfills **Requirement 2.a**
  - **Requirement 2.a:** related sensitive data must not be accessed within two consecutive instructions

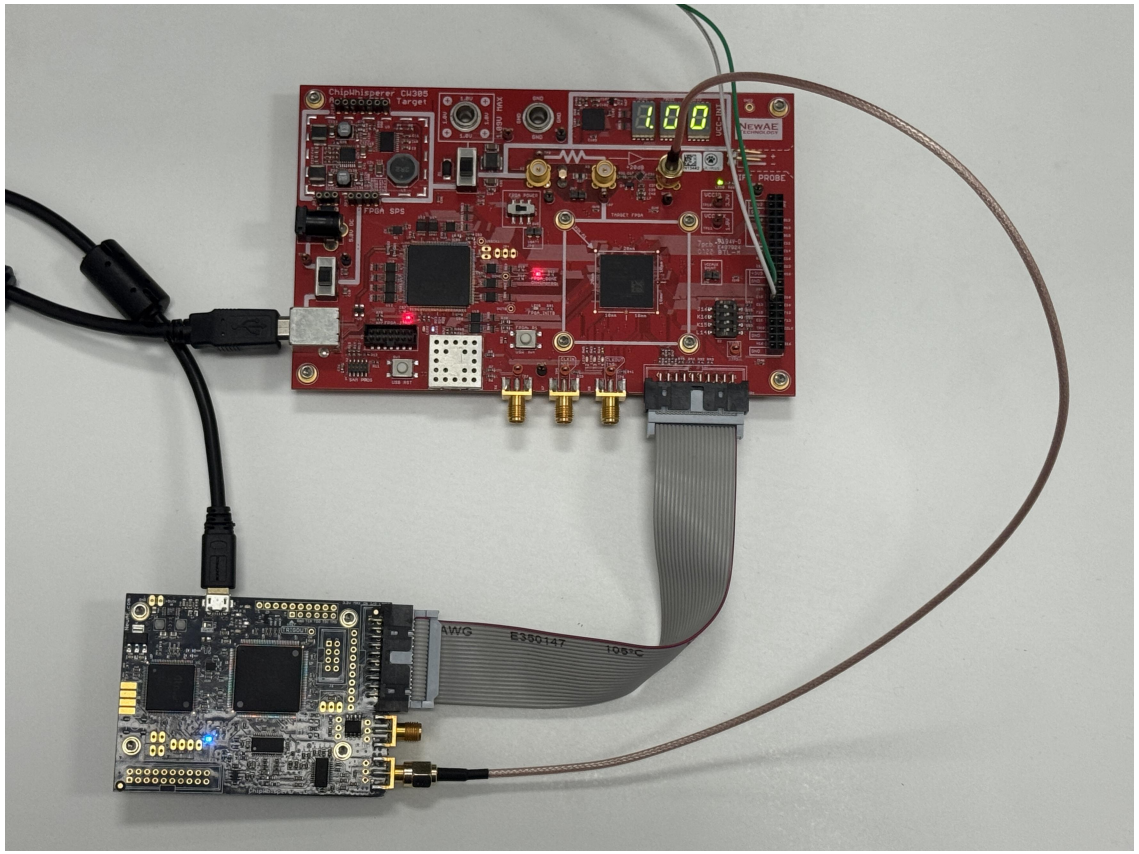
RV32I: Leakage captured

CBM: No leakage

山东大学 SHANDONG UNIVERSITY

Security

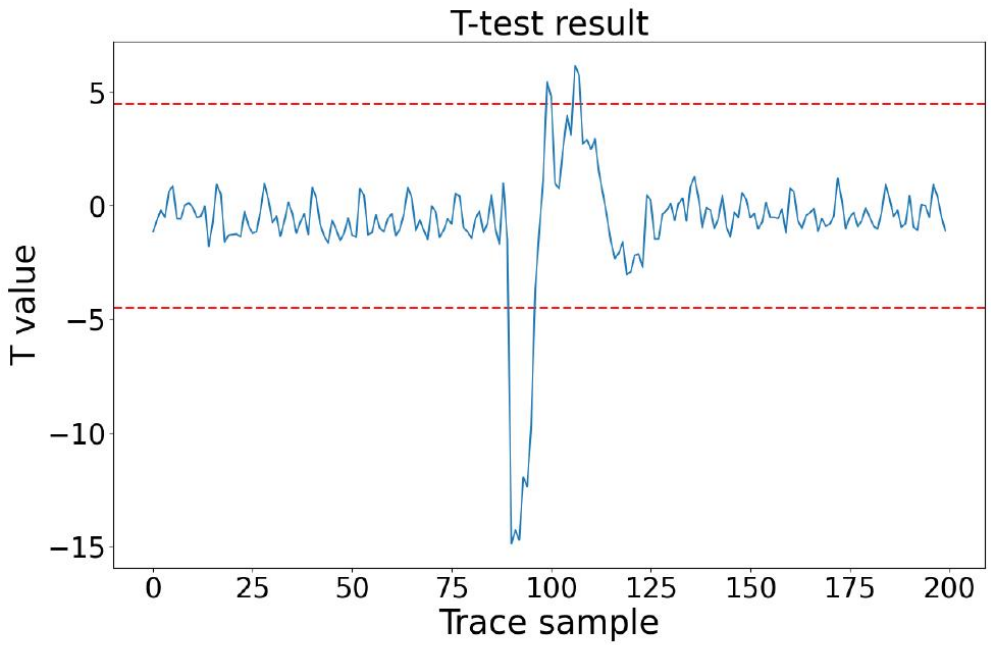
- **Target board**:
  ChipWhisperer CW305
- **Capture board**:
  ChipWhisperer-Lite (CW1173)
- **Core frequency**: 8 MHz
- **Power amplification**:
