November 20th, 2025

Code Encryption for Confidentiality and Execution Integrity down to Control Signals

Olivier Potin / Mines Saint-Etienne
BITFLIP – Protection schemes against faults

MINES Saint-Étienne
Une école de l'IMT

Context for execution integrity

Context for execution integrity
Threat model

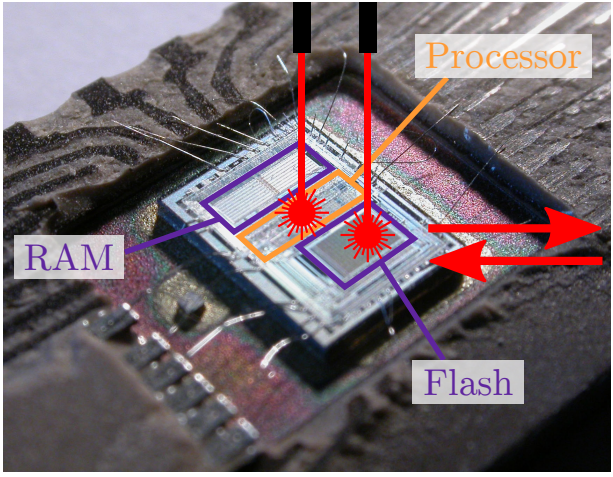
Threat model

- Read instructions in memory
- FIA on instruction memory
- FIA on processor control logic

Use-cases

- ➜ Identify vulnerabilities
- ➜ Bypass PIN, hash, ... verification

FIA: Fault Injection Attacks



Microcontroller

Context for execution integrity
Security properties

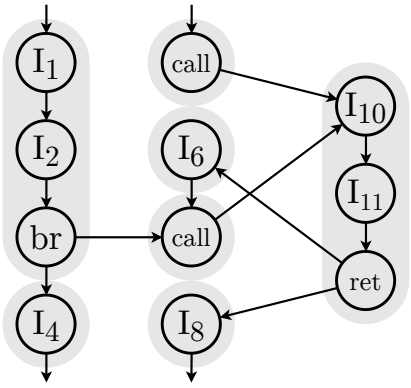
Integrity of:

- ➜ Control Flow (CFI)
- ➜ Instructions
- ➜ Control Signals
- ➜ ~~Data~~

Context for execution integrity

Control Flow and Instruction Integrity

CFG

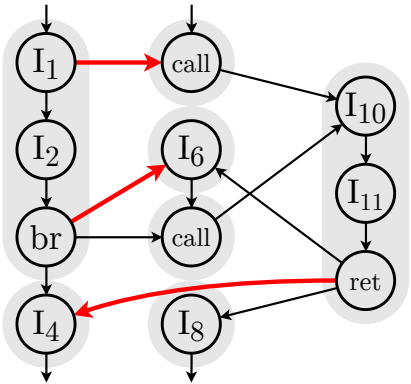


Control Flow Graph

Context for execution integrity

Control Flow and Instruction Integrity

CFG



Control Flow Graph

Control Flow Integrity

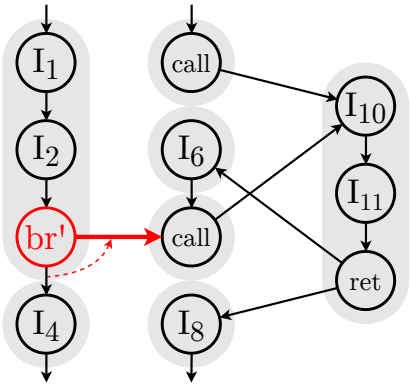
Tamper with:

- PC
- Data
- Microarchiteture
- Instruction

Context for execution integrity

Control Flow and Instruction Integrity

CFG



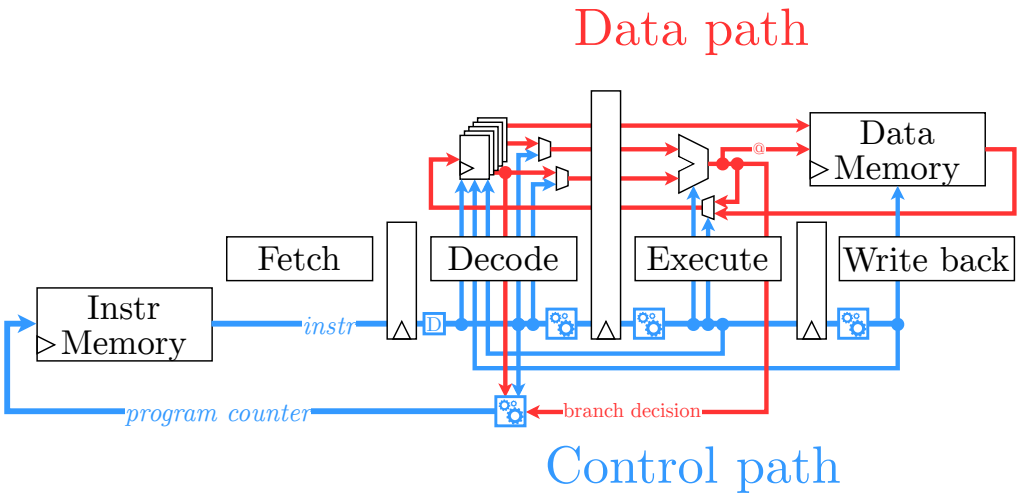
Control Flow Graph

Control Flow Integrity

Instruction Integrity

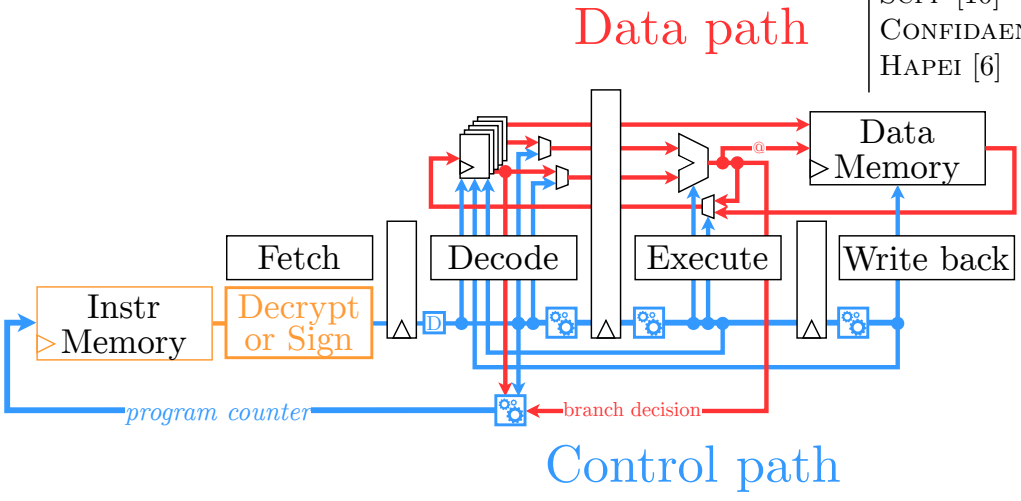
Context for execution integrity

Program execution in a pipeline



Context for execution integrity
Program execution in a pipeline

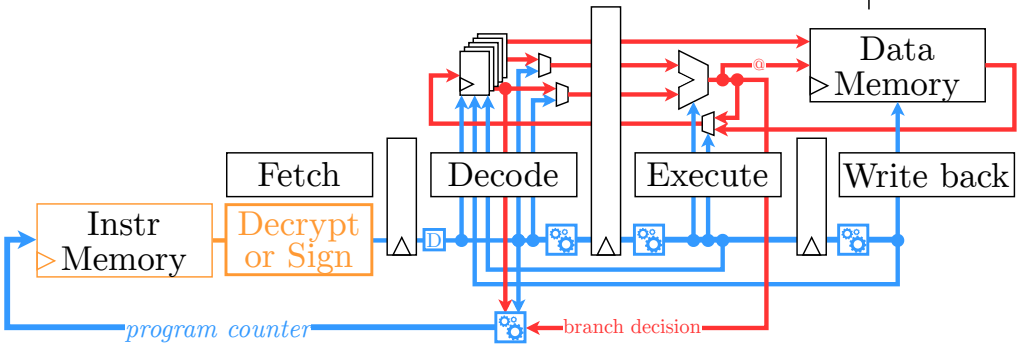
- Gpsa-Csm [11]
- Sofia [4]
- Ccfi-Cache [3]
- Scfp [10]
- Confidaent [9]
- Hapei [6]



