MINES Saint-Étienne

Une école de l'IMT

Anthony ZGHEIB

A CCFI verification scheme based on the RISC-V Trace Encoder

14th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)

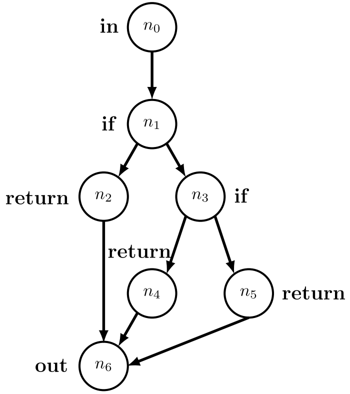
April 3rd, 2023

INSPIRING INNOVATION SINCE 1816

What is the execution flow of a program?

- It follows a specific path in its Control Flow Graph (CFG).
  - The CFG is a graph that shows all the **legitimate** paths of a program.
- Example:

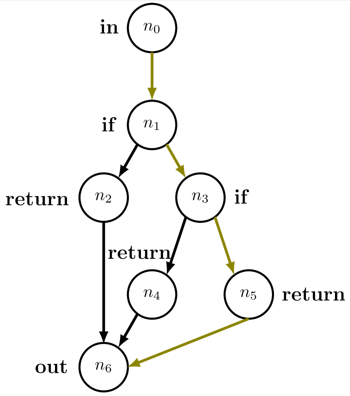
```
int isabsequal (int x, int y)
{
    if(x == y)
        return 1;
    else if (x == -y)
        return -1;
    else
        return 0;
    end if;
}
```



Why is it necessary to guarantee the execution flow?

- x is different than y

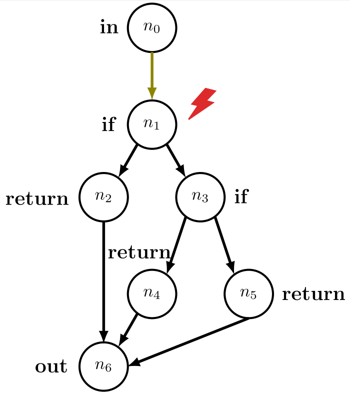
```
int isabsequal (int x, int y)
{
    if(x == y)
      return 1;
    else if (x == -y)
      return -1;
    else
      return 0;
    end if;
}
```



Why is it necessary to guarantee the execution flow?

- x is different than y

```
int isabsequal (int x, int y)
{
    if(x == y)
        return 1;
    else if (x == -y)
        return -1;
    else
        return 0;
    end if;
}
```

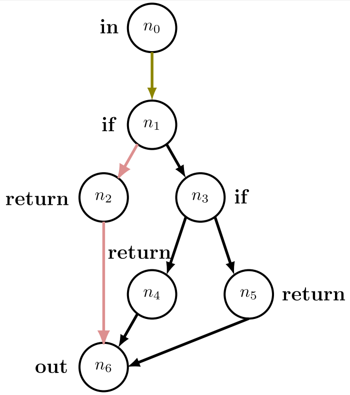


- Hardware attacks: Fault Injection Attacks (FIA) [2].

Why is it necessary to guarantee the execution flow?

- x is different than y

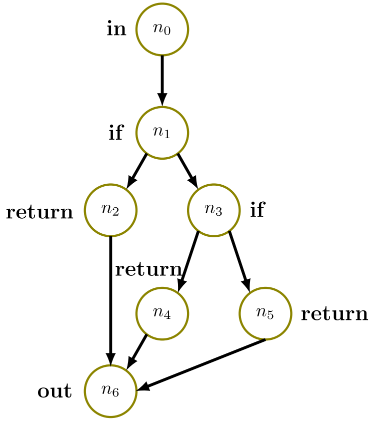
```
int isabsequal (int x, int y)
{
    if(x == y)
        return 1;
    else if (x == -y)
        return -1;
    else
        return 0;
    end if;
}
```



- Hardware attacks: Fault Injection Attacks (FIA) [2].

