Sponge-Based Control-Flow Protection for IoT Devices

Werner, Unterluggauer, Schaffenrath, Mangard

25th April 2018, London

Graz University of Technology

Motivation and Context

Logical Attacks

- Exploit software and design bugs
- Mounted via external interfaces
- Applicable via the Internet
- Attack techniques:
  - Code: code injection, ret2libc, ROP, JOP
  - Data: DOP
- Countermeasures:
  - Correct software
  - SW: W$\oplus$X, ASLR, CFI, CPI, DFI, WIT
  - HW: processor privilege levels and access control

Physical Attacks

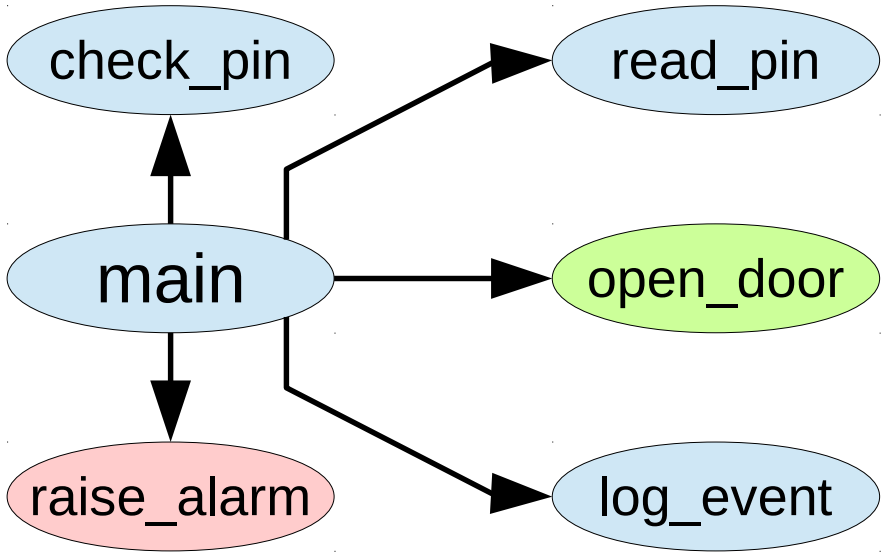
- Tamper with the operation conditions to induce faults
- Exploit the physical access to a device
- Mostly local exploitation (IoT, cloud)
- Huge portfolio of attacks [BDL97]
  and countermeasures for crypto [Bar+04]
- Only little work on protecting processors [Cle+16]
- Prominent example: Xbox 360 reset glitch hack

Contribution

- Sponge-based Control-Flow Protection (SCFP)
  - Hardware supported Control-Flow Integrity (CFI) scheme
  - Encrypts the instruction stream with instruction granularity
  - Protects against logical and physical attacks
- Present and analyzed two suitable sponge constructions
- Discuss three SCFP instantiations (IE, AEE, AEE-Light)
- Evaluate AEE-Light in a RISC-V processor
  - 9.1 % runtime overhead
  - 19.8 % code size overhead

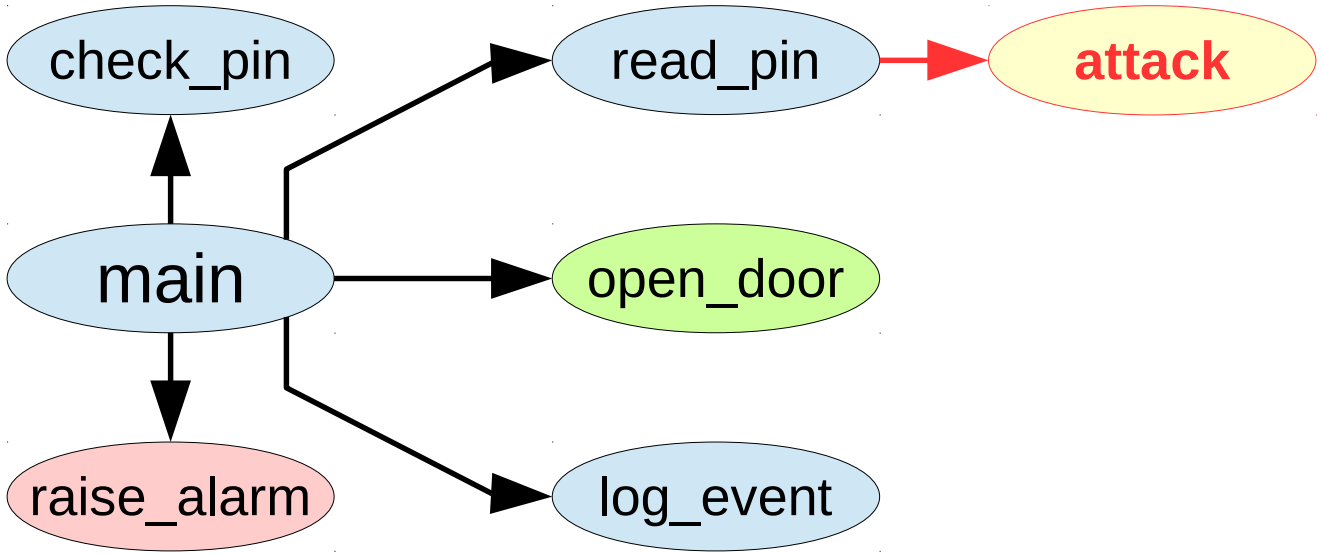
Why Control-Flow Integrity?

```
unsigned pin = read_pin();
bool auth = check_pin(pin);
if( auth ) {
    open_door();
} else {
    raise_alarm();
}
log_event();
```



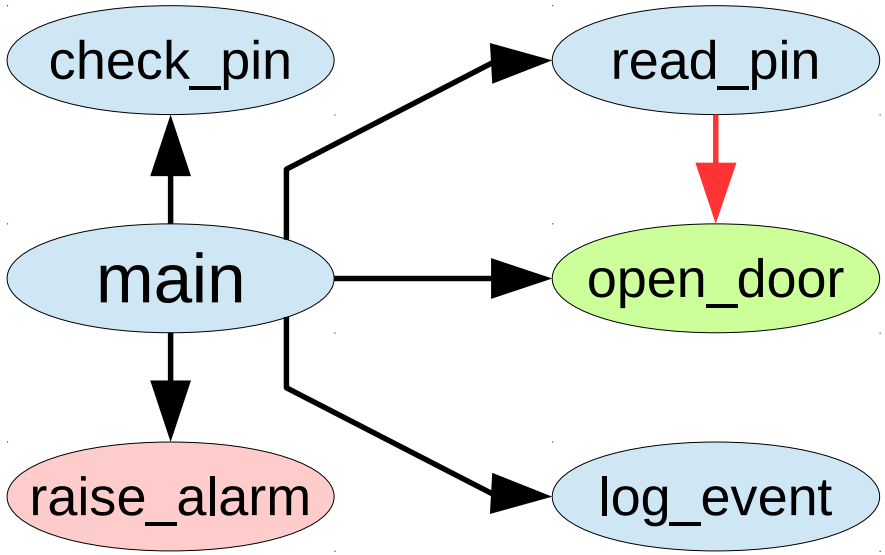
Why Control-Flow Integrity?

```
unsigned pin = read_pin();
bool auth = check_pin(pin);
if( auth ) {
    open_door();
} else {
    raise_alarm();
}
log_event();
```



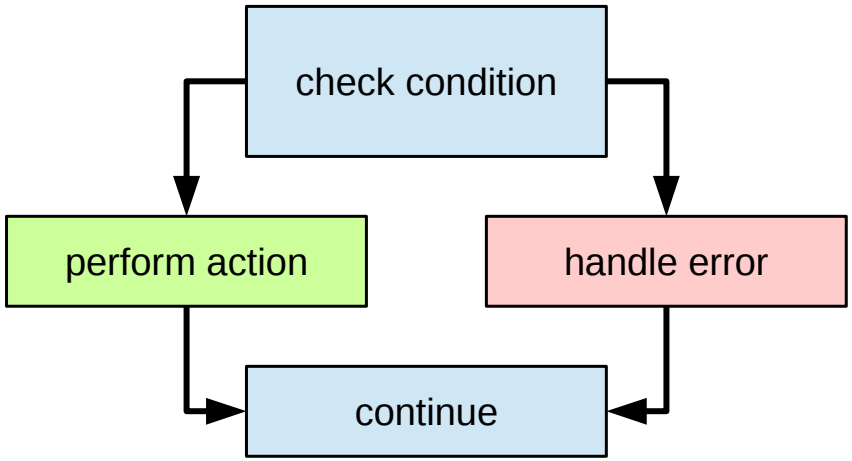
Why Control-Flow Integrity?


```
unsigned pin = read_pin();
bool auth = check_pin(pin);
if( auth ) {
    open_door();
} else {
    raise_alarm();
}
log_event();
```



Why Control-Flow Integrity?