Context for execution integrity
Program execution in a pipeline

- Gpsa-Csm [11]
- Sofia [4]
- Ccfi-Cache [3]
- Scfp [10]
- Confidaent [9]
- Hapei [6]

✔ bge r10,r11,50 ➜ blt

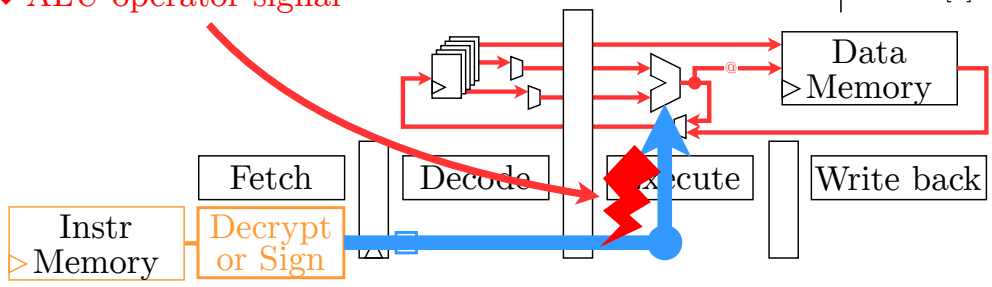


Context for execution integrity
Program execution in a pipeline

- GPSA-CSM [11]
- SOFIA [4]
- CCFI-CACHE [3]
- SCFP [10]
- CONFIDAENT [9]
- HAPEI [6]

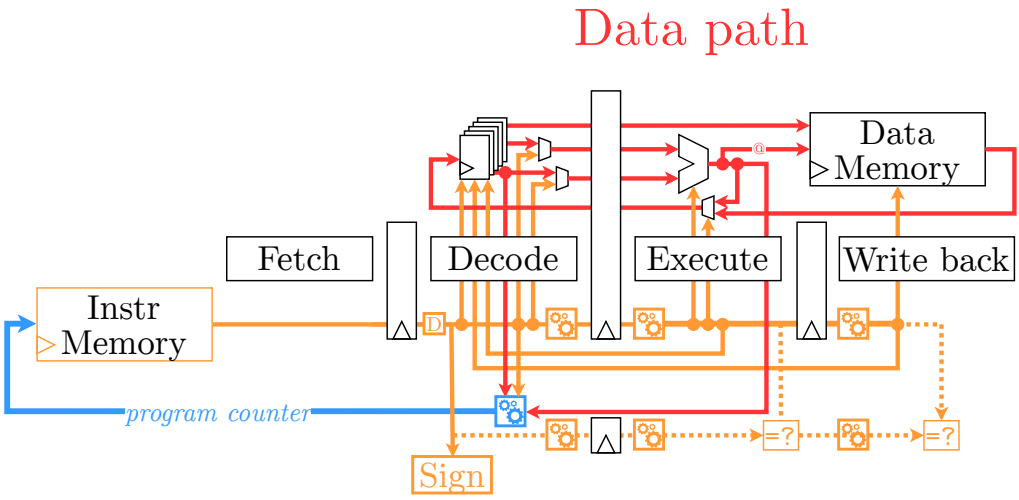
✔ bge r10,r11,50 ➜ blt

✖ ALU operator signal



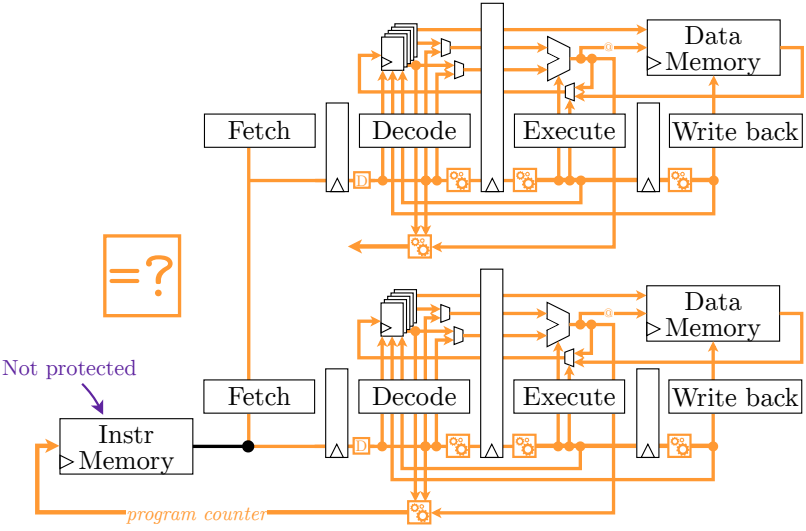
Context for execution integrity

Program execution in a pipeline



Context for execution integrity

Program execution in a pipeline



Context for execution integrity

State-of-the-art execution integrity

Name	Instruction Confidentiality	Control Flow Integrity	Instruction Integrity	Control Signal Integrity	Fine-grained CFI	No clock cycle penalty	No memory overhead
Instr. Enc.[5]	✓		✓			✓	×
Confidaent [9]	✓	✓	✓				×
Sofia [4]	✓	✓	✓		✓		×
Ccfi-Cache [3]		✓	✓		✓	✓	×
Cifer [13]		✓	✓	(✓)		✓	×
Soft-Only [1]		✓	✓				×
SecDec [8]		✓	✓	(✓)			×
Hapei [6]	✓	✓	✓		✓		×
Scfp [10]	✓	✓	✓				×
Gpsa-Csm [11]		✓	✓				×
Mafia [2]		✓	✓	✓			×
This work	✓	✓	✓	✓	✓	✓	×

Proposed scheme

Proposed scheme
Goal

Security properties

- **Integrity** of Control Flow, Instructions and Control Signals (from all stages)
- **Confidentiality** of Instructions

Can these properties be provided together ?

(by another approach than redundancy)

MINES Saint-Étienne
Une école de l'IMT

Proposed scheme
Constraints

Security properties

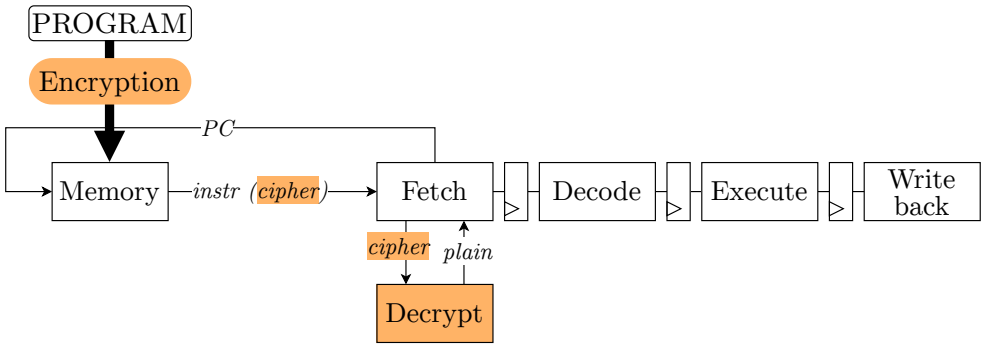
- ➜ **Integrity** of Control Flow, Instructions and Control Signals (from all stages)
- ➜ **Confidentiality** of Instructions

Constraints

- ➜ No insertion of instructions
- ✔ Commercial off-the-shelf binaries

Proposed scheme
Our mechanism

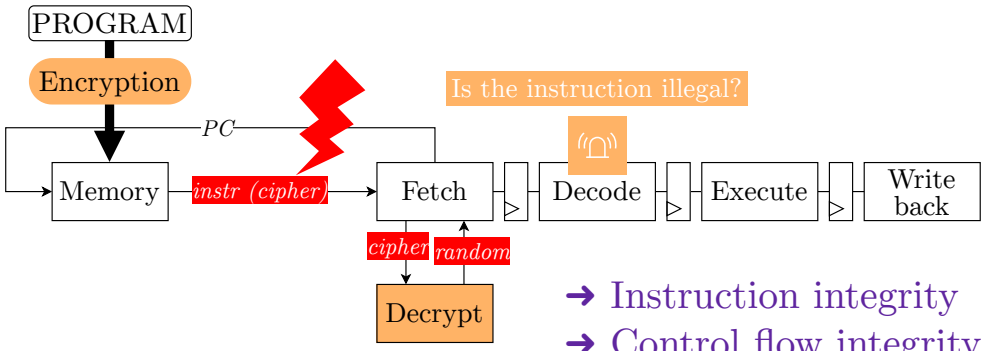
Chained Instruction Encryption with Absorption of Control Signals