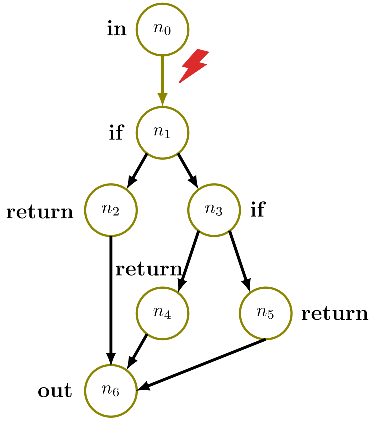
Control Flow Integrity (CFI) approach

- It checks a program execution flow and detects if it is correctly executed and not altered by software or physical attacks.



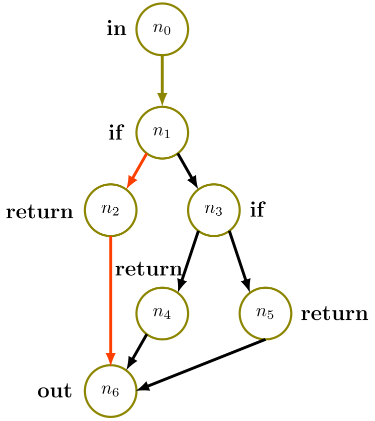
CFI Limitations

- FIA on instructions between two discontinuity ones.
- E.g. by corrupting the reading addresses of x or y values.



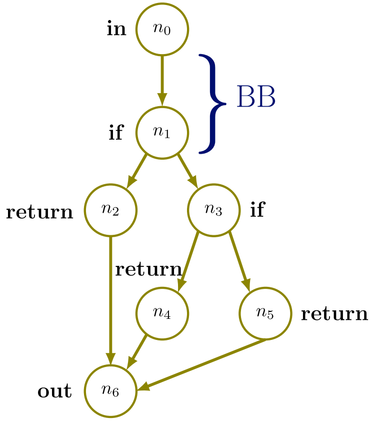
CFI Limitations

- FIA on instructions between two discontinuity ones.
- E.g. by corrupting the reading addresses of x or y values.



Code and Control Flow Integrity (CCFI) approach

- In addition to CFI features, CCFI detects FIA on Basic Blocks (BB)[1].
- It checks the integrity of **all** executed instructions.



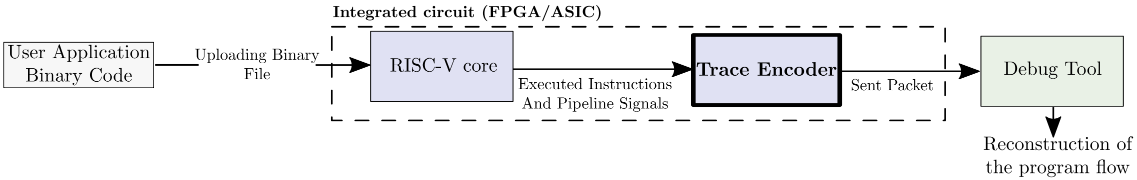
[1] BB is a set of successive instructions where their execution is done consecutively and in order.

Table of contents

A new CCFI scheme based on the Trace Encoder (TE)

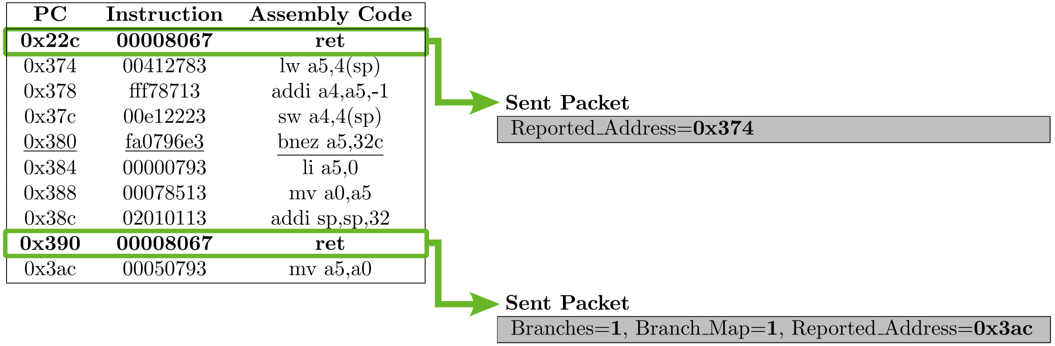
TE Overview

- Embedded debug module designed by RISC-V foundation [7].
- Used by developers for debug purposes.
- It compresses, at runtime, the sequence of discontinuities executed by the RISC-V core into trace packets.