  CW305 20dB gain
  CW-Lite 20dB gain
- **Uart Tx**: E13 Pin
- **Uart Rx**: E15 Pin
- Software fulfills **Requirement 2.b**



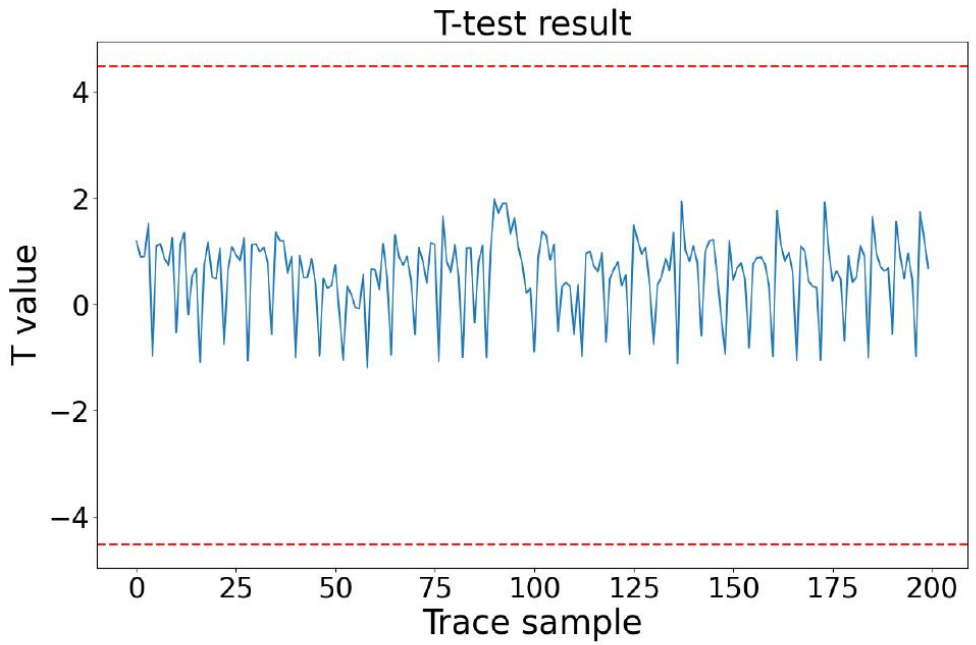
Requirement 2.b: related sensitive data must not be accessed within two consecutive CBM instructions

Empritical tests regarding overwriting

- Use “mv” instruction to overwrite $\hat{y}$ with $\hat{x}$
- Number of traces used for t-test analysis: 10 million