1. Chained Encryption of Instructions (before programming memory)
2. On-the-fly Decryption

Proposed scheme
Our mechanism

Chained Instruction Encryption with Absorption of Control Signals

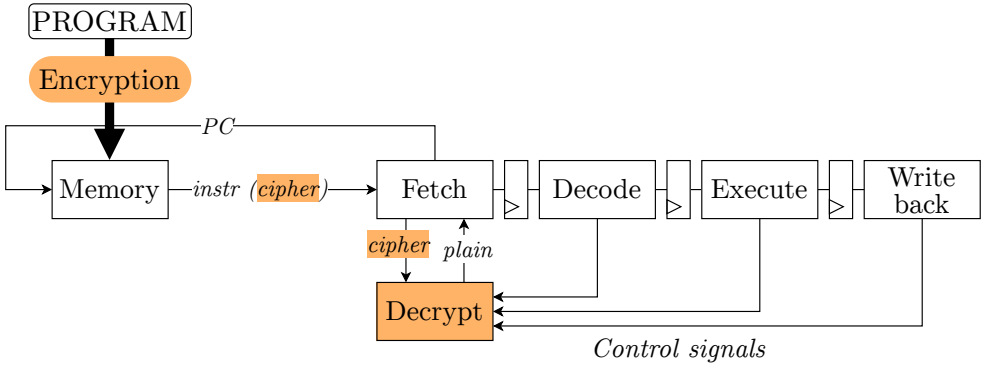


➜ Instruction integrity
➜ Control flow integrity

1. Chained Encryption of Instructions (before programming memory)
2. On-the-fly Decryption

Proposed scheme
Our mechanism

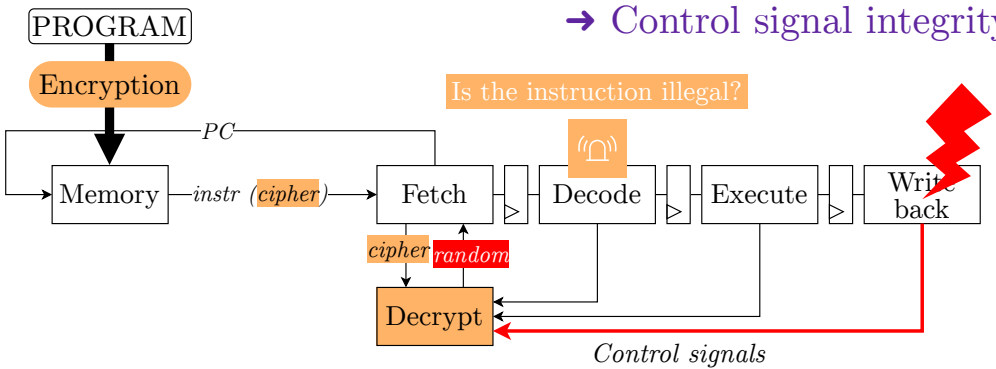
Chained Instruction Encryption with Absorption of Control Signals



1. Chained Encryption of Instructions (before programming memory)
2. On-the-fly Decryption

Proposed scheme
Our mechanism

Chained Instruction Encryption with Absorption of Control Signals



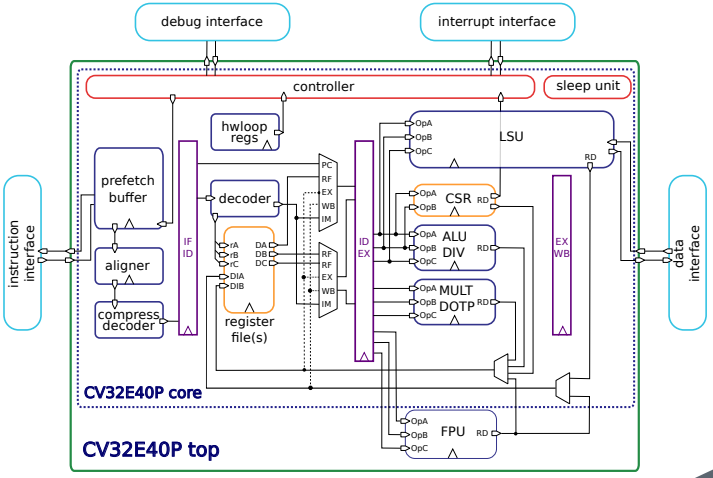
➜ Control signal integrity

1. Chained Encryption of Instructions (before programming memory)
2. On-the-fly Decryption

Proposed scheme
Implementation
RISC-V core: CV32E40P

CV32E40P: RISC-V core

- ➜ Embedded system
- 32-bit
- 4-stage
- In-order
- Forwarding



Proposed scheme
Implementation
Authenticated Encryption: Ascon

Ascon: cipher suite, which provides Authenticated Encryption with Associated Data

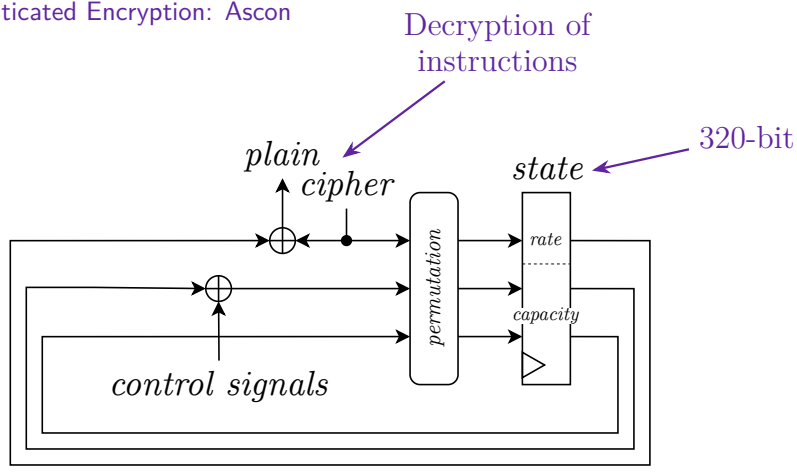
- Winner of CAESAR Authenticated Encryption - Lightweight Cryptography (2019)
- Winner of NIST - Lightweight Cryptography (2023) $\Rightarrow$ **standardize the Ascon family**
- Lightweight (better Throughput per Area than AES [1])
- Highly tested
- Large security margins

[1] *"Need for Low-latency Ciphers: A Comparative Study of NIST LWC Finalists"*, NIST LWC Workshop 2022, Tolga Yalcin - Google