Integrated circuit (FPGA/ASIC)
User Application Binary Code
Uploading Binary File
RISC-V core
Executed Instructions And Pipeline Signals
Trace Encoder
Sent Packet
Debug Tool
Reconstruction of the program flow

Functionality - TE - Example

- A packet is sent when an unpredictable discontinuity (target address is not known from the binary code) is executed, *e.g:* a return instruction.



PC	Instruction	Assembly Code
0x22c	**00008067**	**ret**
0x374	00412783	lw a5,4(sp)
0x378	fff78713	addi a4,a5,-1
0x37c	00e12223	sw a4,4(sp)
0x380	fa0796e3	bnez a5,32c
0x384	00000793	li a5,0
0x388	00078513	mv a0,a5
0x38c	02010113	addi sp,sp,32
0x390	**00008067**	**ret**
0x3ac	00050793	mv a5,a0

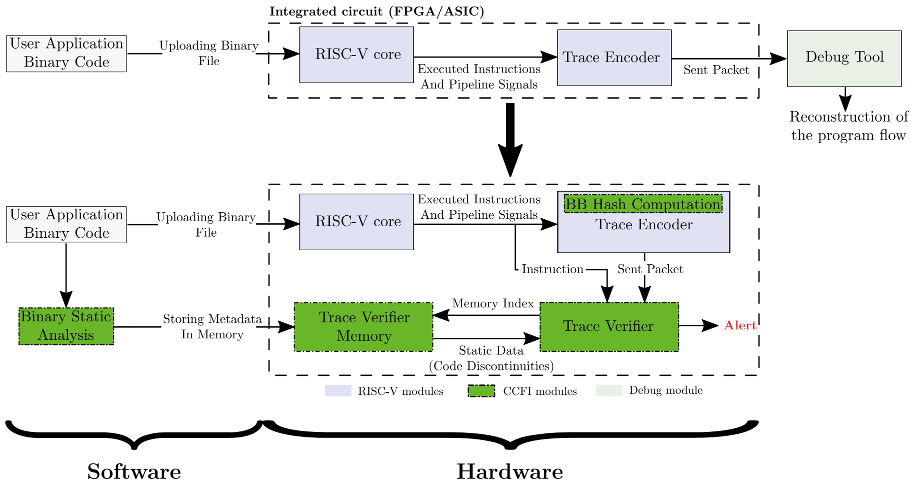
Sent Packet

Reported_Address=**0x374**

Sent Packet

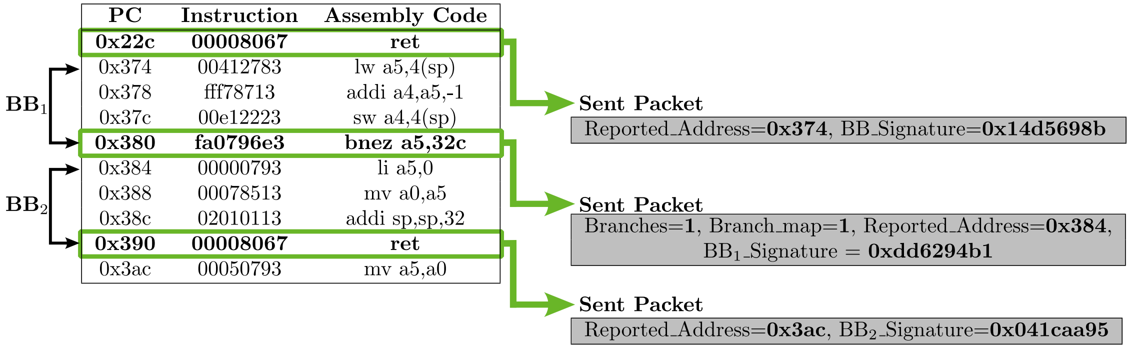
Branches=**1**, Branch_Map=**1**, Reported_Address=**0x3ac**

A co-design CCFI verification system based on the TE



Enhancement of the TE

- A packet is sent after **each** discontinuity, not just after unpredictable instructions.
- A BB hash computation module based on a MISR [6] is integrated.