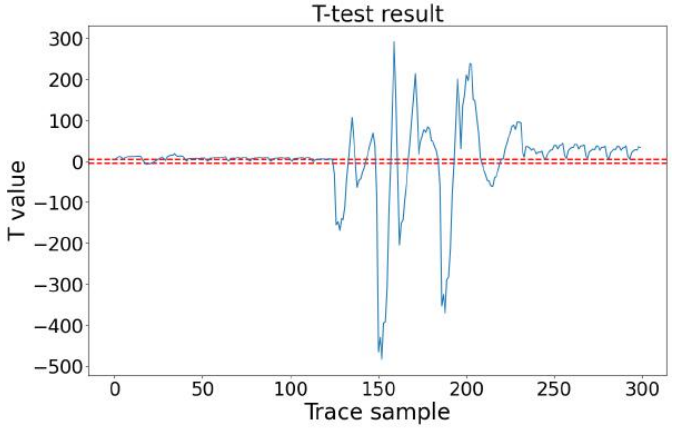
(a) Base core: RV32I instructions



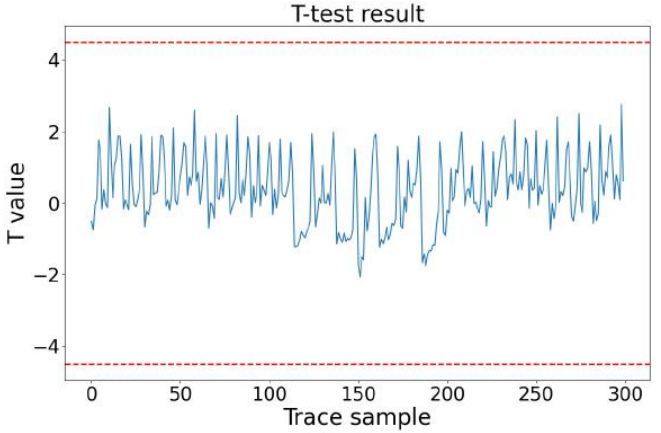
(b) ISE-extended core: CBM instructions

Empritical tests regarding first-order ISW multiplication

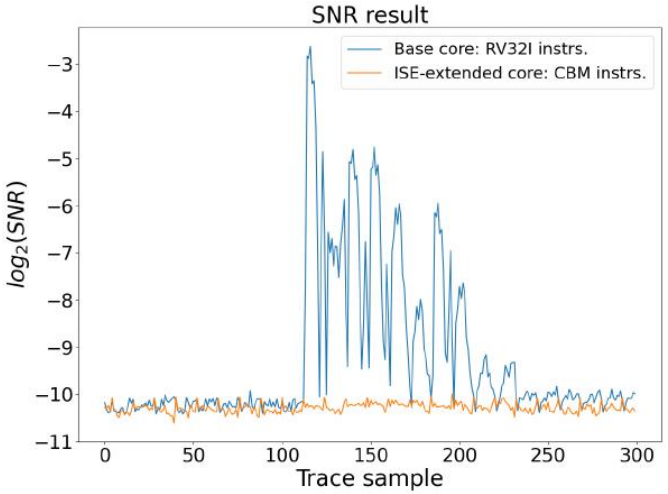
- Number of traces used for t-test analysis: 10 million
- Number of traces used for SNR and MI analysis on a share: 300 thousand