Proposed scheme

Implementation

Authenticated Encryption: Ascon

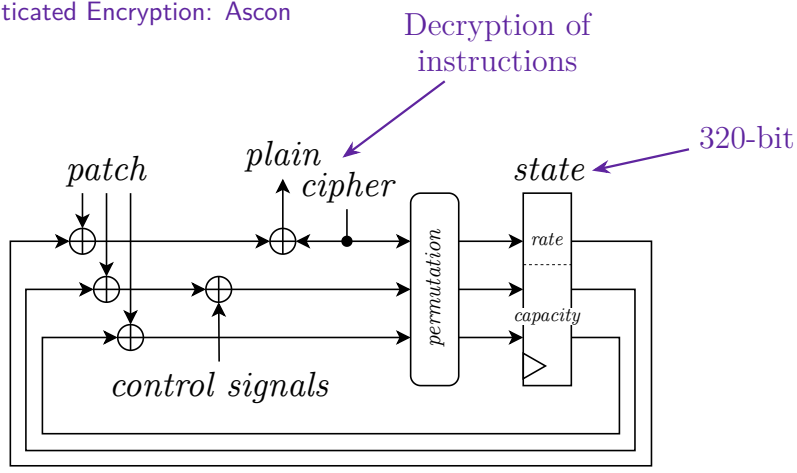


Hardware decryption

Proposed scheme

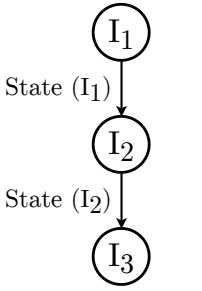
Implementation

Authenticated Encryption: Ascon

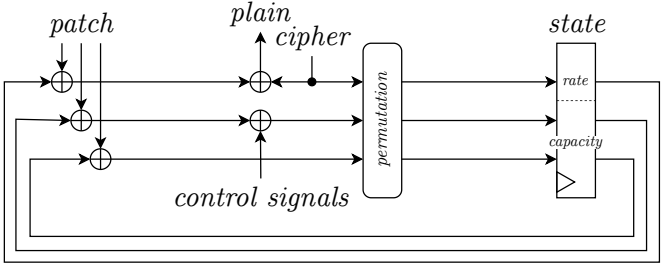


Hardware decryption

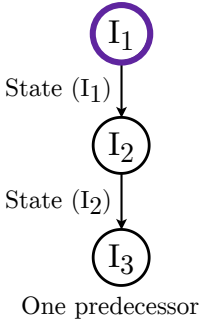
Proposed scheme
Need for patches



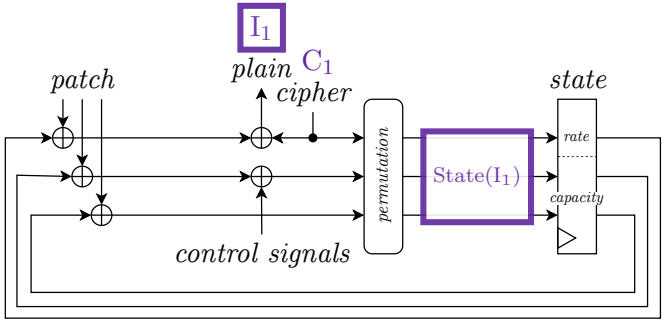
One predecessor



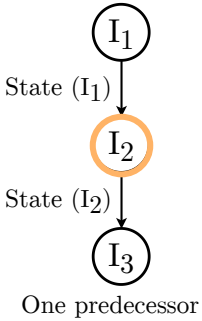
Proposed scheme
Need for patches



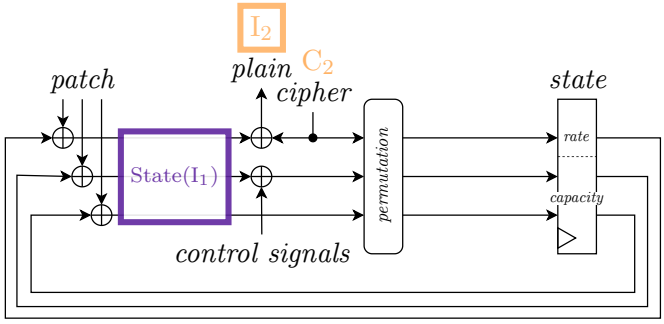
One predecessor



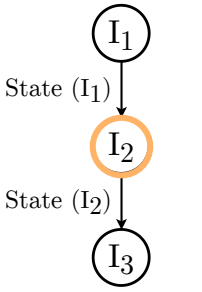
Proposed scheme
Need for patches



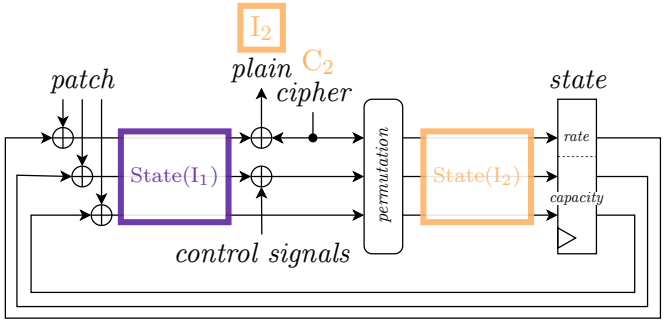
One predecessor



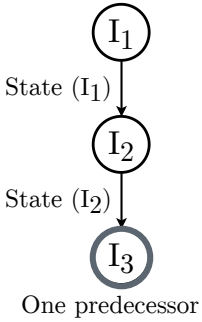
Proposed scheme
Need for patches

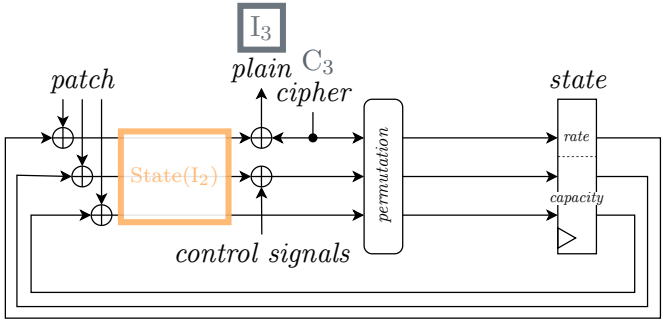


One predecessor

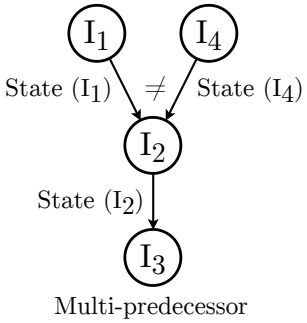


Proposed scheme
Need for patches

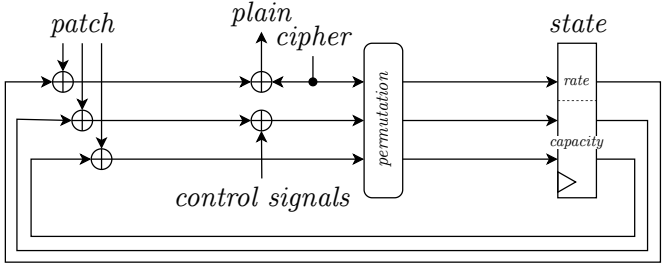




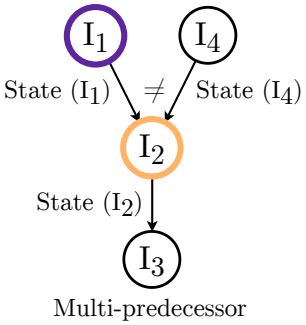
Proposed scheme
Need for patches

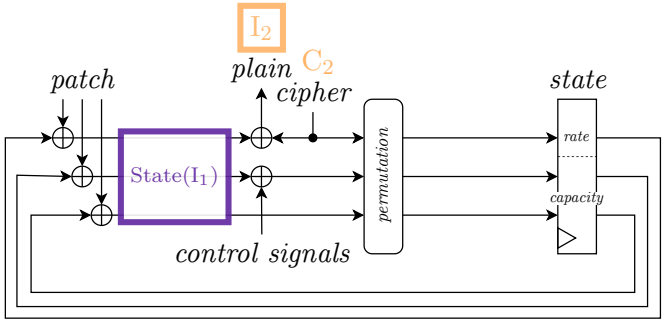


Multi-predecessor

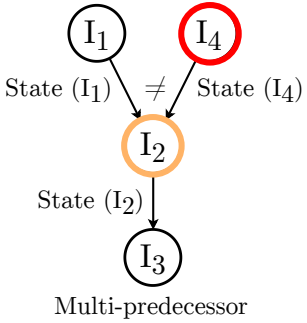


Proposed scheme
Need for patches

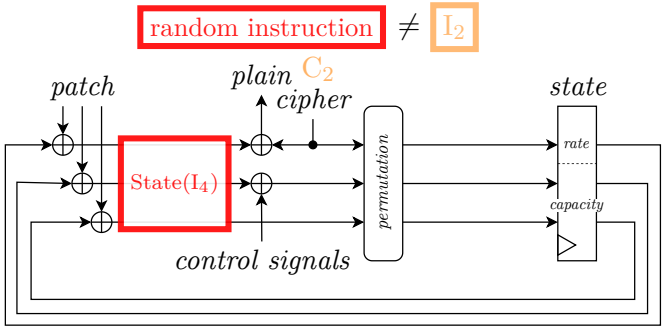




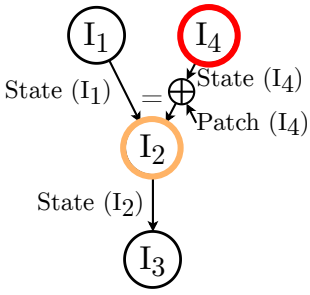
Proposed scheme
Need for patches



Multi-predecessor

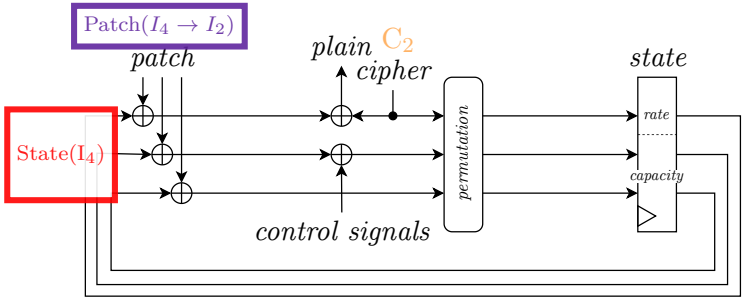


Proposed scheme
Need for patches

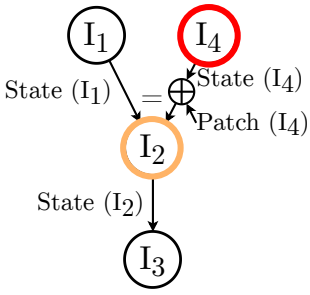


Patching multi-predecessor

Patch (I_4) = State (I_1) xor State (I_4)