PC	Instruction	Assembly Code
0x22c	**00008067**	**ret**
0x374	00412783	lw a5,4(sp)
0x378	fff78713	addi a4,a5,-1
0x37c	00e12223	sw a4,4(sp)
0x380	**fa0796e3**	**bnez a5,32c**
0x384	00000793	li a5,0
0x388	00078513	mv a0,a5
0x38c	02010113	addi sp,sp,32
0x390	**00008067**	**ret**
0x3ac	00050793	mv a5,a0

Sent Packet
Reported_Address=**0x374**, BB_Signature=**0x14d5698b**

Sent Packet
Branches=**1**, Branch_map=**1**, Reported_Address=**0x384**,
BB_1_Signature = **0xdd6294b1**

Sent Packet
Reported_Address=**0x3ac**, BB_2_Signature=**0x041caa95**

TE-based CCFI solution features

Solution characteristics

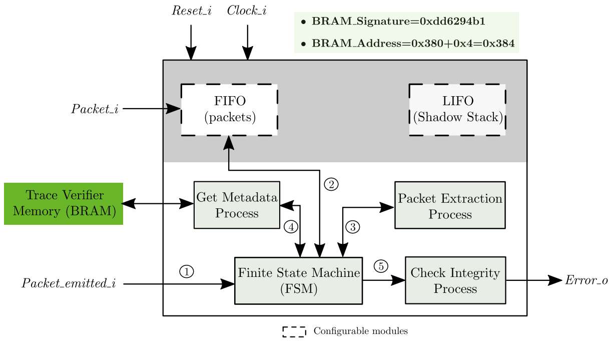
- Verification process starts when a packet is sent.
- Navigation through static data and constitution of the program's followed path.

PACKET CONTENT

- Branches =1
- Branch_map=1
- Reported_Address=0x384
- BB_1_Signature= 0xdd6294b1

RAM

Index	:	Content				
3B	:	$\underbrace{00000328}_{\text{PC}}$	$\underbrace{04c0006f}_{\text{Instruction}}$	$\underbrace{003E}_{\text{J Index}}$	$\underbrace{\boxed{dd6294b1}}_{\text{Hash signature Basic Block 1}}$	$\underbrace{00000000}_{\text{Unused}}$
3E	:	$\underbrace{\boxed{00000380}}_{\text{PC}}$	$\underbrace{fa0796e3}_{\text{Instruction}}$	$\underbrace{003C}_{\text{Index if Br Taken}}$	$\underbrace{18d05141}_{\text{Hash signature if Br Taken}}$	$\underbrace{041caa95}_{\text{Hash signature Basic Block 2}}$


- BRAM_Signature=0xdd6294b1
- BRAM_Address=0x380+0x4=0x384

Example - Memcmp Function

- It compares the values of two arrays.

```
int memcmp(const void *src1, const void *src2,
 uint32_t n) {
unsigned char *s1 = (unsigned char *) src1;
unsigned char *s2 = (unsigned char *) src2;

while (n--) {
if (*s1 != *s2) {
return *s1 - *s2; }
s1++;
s2++; }
return 0; }
```

BB_1

PC	Instruction	Assembly Code
0x32c	01c12783	lw a5, 28(sp)
...	...	...
0x374	00412783	lw a5,4(sp)
0x378	fff78713	addi a4,a5,-1
0x37c	00e12223	sw a4,4(sp)
0x380	**fa0796e3**	**bnez a5,32c**
0x384	00000793	li a5,0

Sent Packet

Branches=**1**, Branch_map=**0**, Reported_Address=**0x32c**,
BB_1_Signature = **0xdd6294b1**

Example - Memcmp Function

- Simulation of a FIA (**4 bitflips**) on a lw instruction.

```
int memcmp(const void *src1, const void *src2,
 uint32_t n) {
unsigned char *s1 = (unsigned char *) src1;
unsigned char *s2 = (unsigned char *) src2;

while (n--) {
if (*s1 != *s2) {
return *s1 - *s2; }
s1++;
s2++; }
return 0; }
```

BB$_1$

PC	Instruction	Assembly Code
0x32c	01c12783	lw a5, 28(sp)
...	...	...
0x374	00000793	li a5,0
0x378	fff78713	addi a4,a5,-1
0x37c	00e12223	sw a4,4(sp)
0x380	**fa0796e3**	**bnez a5,32c**
0x384	00000793	li a5,0