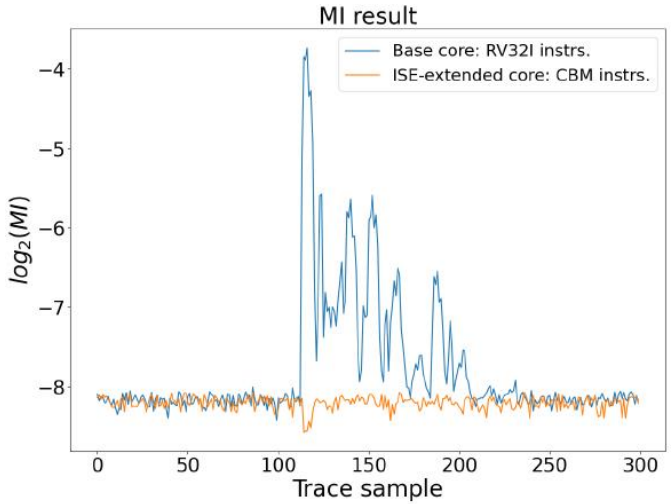
(a) Base core: RV32I instructions



(b) ISE-extended core: CBM instructions



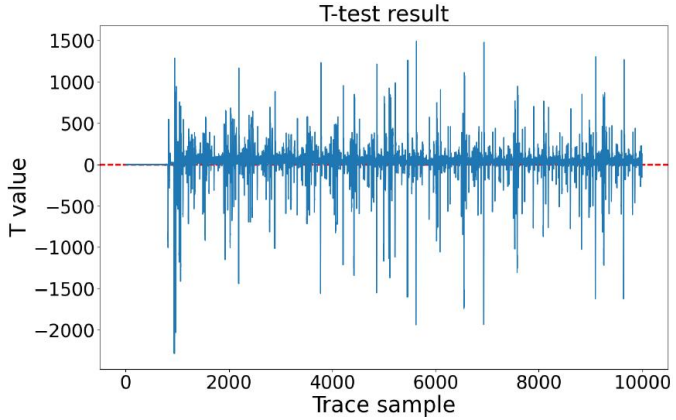
(c) SNR



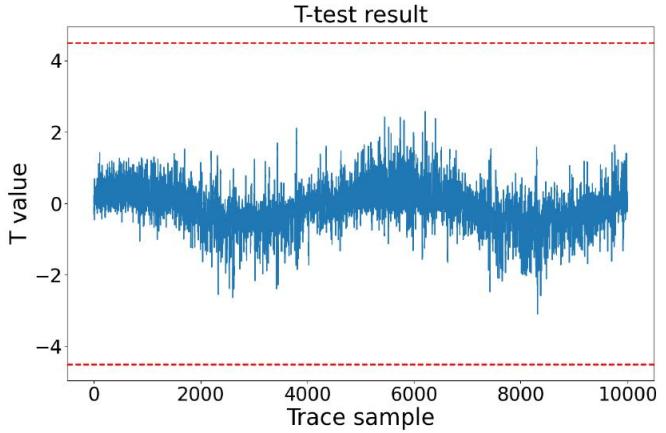
(d) MI

Empritical tests regarding first-order bit-sliced AES S-Box

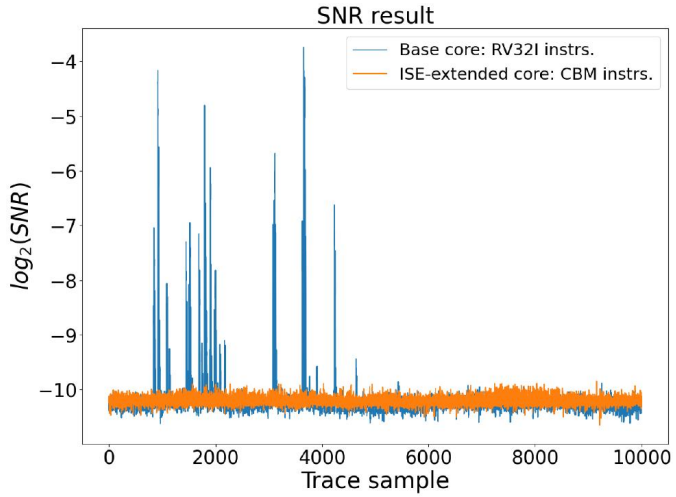
- Number of traces used for t-test analysis: 10 million
- Number of traces used for SNR and MI analysis on a share: 300 thousand