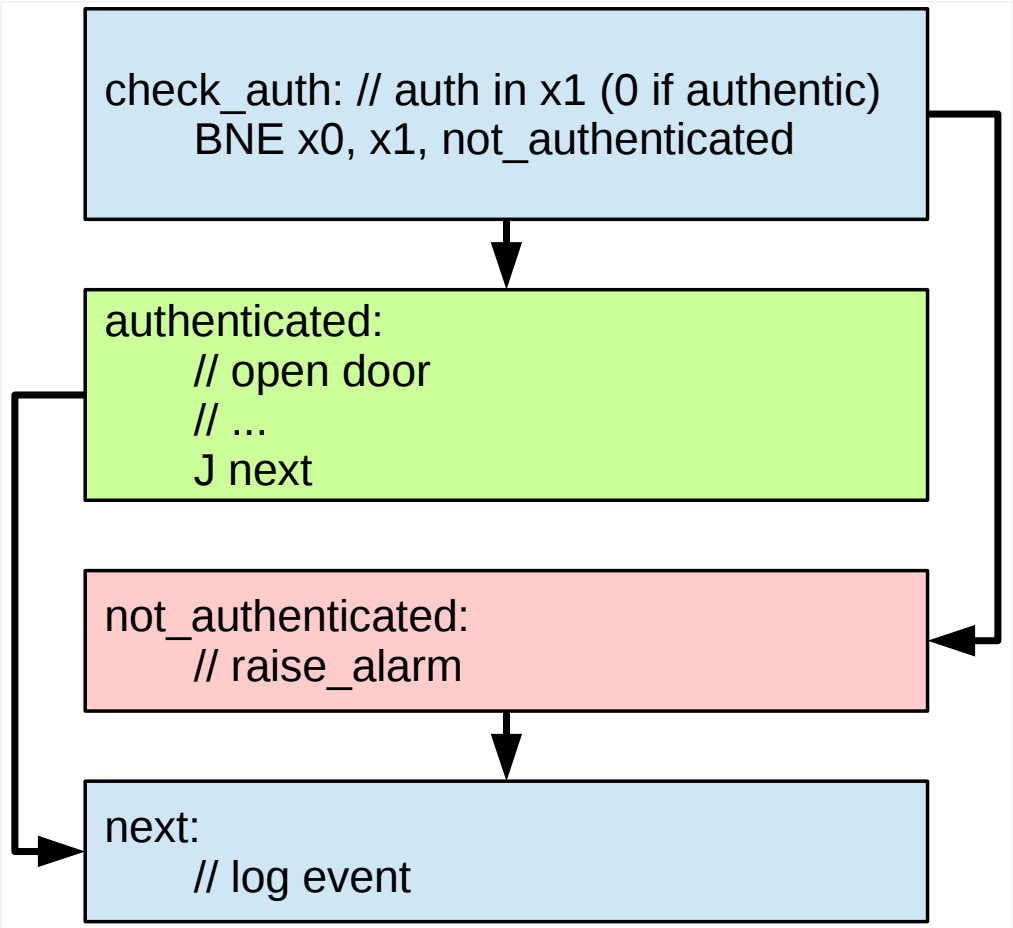
```
unsigned pin = read_pin();
bool auth = check_pin(pin);
if( auth ) {
    open_door();
} else {
    raise_alarm();
}
log_event();
```



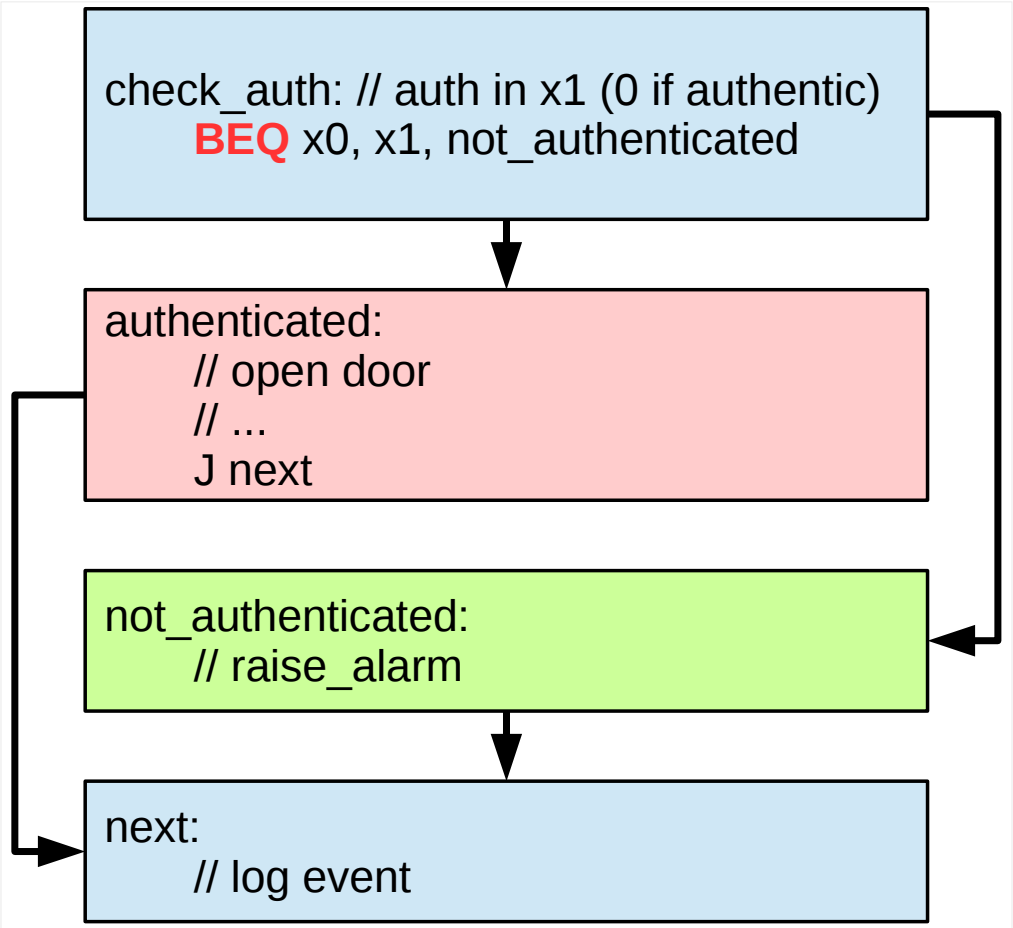
Why Control-Flow Integrity?


```
check_auth: // auth in x1 (0 if authentic)
        BNE x0, x1, not_authenticated

authenticated:
        // open door
        // ...
        J next

not_authenticated:
        // raise_alarm

next:
        // log event
```