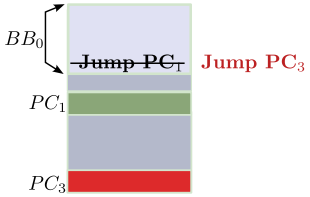
Sent Packet

Branches=**1**, Branch_map=**1**, Reported_Address=**0x384**, BB$_1$_Signature = **0xdf6b9431**

- FIA detection (0xdf6b9431 $\neq$ 0xdd6294b1)

Table of contents

Covered threats

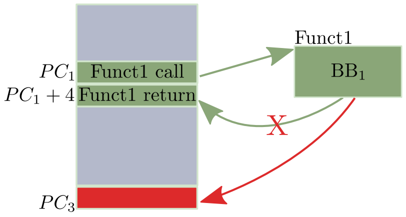


- **Packet Content Without FIA**
  - Reported_address: PC_1
  - Reported_signature: signature of BB$_0$
- **Packet Content With FIA**
  - Reported_address: **PC$_3$**
  - Reported_signature: signature of BB$'_0$

Covered threats

- Corruption of **any** discontinuity instruction.
- Faults on **any** instruction of the BB (e.g. changing the return address of a call).

Covered threats

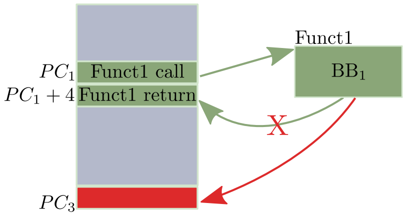


- **Packet Content Without FIA**
  - Reported_address: $PC_1 + 4$
  - Reported_signature: signature of BB$_1$
- **Packet Content With FIA**
  - Reported_address: $\mathbf{PC_3}$
  - Reported_signature: signature of BB$'_1$

Covered threats

- Corruption of **any** discontinuity instruction.
- Faults on **any** instruction of the BB (e.g. changing the return address of a call).

Covered threats



- **Packet Content Without FIA**
  - Reported_address: $PC_1 + 4$
  - Reported_signature: signature of BB$_1$
- **Packet Content With FIA**
  - Reported_address: $\mathbf{PC_3}$
  - Reported_signature: signature of BB$_1'$

Covered threats

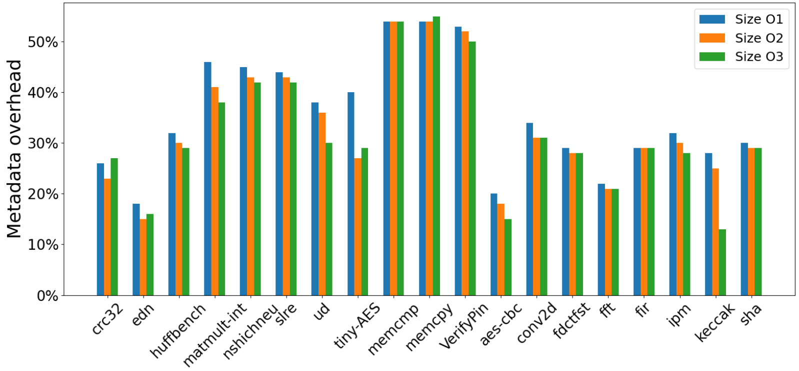
- Corruption of **any** discontinuity instruction.
- Faults on **any** instruction of the BB (e.g. changing the return address of a call).

Limitation

- FIA on core's pipeline signals (e.g. ALU_OP, Multiplexers, etc).

Table of contents

Memory overhead - Benchmarks



- Metadata size depends of the application.
- For simulated benchmarks, metadata-code size ratio ranges from 15% to 55%.

Hardware overhead - FPGA implementation

- Simulated RISC-V core: IBEX (645 slices*)
  - ISA extension: RV32IM
- Trace Encoder: 239 slices +
  62 slices (enhancement & MISR)
- Trace Verifier: 170 slices (core) +
  15 slices (FIFO/LIFO)
  = **185 slices**

Nexys Artix-7 board [1]

- The TV represents **27,9%** in terms of slices with respect to the TE + IBEX.

*Slice : four 6-input LUTs, 8 flip-flops, multiplexers and carry units.

Table of contents

Conclusion

- Verification of programs CCFI running on the IBEX core with a RISC-V ISA RV32IM+**C** extension.
  - Enhancing the TE standard.
  - Adding a MISR computation module to the TE.
- Detection of FIA on **any** instruction till the IBEX Decode Stage.