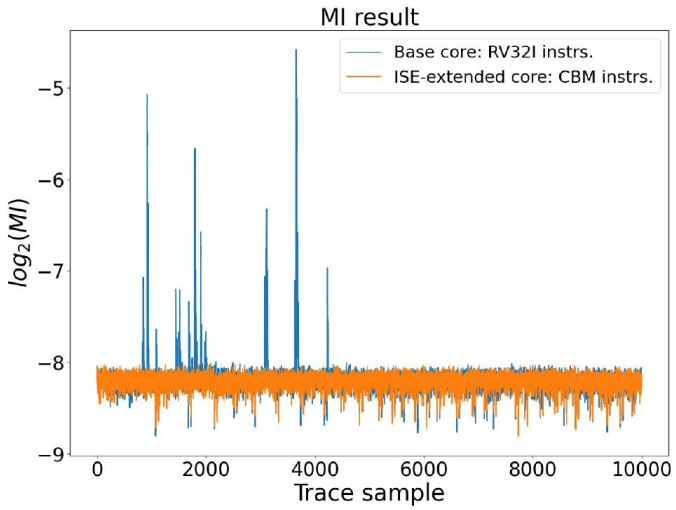
(a) Base core: RV32I instructions



(b) ISE-extended core: CBM instructions



(c) SNR



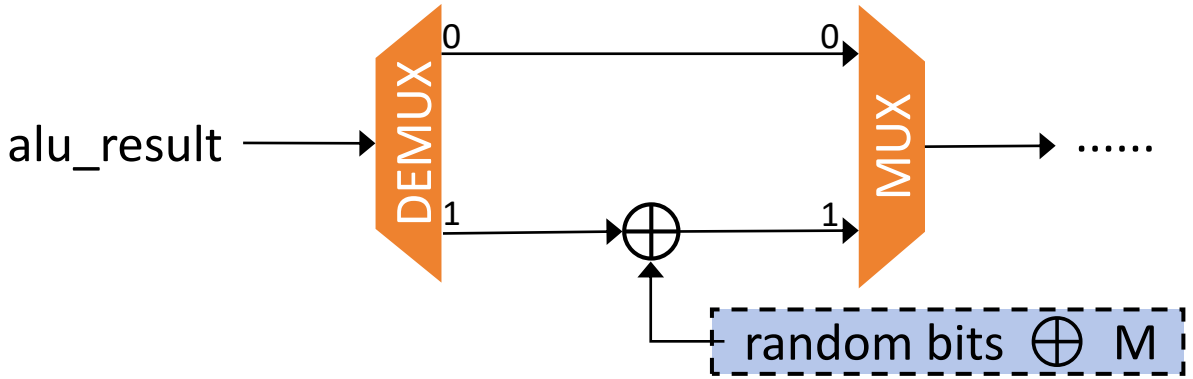
(d) MI

Outline

山东大学 SHANDONG UNIVERSITY

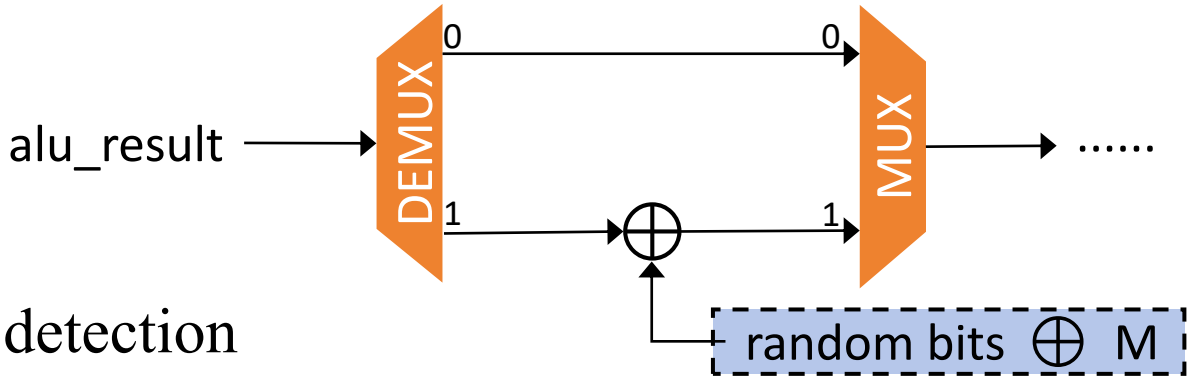
Future works

- Increase computational capacity
  - Use a mask (M) stored in a CSR to dynamically control the computational capacity