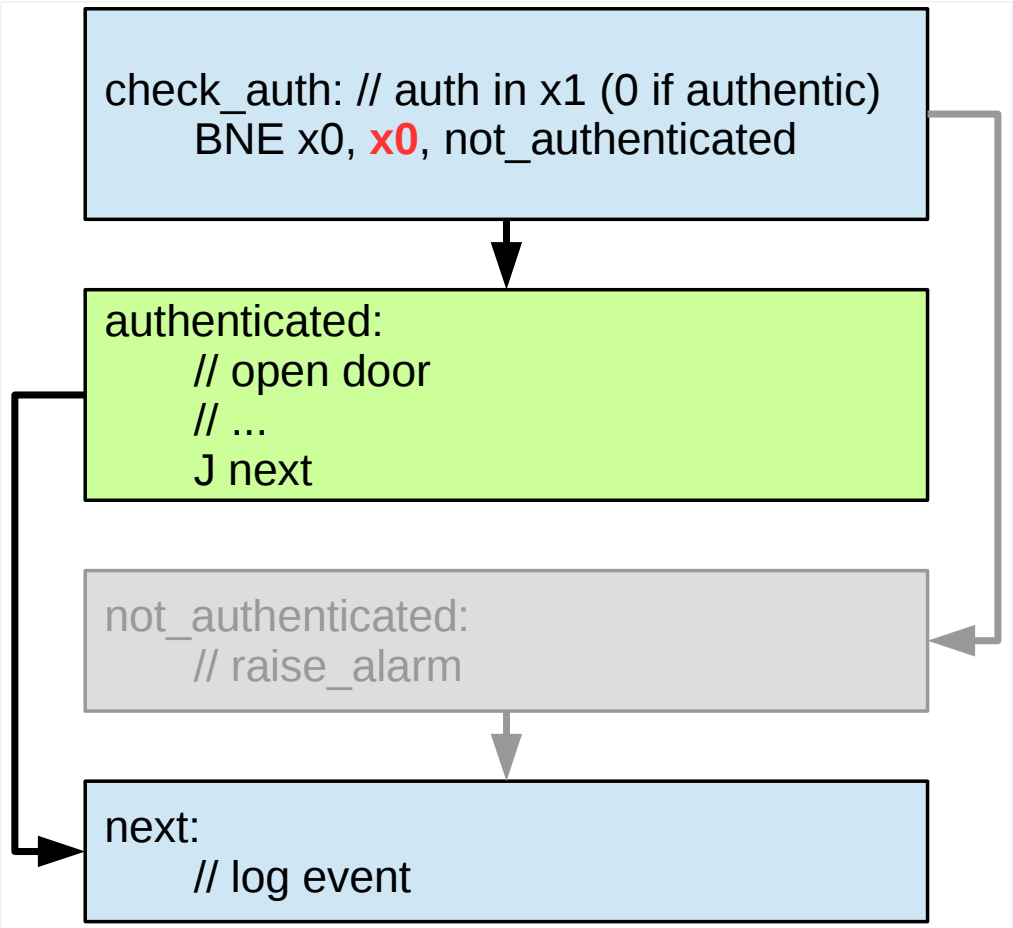
Why Control-Flow Integrity?



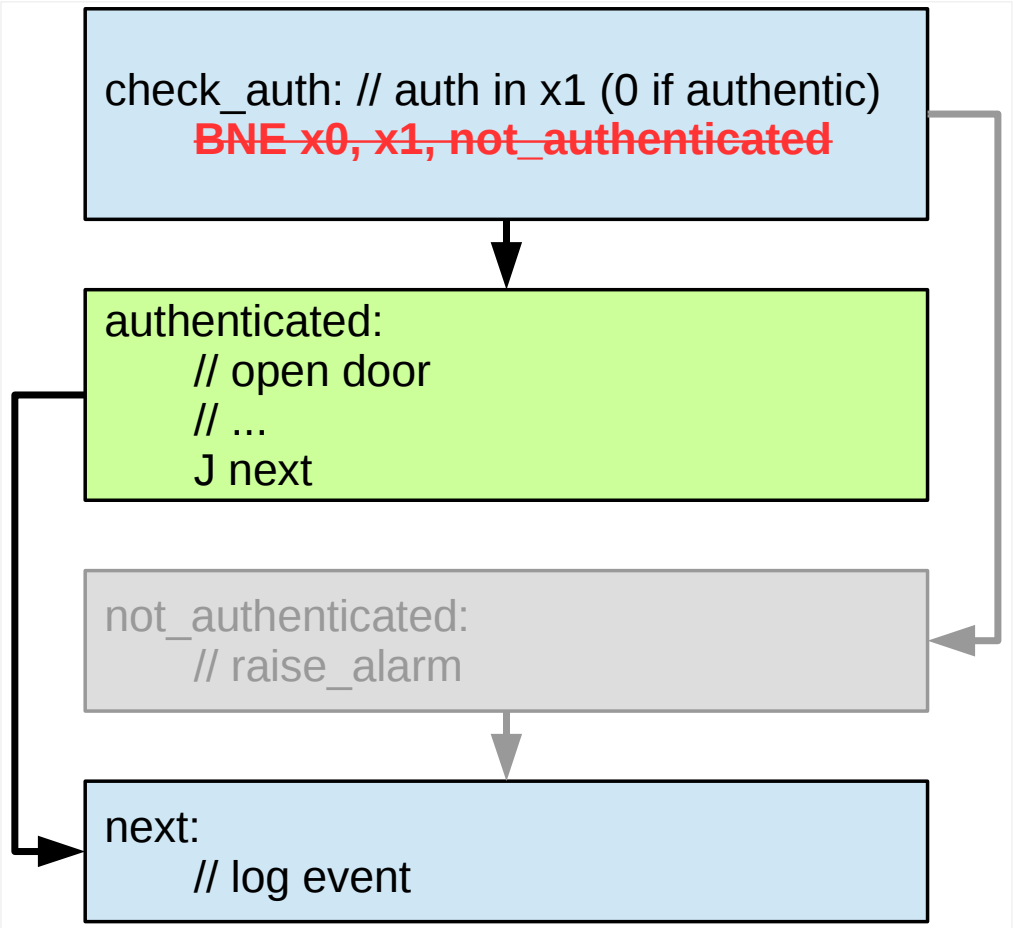
Why Control-Flow Integrity?



Why Control-Flow Integrity?

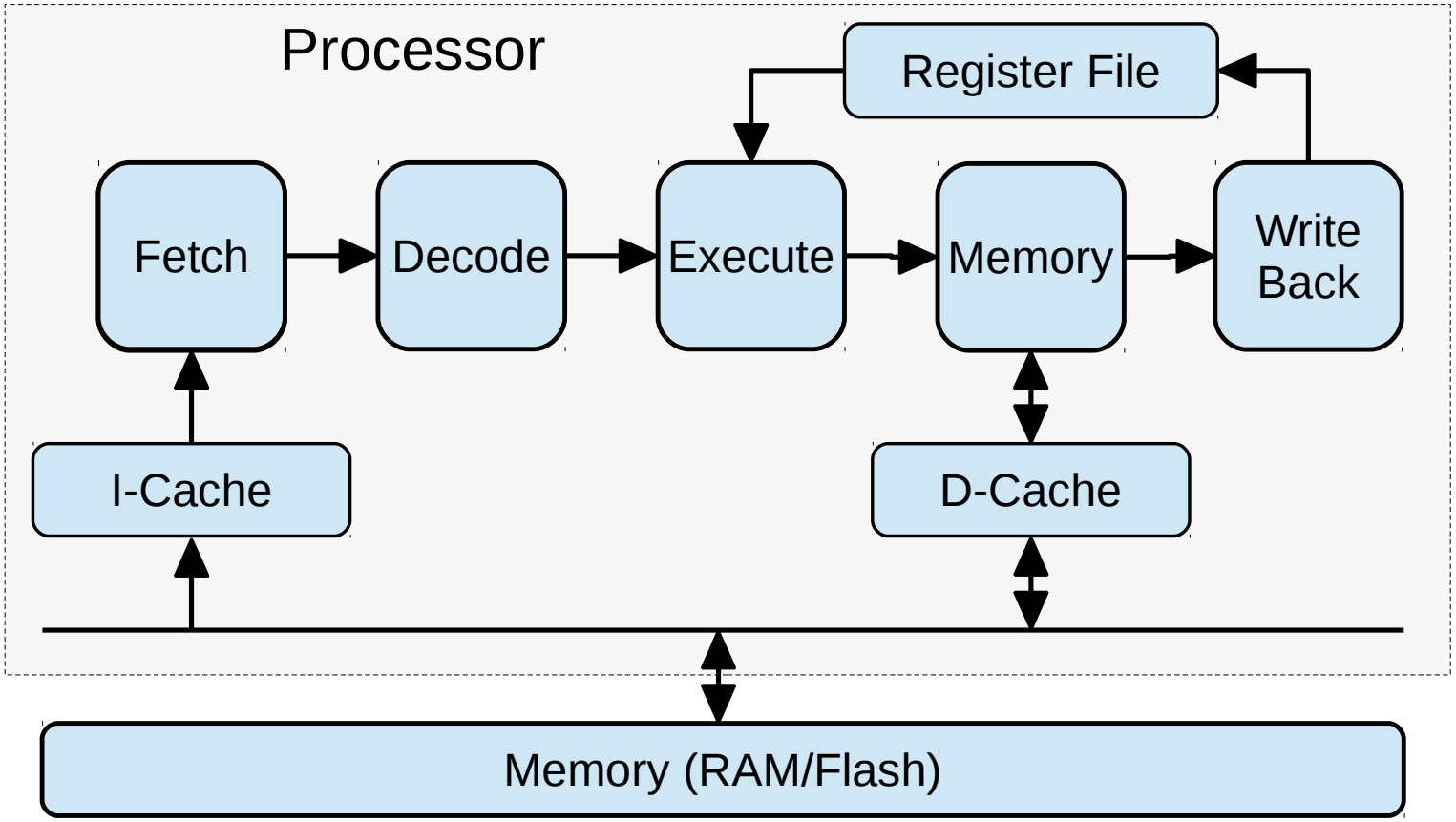


Why Control-Flow Integrity?