Conclusion

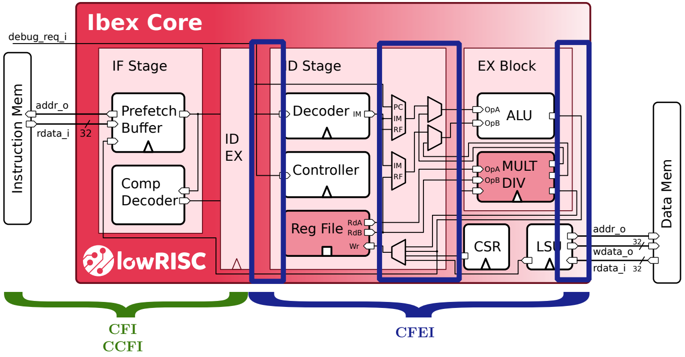
Solution	SOFIA [5]	SCFP [8]	SCI-FI [3]	CCFI-Cache [4]	ATRIUM [9]	TE-CFI [10]	**This Work**
No User Code Modification	✗	✗	✗	✗	✓	✓	✔
No Compiler Modification	✓	✗	✗	✗	✓	✓	✔
No Pipeline Modification	✗	✗	✗	✓	✓	✓	✔
No Performance Overhead	✗	✗	✗	✗	✗	✓	✔
Backward Edge Protection	✓	✓	✓	✓	✓	✓	✔
Forward Edge Protection	✗	✓	✗	(✗)	✗	✗	✘
Code Integrity	✓	✓	✓	✓	✓	✗	✔
Code Confidentiality	✓	✓	✗	✗	✗	✗	✘

Conclusion

Solution	SOFIA [5]	SCFP [8]	SCI-FI [3]	CCFI-Cache [4]	ATRIUM [9]	TE-CFI[10]	**This Work**
Code Size (%)	141	19.8	25.4	<30	0	0	**0**
Performance (%)	110	9.1	17.5	32	<22.7	0	**0**
Hardware Area (%)	28.2	N/A	<23.8	10	<20	17	**27.9**
TV BRAM Size (%)	0	0	0	0	0	4.29	**6.25**

Perspectives

- Verify program's CCFI on the IBEX with the branch prediction feature enabled.
- Check that instructions are unaltered within the core's pipeline.
    - Control Flow and Execution Integrity (CFEI) verification.



MINES Saint-Étienne

Une école de l'IMT

Thank you for your attention!

This research is part of the Projet COFFI: ANR-18-CES39-003

INSPIRING INNOVATION SINCE 1816

References I

[1] Nexys Video Artix. “FPGA: Trainer Board for Multimedia Applications”. In: *URL: https://store. digilentinc. com/nexys-video-artix-7-fpga-trainer-board-for-multimedia-applications.(Accessed on: May 20, 2021)* (7).

[2] Alessandro Barenghi et al. “Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures”. In: *Proc. IEEE* 100.11 (2012), pp. 3056–3076.

[3] T. Chamelot, D. Couroussé, and K. Heydemann. “SCI-FI: control signal code and control flow integrity against fault injection attacks”. In: *2022 Design, Automation & Test in Europe Conference & Exhibition*. IEEE. Aug. 2022, pp. 556–559.

References II

[4] J. L. Danger et al. “CCFI-cache: A transparent and flexible hardware protection for code and control-flow integrity”. In: *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE. 2018, pp. 529–536.

[5] R. De Clercq et al. “SOFIA: software and control flow integrity architecture”. In: *Computers & Security* 68 (2017), pp. 16–35.

[6] Fayez Elguibaly and M Watheq El-Kharashi. “Multiple-input signature registers: an improved design”. In: *1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM. 10 Years Networking the Pacific Rim, 1987-1997*. Vol. 2. IEEE. 1997, pp. 519–522.

[7] RISC-V International. *Efficient Trace for RISC-V*. Nov. 2020. URL: https://github.com/riscv/riscv-trace-spec.

References III

[8] M. Werner et al. "Sponge-based control-flow protection for iot devices". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 214–226.

[9] S. Zeitouni et al. "Atrium: Runtime attestation resilient under memory attacks". In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2017, pp. 384–391.

[10] A. Zgheib et al. "A CFI Verification System based on the RISC-V Instruction Trace Encoder". In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE. 2022.