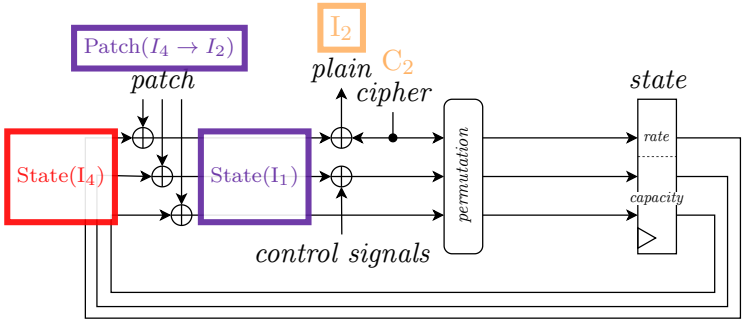


Proposed scheme
Need for patches

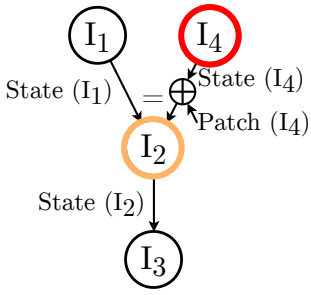


Patching multi-predecessor

Patch (I_4) = State (I_1) xor State (I_4)

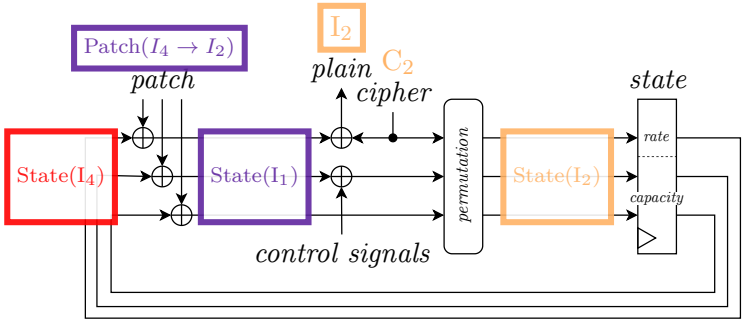


Proposed scheme
Need for patches



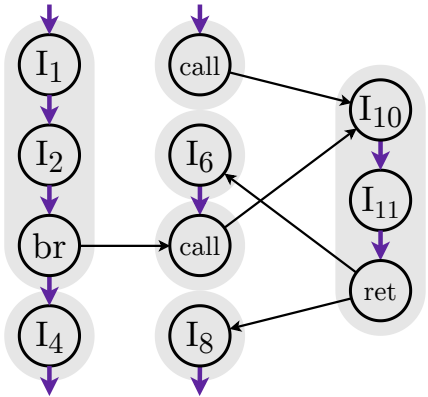
Patching multi-predecessor

Patch (I_4) = State (I_1) xor State (I_4)



Proposed scheme
Patch policy

"The instructions are encrypted from the state of the previous instruction in memory"

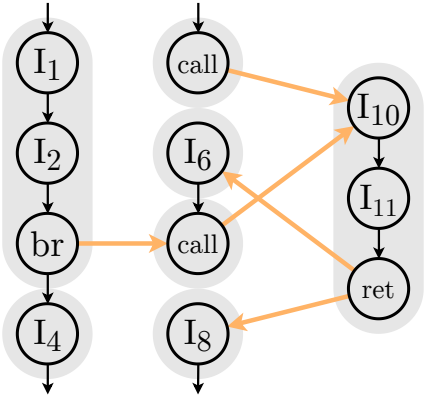


Control Flow Graph

➜ 85.5 % of instructions are sequential (Embench)

Chain sequential instr.

Proposed scheme
Patch policy



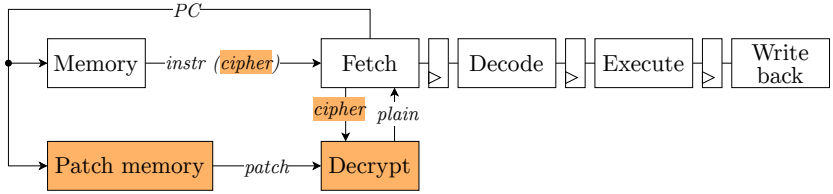
Control Flow Graph

Patches
(for non-sequential control transfer)

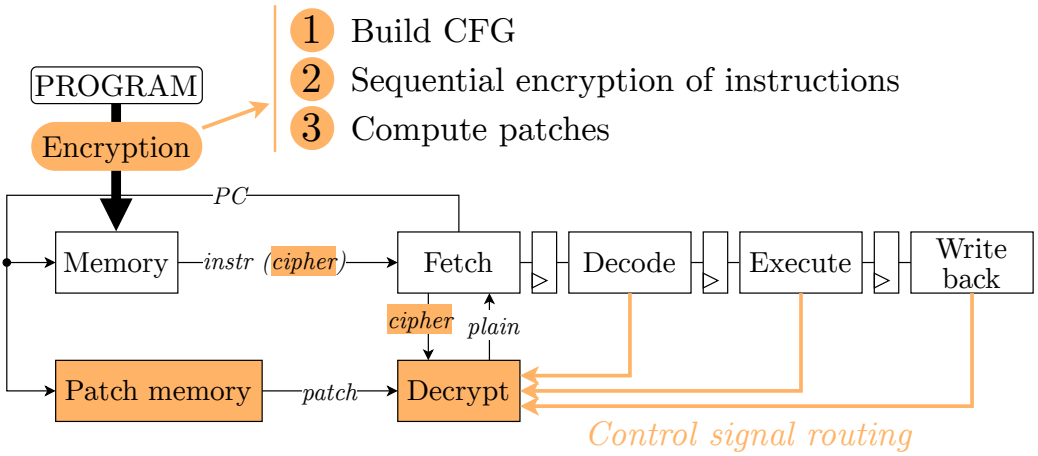
Proposed scheme
Patch policy

- Patches stored in external memory
- Patches also addressed by the Program Counter
  - ➜ No need for custom instructions

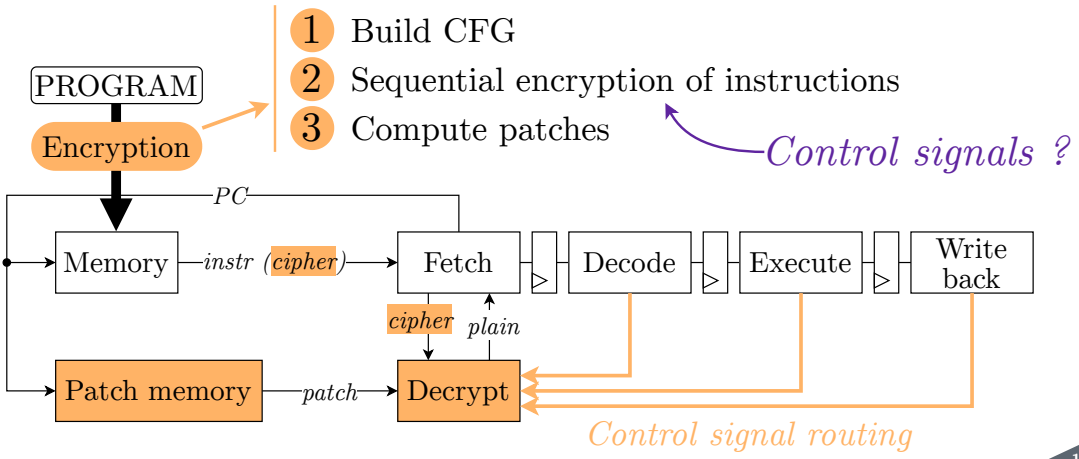
⇒ **zero clock cycle overhead**



Proposed scheme
Solution flow



Proposed scheme
Solution flow



Proposed scheme
Software model to infer control signal values

✔ Instructions (machine code)
✔ Microarchitecture netlist

➜ Infer control signal values

Software model of control signal propagation

- Analyze 41 control signals from the processor description (computation, propagation, ...)
- During sequential encryption of instructions:
  - ➜ Model signal propagation
  - ➜ Absorb control signals in the state
- Adapt patches

Proposed scheme

Software model to infer control signal values

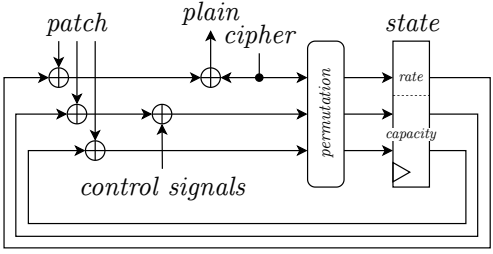
Sequential encryption of instructions

Software encryption

Algorithm 1 Sequential encryption of instructions.

```
1: state ← initialization(key, nonce)
2: for each instruction in code do
3:     state_rate ← state_rate ⊕ instruction
4:     instruction_cipher ← state_rate
5:     state ← permutation(state, 6)
```

Hardware decryption



Without signal absorption

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

Software encryption

Algorithm 1 Sequential encryption of instructions and propagation of control signals.