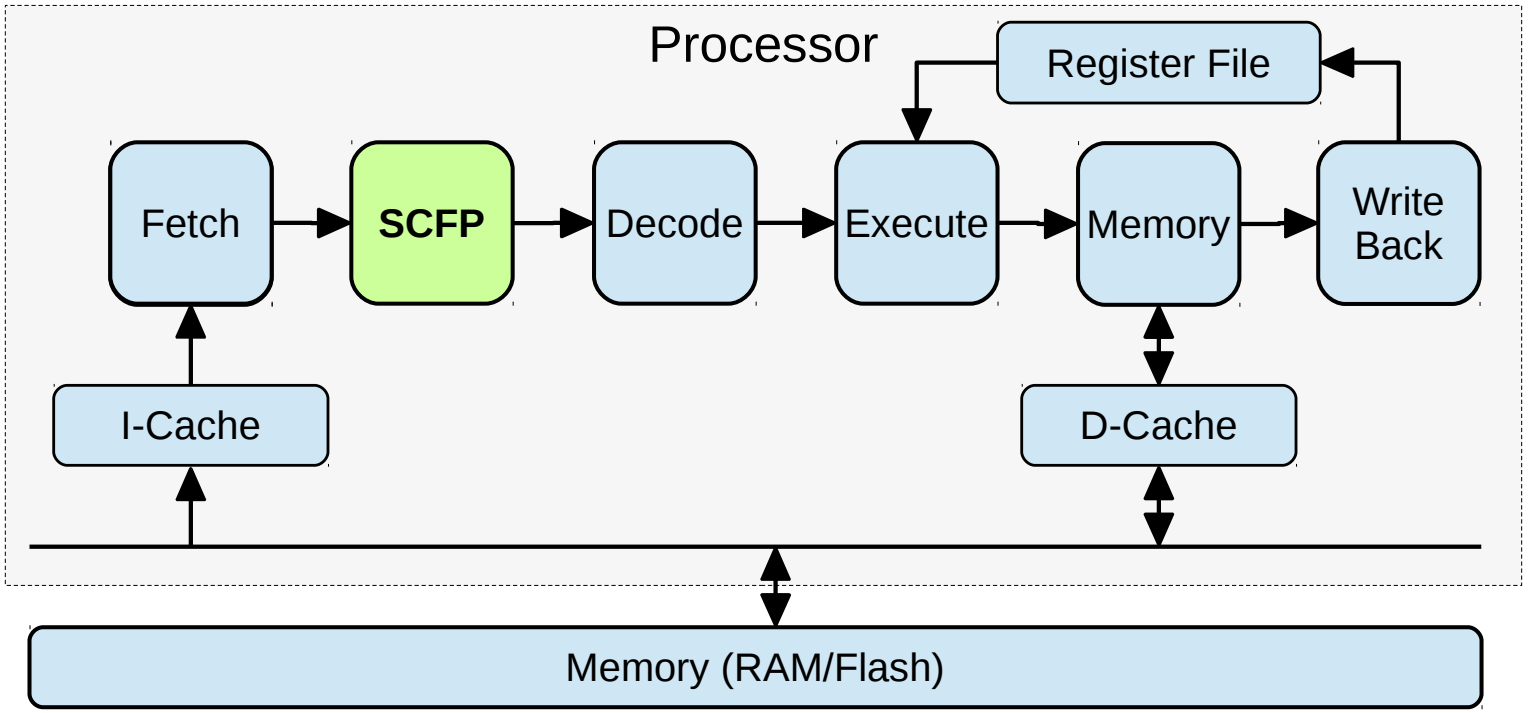


SCFP Concept

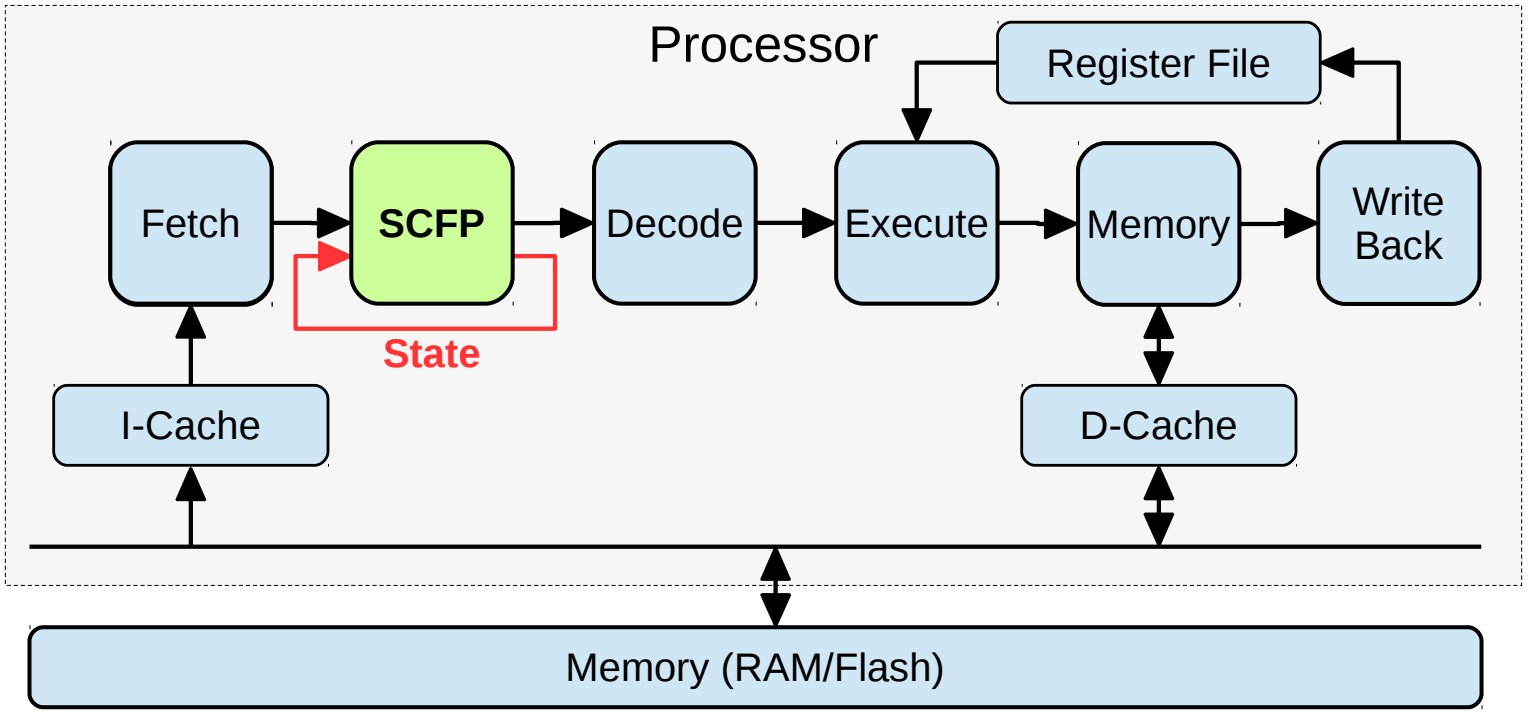
High-Level Concept



High-Level Concept



High-Level Concept



Decryption/Execution Example


```
strcmp

            : ec d0 ee 97
            : 28 ce 77 80
            : 75 41 64 b1

            : 4b f4 51 75
            : d9 a6 02 ad
            : 51 7d 34 43

            : 4d 1b c0 0f
            : a3 0f 21 3e
```