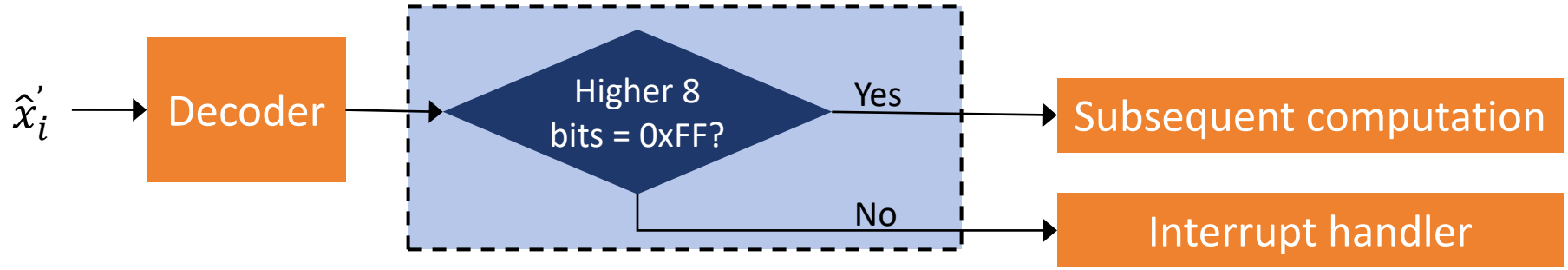


Future works

- Increase computational capacity
  - Use a mask (M) stored in a CSR to dynamically control the computational capacity

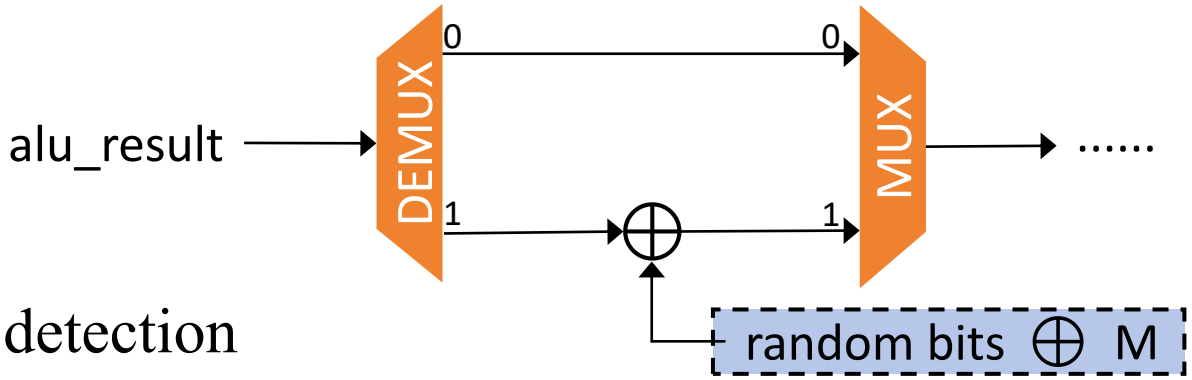


- Introduce fault detection
  - Given a Boolean share x_i, its protected-Boolean share is $\hat{x}_i^{'} = A \cdot (1^8 \,||\, r^{'} \,||\, x_i)$

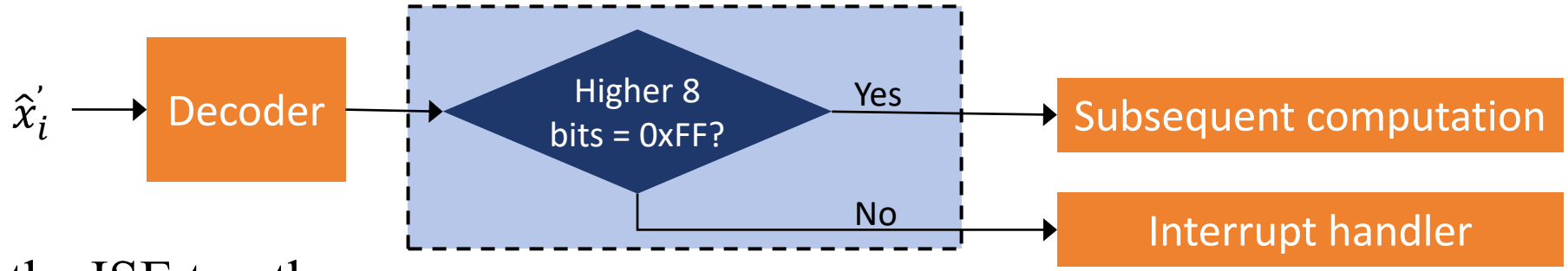


Future works

- Increase computational capacity
  - Use a mask (M) stored in a CSR to dynamically control the computational capacity



- Introduce fault detection
  - Given a Boolean share x_i, its protected-Boolean share is $\hat{x}_i^{'} = A \cdot (1^8 \,||\, r^{'} \,||\, x_i)$



- Apply the ISE to other areas
  - Apply this ISE to the masked implementations of some algorithms, such as Kyber and Dilithium

Thank you for your attention!