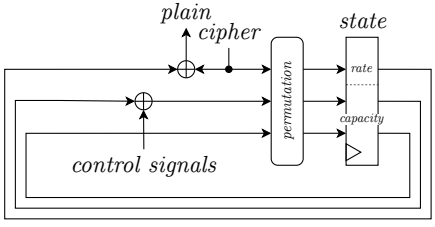
1: state ← initialization(key, nonce)
2: $CS_{modeled}[if, id, ex, wb] \leftarrow CS_{reset}$
3: **for** each instruction in code **do**
4: ▷ *Transmission between stages*
5: $CS_{modeled}[wb] \leftarrow CS_{modeled}[ex]$ / $CS_{modeled}[id]$ / $CS_{reset}(wb)$ ▷ *stall? multi-cycle?*
6: $CS_{modeled}[ex] \leftarrow CS_{modeled}[id]$ / $CS_{modeled}[if]$ / $CS_{reset}(ex)$ ▷ *stall? multi-cycle?*
7: $CS_{modeled}[id] \leftarrow CS_{modeled}[if]$ ▷ *multi-cycle?*
8:
9: ▷ *Evaluation*
10: $CS_{modeled}[if] \leftarrow$ Function(instruction) ▷ *Extracted signals ①*
11: $CS_{modeled}[if] \leftarrow$ LookUp Table(instruction) ▷ *Decoded signals ②*
12: $CS_{modeled}[id, ex, wb] \leftarrow$ Function($CS_{modeled}$) ▷ *Combinational signals ③*
13:
14: ▷ *Encryption and absorption of signals*
15: $CS \leftarrow selection(CS_{modeled})$
16: state ← state $\oplus$ $0^* || CS || 0^*$ ← With signal absorption
17: $state_{rate} \leftarrow state_{rate} \oplus$ instruction
18: instruction_cipher ← $state_{rate}$
19: state ← permutation(state, 6)

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

➜ Absorb control signals in the state



➜ Model signal propagation

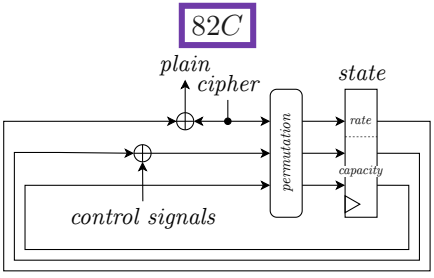
cyc	MEM	IF	ID	EX	WB
0	82C	828	18	03	
1	830	82C			
2	834	830			
3	838	834			
	addresses		alu_operator		

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

➜ Absorb control signals in the state



➜ Model signal propagation

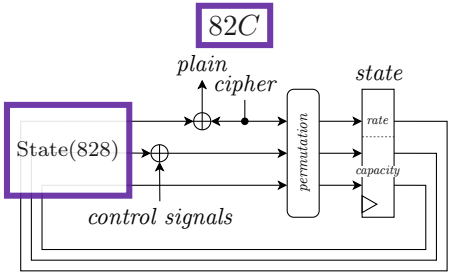
cyc	MEM	IF	ID	EX	WB
0	82C	828	18	03	
1	830	82C			
2	834	830			
3	838	834			
	addresses		alu_operator		

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

➜ Absorb control signals in the state



➜ Model signal propagation

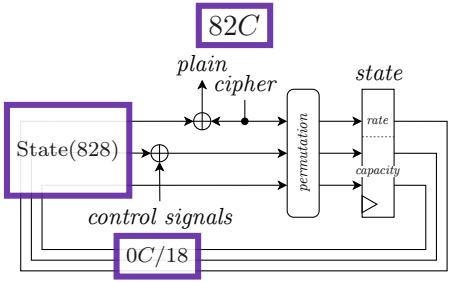
cyc	MEM	IF	ID	EX	WB
0	82C	828	18	03	
1	830	82C			
2	834	830			
3	838	834			
	addresses		alu_operator		

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

➜ Absorb control signals in the state



➜ Model signal propagation

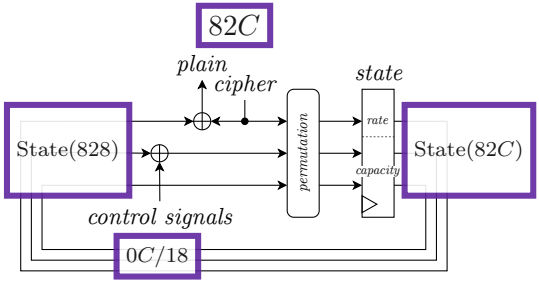
cyc	MEM	IF	ID	EX	WB
0	82C	828	18	03	
1	830	82C	0C	18	
2	834	830			
3	838	834			
	addresses		alu_operator		

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

➜ Absorb control signals in the state



➜ Model signal propagation

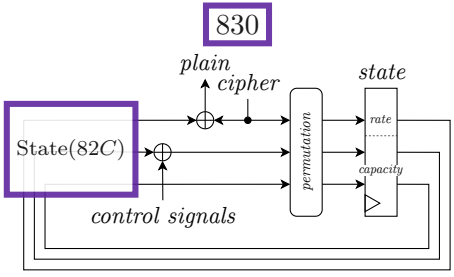
cyc	MEM	IF	ID	EX	WB
0	82C	828	18	03	
1	830	82C	0C	18	
2	834	830			
3	838	834			
	addresses		alu_operator		

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

➜ Absorb control signals in the state



➜ Model signal propagation

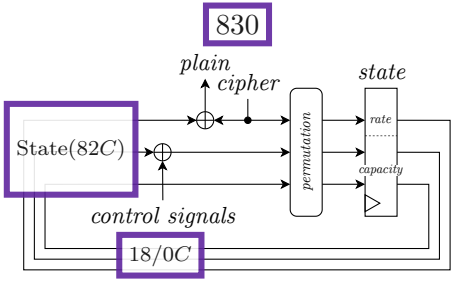
cyc	MEM	IF	ID	EX	WB
0	82C	828	18	03	
1	830	82C	0C	18	
2	834	830			
3	838	834			
	addresses		alu_operator		

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

➜ Absorb control signals in the state



➜ Model signal propagation

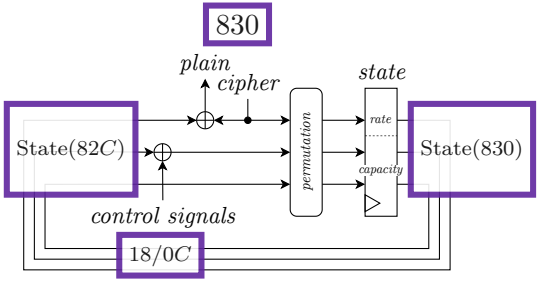
cyc	MEM	IF	ID	EX	WB
0	82C	828	18	03	
1	830	82C	0C	18	
2	834	830	18	0C	
3	838	834			
	addresses		alu_operator		

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

➜ Absorb control signals in the state



➜ Model signal propagation

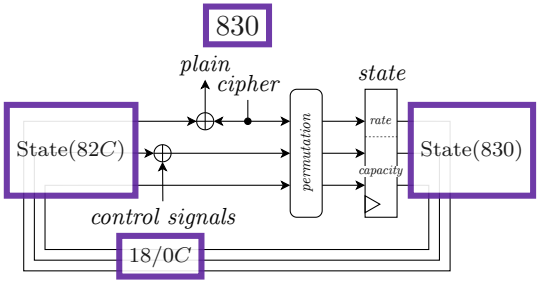
cyc	MEM	IF	ID	EX	WB
0	82C	828	18	03	
1	830	82C	0C	18	
2	834	830	18	0C	
3	838	834			
	addresses		alu_operator		