Decryption/Execution Example


```
strcmp
0x1b2a0645
             : ec d0 ee 97
             : 28 ce 77 80
             : 75 41 64 b1
```

```
             : 4b f4 51 75
             : d9 a6 02 ad
             : 51 7d 34 43
```

```
             : 4d 1b c0 0f
             : a3 0f 21 3e
```

Decryption/Execution Example


```
strcmp
0x1b2a0645
0xdd3fbcce : 03 06 05 00 : lb a2, 0(a0)
           : 28 ce 77 80
           : 75 41 64 b1
```

```
           : 4b f4 51 75
           : d9 a6 02 ad
           : 51 7d 34 43
```

```
           : 4d 1b c0 0f
           : a3 0f 21 3e
```

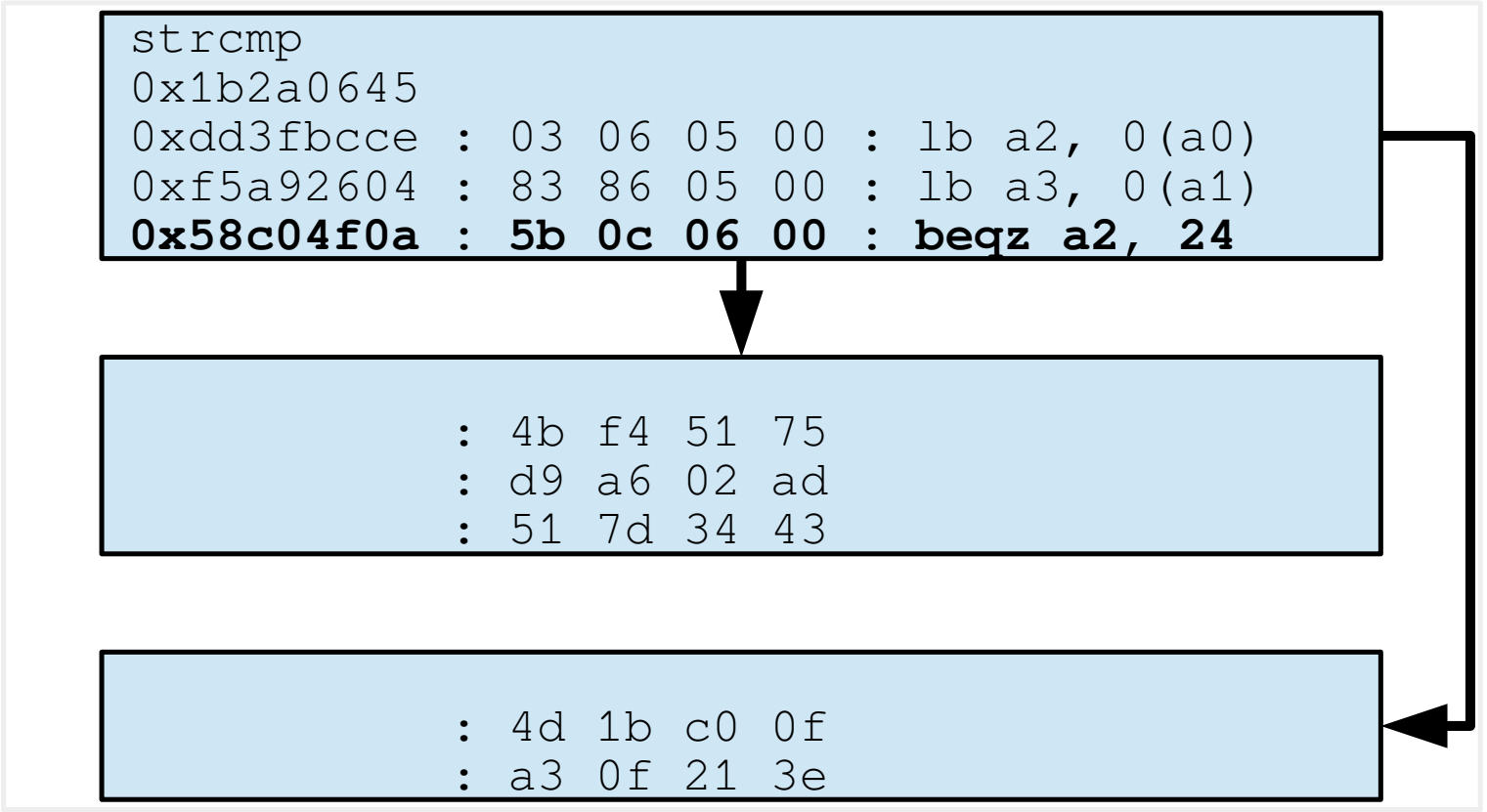
Decryption/Execution Example


```
strcmp
0x1b2a0645
0xdd3fbcce : 03 06 05 00 : lb a2, 0(a0)
0xf5a92604 : 83 86 05 00 : lb a3, 0(a1)
           : 75 41 64 b1
```

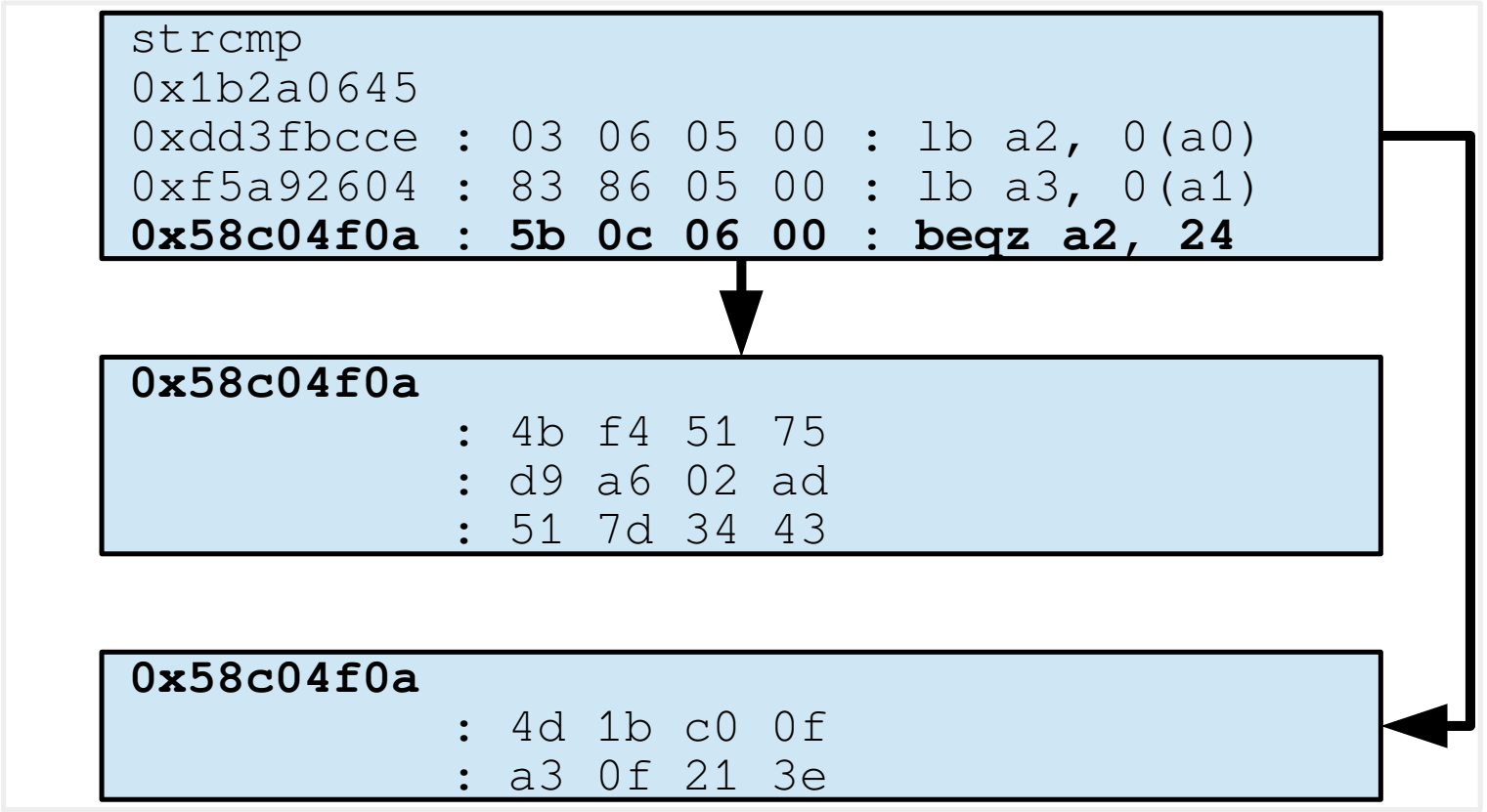
```
           : 4b f4 51 75
           : d9 a6 02 ad
           : 51 7d 34 43
```

```
           : 4d 1b c0 0f
           : a3 0f 21 3e
```

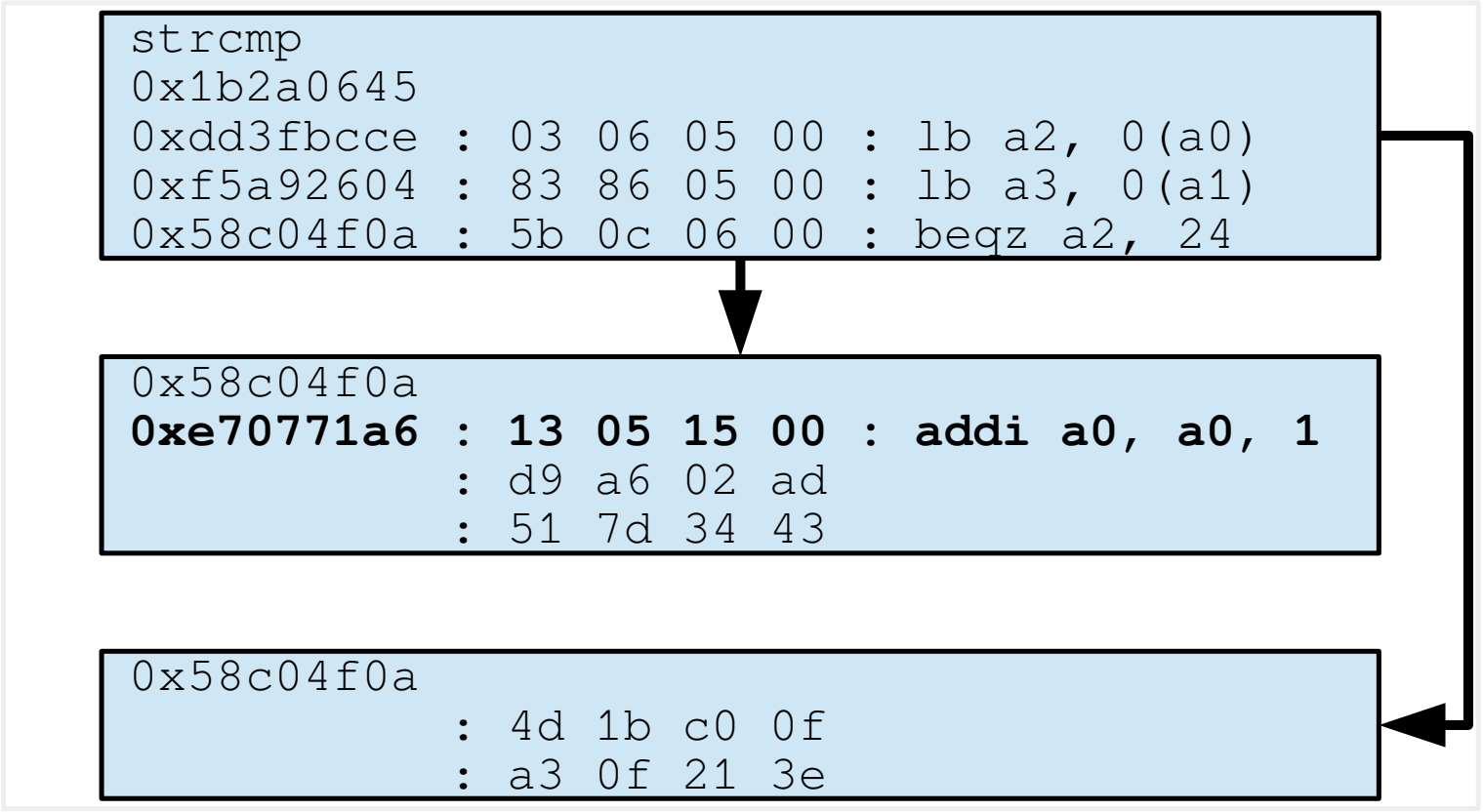
Decryption/Execution Example



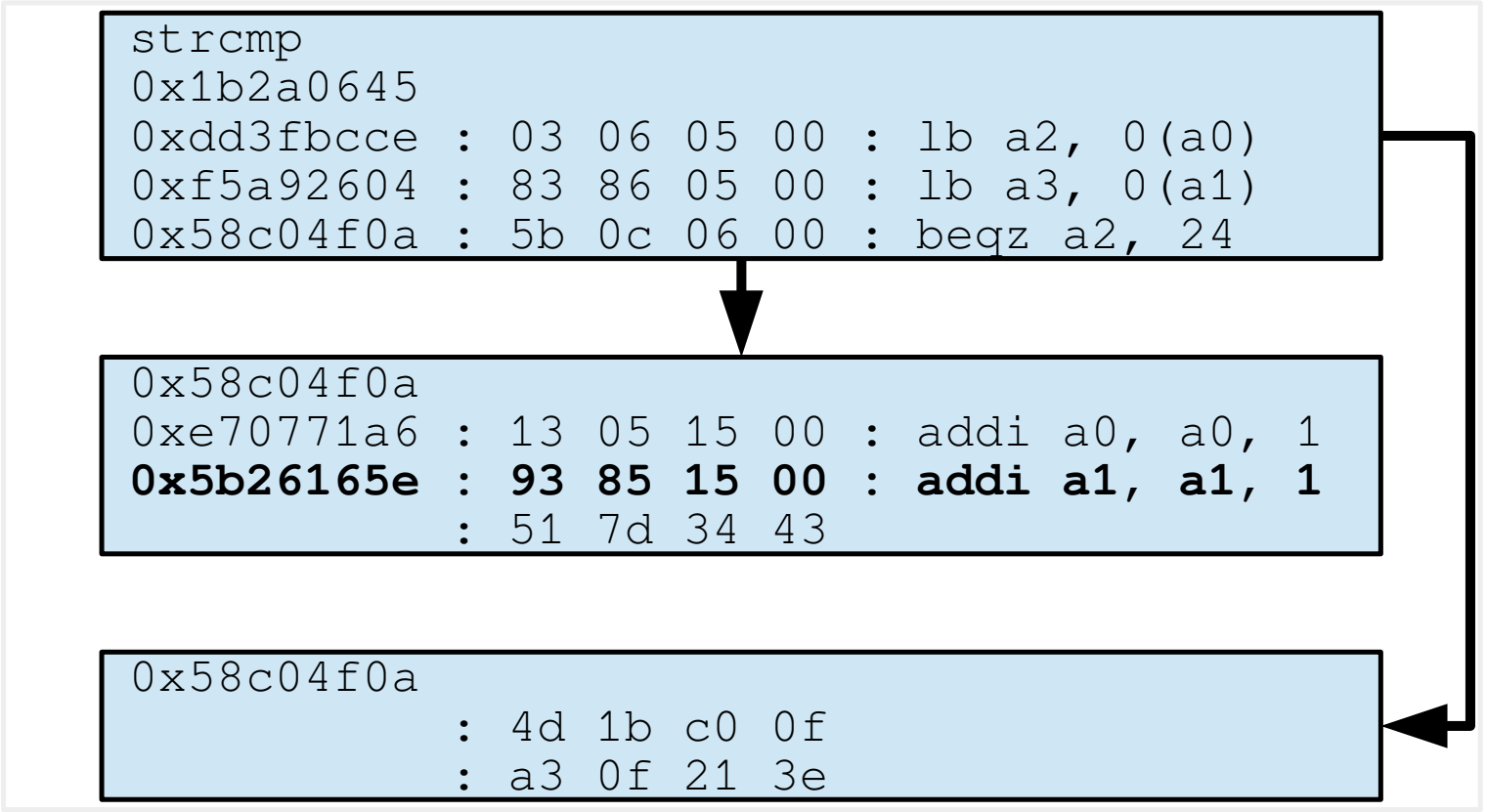
Decryption/Execution Example



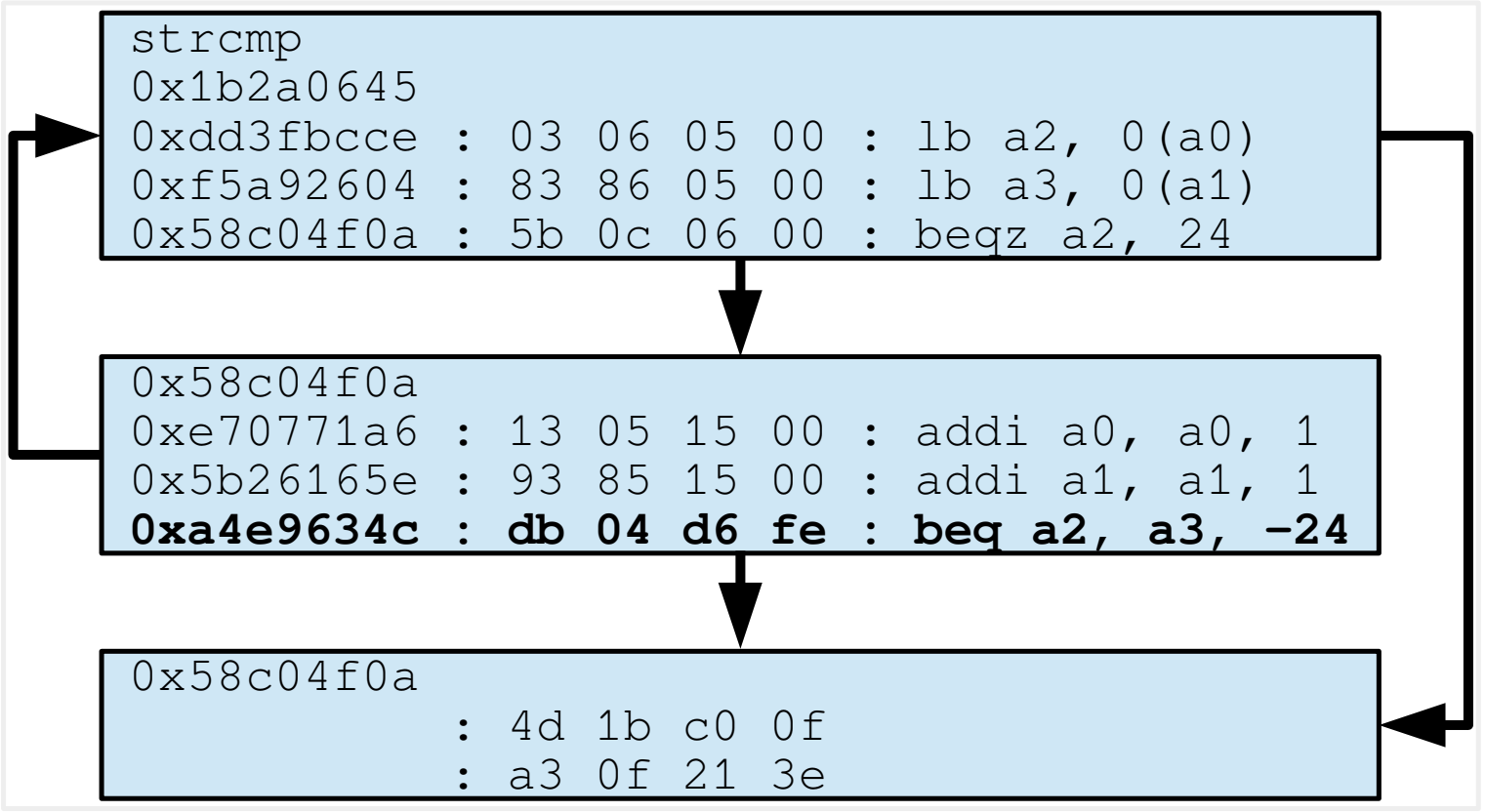