Proposed scheme

Software model to infer control signal values

Sequential encryption of instructions

➜ Absorb control signals in the state



➜ Model signal propagation

cyc	MEM	IF	ID	EX	WB
0	82C	828	18	03	
1	830	82C	0C	18	
2	834	830	18	0C	
3	838	834			
	addresses		alu_operator		

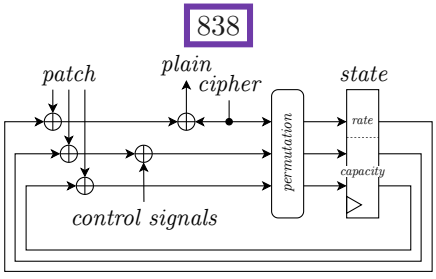
✔ Stall due to multi-cycle instructions

✔ Stall due to data dependancies

Proposed scheme

Software model to infer control signal values

Control transfer: adapt patches

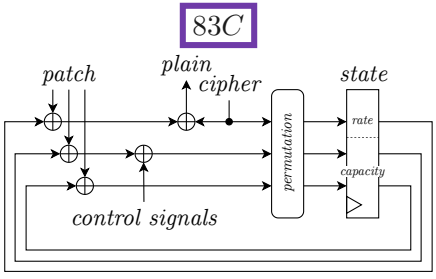


cyc	MEM	IF	ID	EX	WB	
0	82C	828	18	03		
1	830	82C	0C	18		
2	834	830	18	0C		
3	838	834	18	18		
4	83C	838	18	18		*branch fetched*
5						
6						
7						
8						
	addresses		alu_operator			

Proposed scheme

Software model to infer control signal values

Control transfer: adapt patches

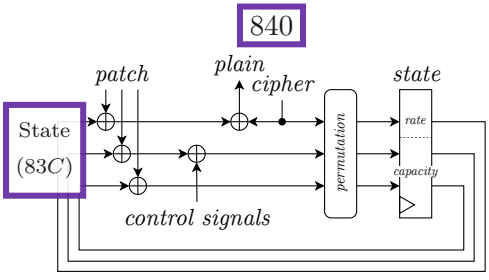


cyc	MEM	IF	ID	EX	WB	
0	82C	828	18	03		
1	830	82C	0C	18		
2	834	830	18	0C		
3	838	834	18	18		
4	83C	838	18	18		*branch fetched*
5	840	83C	0D	18		*branch decoded*
6						
7						
8						
	addresses		alu_operator			

Proposed scheme

Software model to infer control signal values

Control transfer: adapt patches



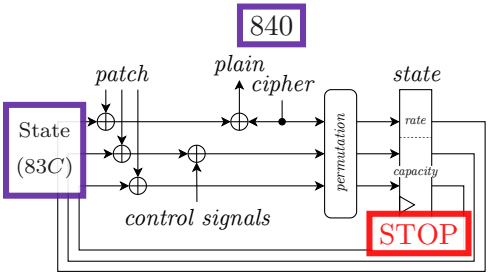
flush

cyc	MEM	IF	ID	EX	WB	
0	82C	828	18	03		
1	830	82C	0C	18		
2	834	830	18	0C		
3	838	834	18	18		
4	83C	838	18	18		*branch fetched*
5	840	83C	0D	18		*branch decoded*
6	82C	840	18	0D		*branch taken*
7						
8						
	addresses		alu_operator			

Proposed scheme

Software model to infer control signal values

Control transfer: adapt patches



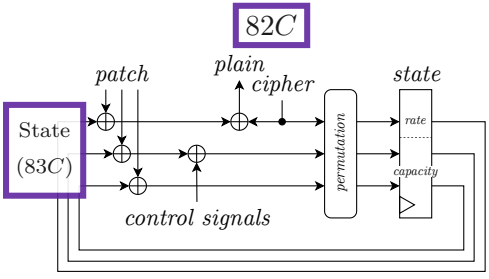
flush

cyc	MEM	IF	ID	EX	WB	
0	82C	828	18	03		
1	830	82C	0C	18		
2	834	830	18	0C		
3	838	834	18	18		
4	83C	838	18	18		*branch fetched*
5	840	83C	0D	18		*branch decoded*
6	82C	840	18	0D		*branch taken*
7						
8						
	addresses		alu_operator			

Proposed scheme

Software model to infer control signal values

Control transfer: adapt patches



cyc	MEM	IF	ID	EX	WB	
0	82C	828	18	03		
1	830	82C	0C	18		
2	834	830	18	0C		
3	838	834	18	18		
4	83C	838	18	18		*branch fetched*
5	840	83C	0D	18		*branch decoded*
6	82C	840	18	0D		*branch taken*
7	830	82C	18	03		
8						
		addresses		alu_operator		

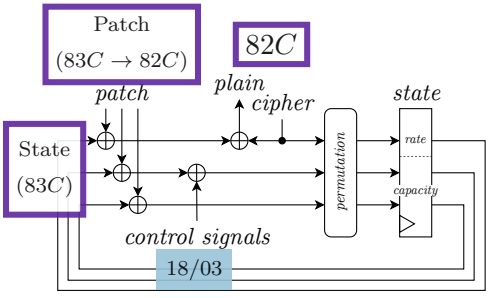
flush

stall

Proposed scheme

Software model to infer control signal values

Control transfer: adapt patches



cyc	MEM	IF	ID	EX	WB	
0	82C	828	18	03		
1	830	82C	0C	18		
2	834	830	18	0C		
3	838	834	18	18		
4	83C	838	18	18		*branch fetched*
5	840	83C	0D	18		*branch decoded*
6	82C	840	18	0D		*branch taken*
7	830	82C	18	03		
8						
		addresses		alu_operator		

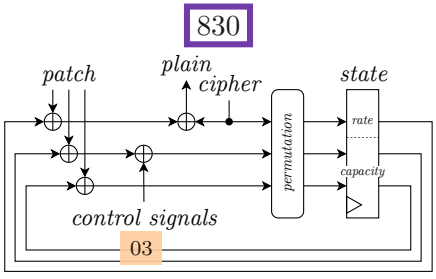
flush

stall

Proposed scheme

Software model to infer control signal values

Control transfer: adapt patches



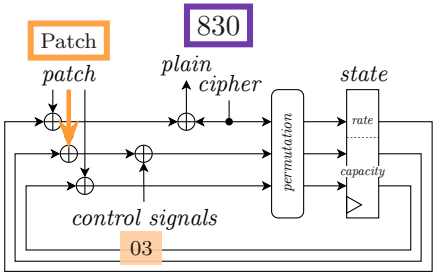
cyc	MEM	IF	ID	EX	WB	
0	82C	828	18	03		
1	830	82C	0C	18		
2	834	830	18	0C		
3	838	834	18	18		
4	83C	838	18	18		*branch fetched*
5	840	83C	0D	18		*branch decoded*
6	82C	840	18	0D		*branch taken*
7	830	82C	18	03		
8	834	830	18	03		
	addresses		alu_operator			

stall

Proposed scheme

Software model to infer control signal values

Control transfer: adapt patches



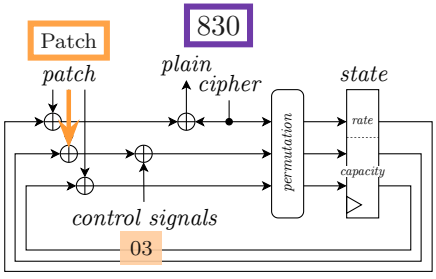
cyc	MEM	IF	ID	EX	WB	
0	82C	828	18	03		
1	830	82C	0C	18		
2	834	830	18	0C		
3	838	834	18	18		
4	83C	838	18	18		*branch fetched*
5	840	83C	0D	18		*branch decoded*
6	82C	840	18	0D		*branch taken*
7	830	82C	18	03		
8	834	830	18	03		
	addresses		alu_operator			

stall

Proposed scheme

Software model to infer control signal values

Control transfer: adapt patches



cyc	MEM	IF	ID	EX	WB	
0	82C	828	18	03		
1	830	82C	0C	18		
2	834	830	18	0C		
3	838	834	18	18		
4	83C	838	18	18		*branch fetched*
5	840	83C	0D	18		*branch decoded*
6	82C	840	18	0D		*branch taken*
7	830	82C	18	03		
8	834	830	18	03		
		addresses		alu_operator		

stall

- ✔ Stall/flush due to *jump*, *branch* execution
- ➜ patches are extended

Validation and characterization

Validation and characterization

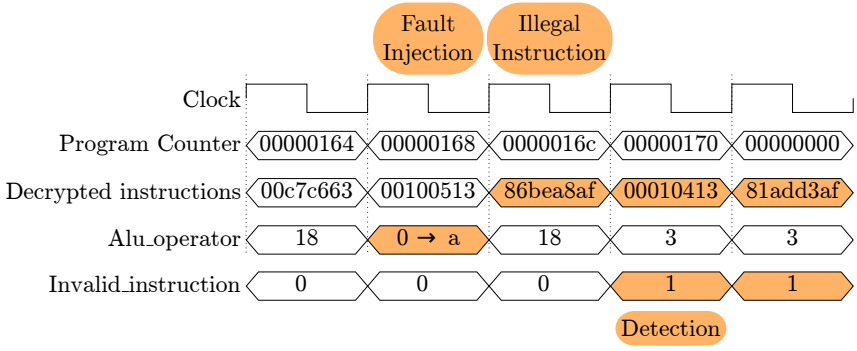
core-v-verif-fpga environnement

Makefile based:

- ➜ Compilation (*Embench* 22 programs) and software encryption flow
- ➜ **Cycle-accurate** simulations with assertion of PC and instruction values (*Verilator*)
- ➜ Execution and validation on **FPGA** board
- ➜ **FIA**: emulated on FPGA (memory only) and on simulation (memory and control signals)

Validation and characterization

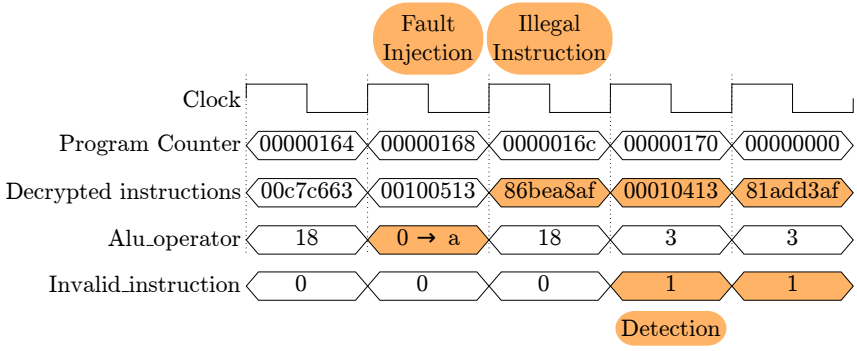
Experimental validation



FIA ➜ Incorrect state ➜ Decryption = random instruction ➜ Illegal instruction

Validation and characterization

Experimental validation



FIA ➜ Incorrect state ➜ Decryption = random instruction ➜ Illegal instruction

Original code is not executed anymore

Detection

MINES Saint-Étienne
Une école de l'IMT

Conclusion

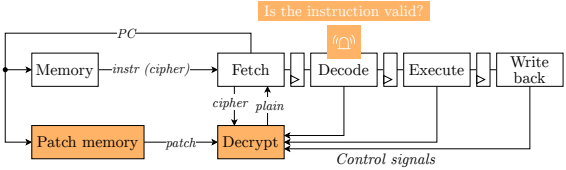
Conclusion

Can these properties be provided together ?

Chained Instruction Encryption with Absorption of Control Signals:

- ✔ **Confidentiality** of Instructions
- ✔ **Integrity** of Control Flow
- ✔ **Integrity** of Instructions
- ✔ **Integrity** of (data-independent) Control Signals

- ✔ **Open-source:** Github
- ✔ Implementation on CV32E40P
- ✔ Validation and Characterization on FPGA
  - ✔ FIA on VerifyPin execution
  - ➜ Memory: $\times$ 10
    - ➜ LUT: $+$ 29.8 % (1 PU)
    - ➜ Cycles: $+$ 0 %
  - ➜ Frequency: $-$ 85.9 % (1 PU)

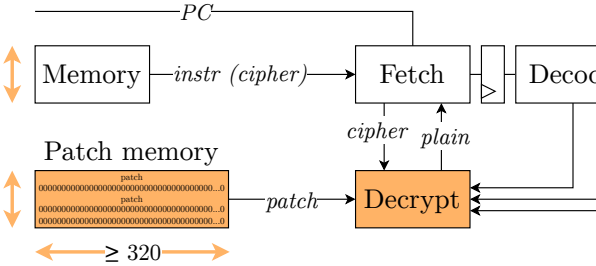


github.com/theophile-gousselot/execution_integrity_down_to_control_signals

Conclusion
Limitations and perspectives

Control Signals can be absorbed in some CFI solutions

- ✔ **Patch policy ➜ Fine-grained CFI**
  - ➜ 0 clock cycle overhead (COTS binaries)
  - ✖ But, patch memory as high as instr. mem.
  - ✖ But, empty at 75 %
- ✔ **Ascon ➜** New standard for lightweight crypto.
  - ✖ But, patches are 320 bits wide
  - ➜ Consider a 64 bits state (e.g., Prince)
- ✔ **Infer control signal values**
  - ➜ By analyzing the microarchitecture and the program
  - ✔ CV32E40P, in-order, no cache



github.com/theophile-gousselot/execution_integrity_down_to_control_signals

MERCI

MINES Saint-Étienne
Une école de l'IMT

Bibliography

[1] François Bonnal, Vincent Dupaquis, Olivier Potin, and Jean-Max Dutertre.
Software-only control-flow integrity against fault injection attacks.
In 2023 26th Euromicro Conference on Digital System Design (DSD), pages 269–277. IEEE, 2023.

[2] Thomas Chamelot, Damien Couroussé, and Karine Heydemann.
Mafia: Protecting the microarchitecture of embedded systems against fault injection attacks.
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023.

[3] Jean-Luc Danger, Adrien Facon, Sylvain Guilley, Karine Heydemann, Ulrich Kühne, Abdelmalek Si Merabet, and Michaël Timbert.
Ccfi-cache: A transparent and flexible hardware protection for code and control-flow integrity.
In 2018 21st Euromicro Conference on Digital System Design (DSD), pages 529–536. IEEE, 2018.

[4] Ruan De Clercq, Johannes Götzfried, David Übler, Pieter Maene, and Ingrid Verbauwhede.
Sofia: Software and control flow integrity architecture.
Computers & Security, 68:16–35, 2017.

[5] Thomas Hiscock, Olivier Savry, and Louis Goubin.
Lightweight instruction-level encryption for embedded processors using stream ciphers.
Microprocessors and Microsystems, 64:43–52, 2019.

[6] Ronan Lashermes, Hélène Le Bouder, and Gaël Thomas.
Hardware-assisted program execution integrity: Hapei.
In Secure IT Systems: 23rd Nordic Conference, NordSec 2018, Oslo, Norway, November 28-30, 2018, Proceedings 23, pages 405–420. Springer, 2018.

[7] Johan Laurent, Vincent Beroulle, Christophe Deleuze, Florian Pebay-Peyroula, and Athanasios Papadimitriou.
Cross-layer analysis of software fault models and countermeasures against hardware fault attacks in a risc-v processor.
Microprocessors and Microsystems, 71:102862, 2019.

MINES Saint-Étienne
Une école de l'IMT

Bibliography

[8] Gaëtan Leplus, Olivier Savry, and Lilian Bossuet.
Secdec: Secure decode stage thanks to masking of instructions with the generated signals.
In 2022 25th Euromicro Conference on Digital System Design (DSD), pages 556–563. IEEE, 2022.

[9] Olivier Savry, Mustapha El-Majihi, and Thomas Hiscock.
Confidaent: Control flow protection with instruction and data authenticated encryption.
In 2020 23rd Euromicro Conference on Digital System Design (DSD), pages 246–253. IEEE, 2020.

[10] Mario Werner, Thomas Unterluggauer, David Schaffenrath, and Stefan Mangard.
Sponge-based control-flow protection for iot devices.
In 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pages 214–226. IEEE, 2018.

[11] Mario Werner, Erich Wenger, and Stefan Mangard.
Protecting the control flow of embedded processors against fault attacks.
In Smart Card Research and Advanced Applications: 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers 14, pages 161–176. Springer, 2016.

[12] Bilgiday Yuce, Patrick Schaumont, and Marc Witteman.
Fault attacks on secure embedded software: Threats, design, and evaluation.
Journal of Hardware and Systems Security, 2:111–130, 2018.

[13] Anthony Zgheib, Olivier Potin, Jean-Baptiste Rigaud, and Jean-Max Dutertre.
Cifer: Code integrity and control flow verification for programs executed on a risc-v core.
In 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pages 100–110. IEEE, 2023.