Decryption/Execution Example



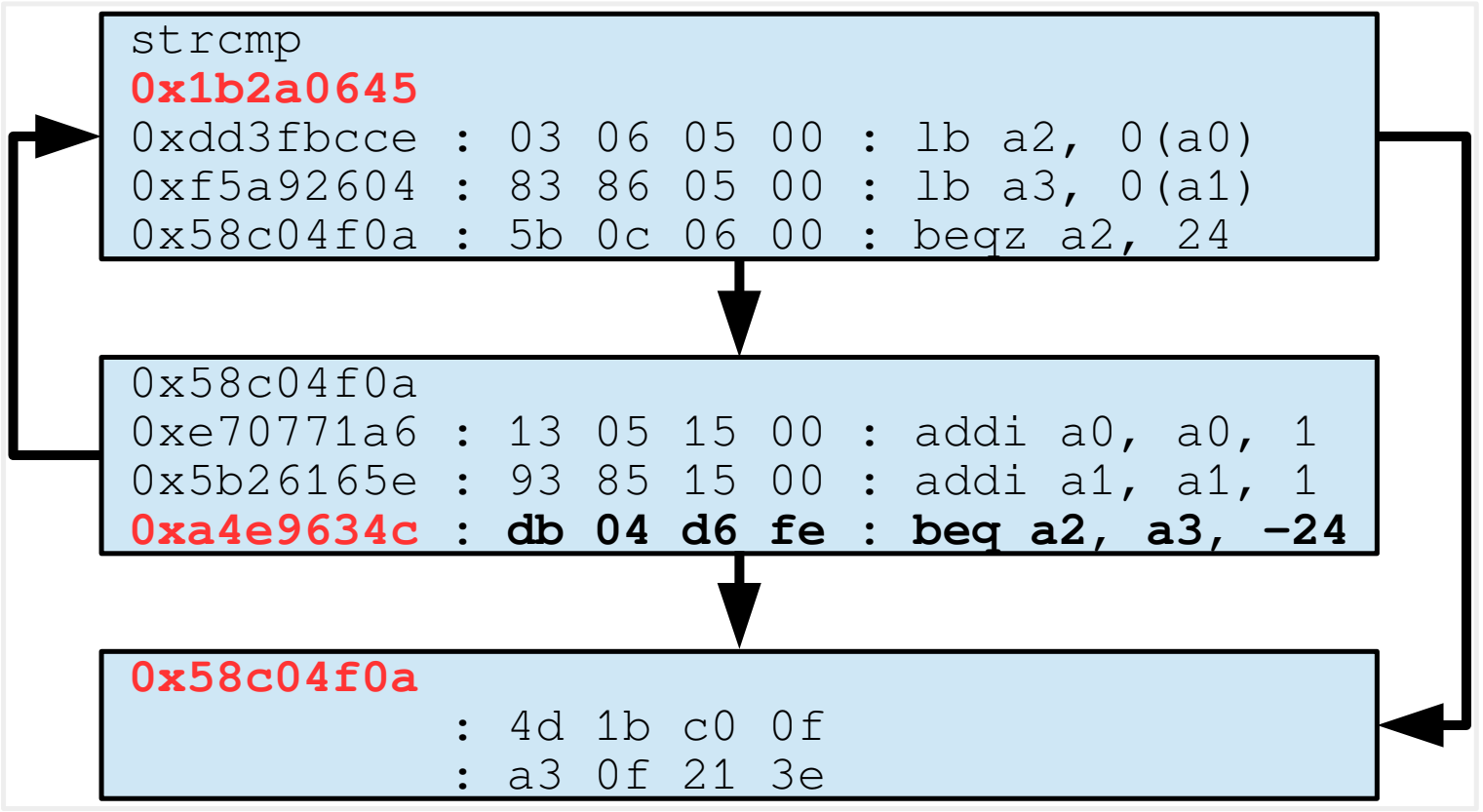
Decryption/Execution Example



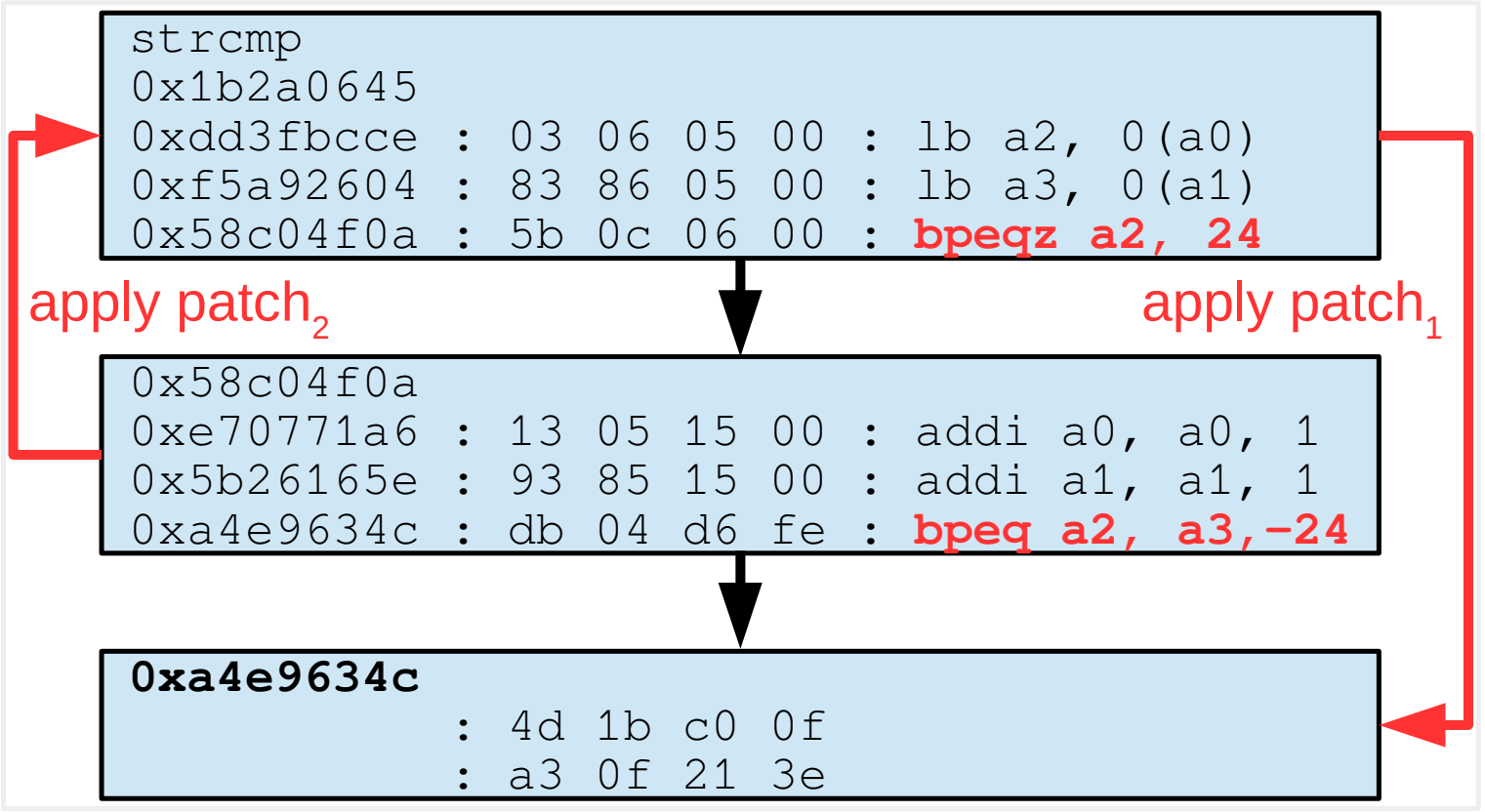
Decryption/Execution Example



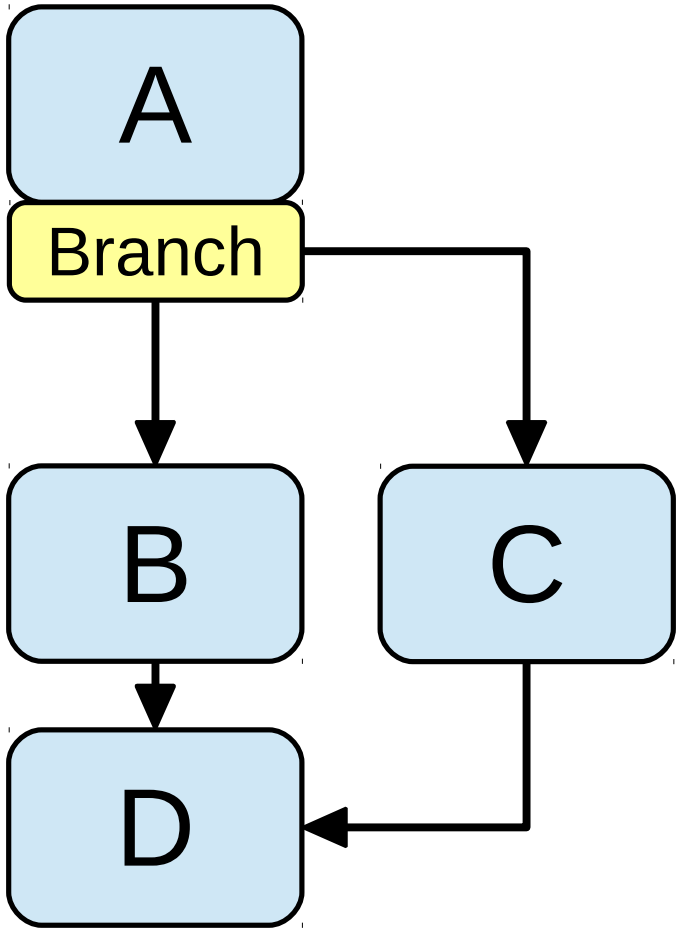
Decryption/Execution Example



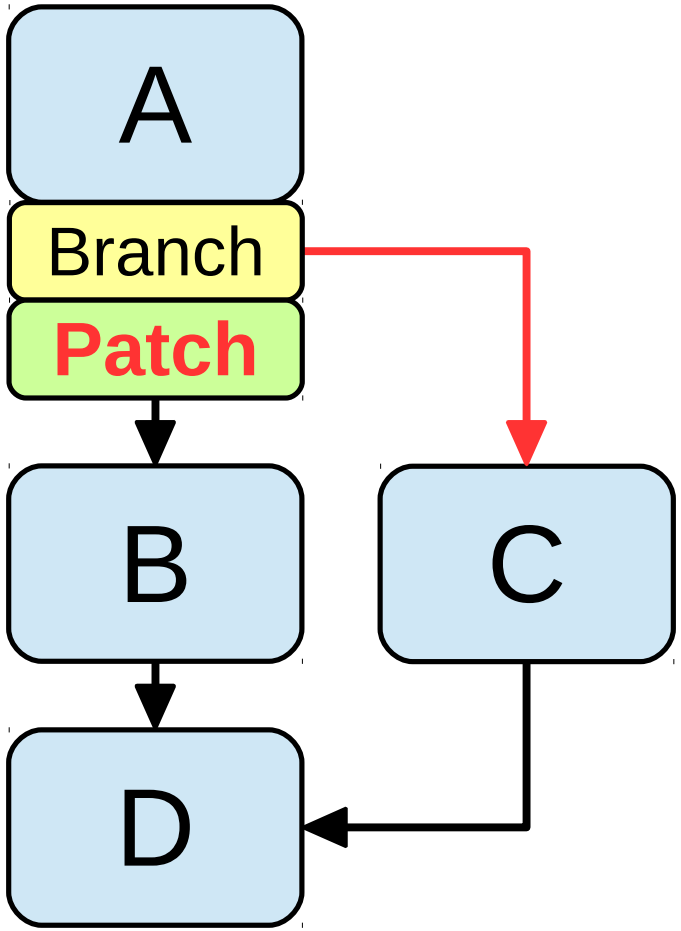
Decryption/Execution Example



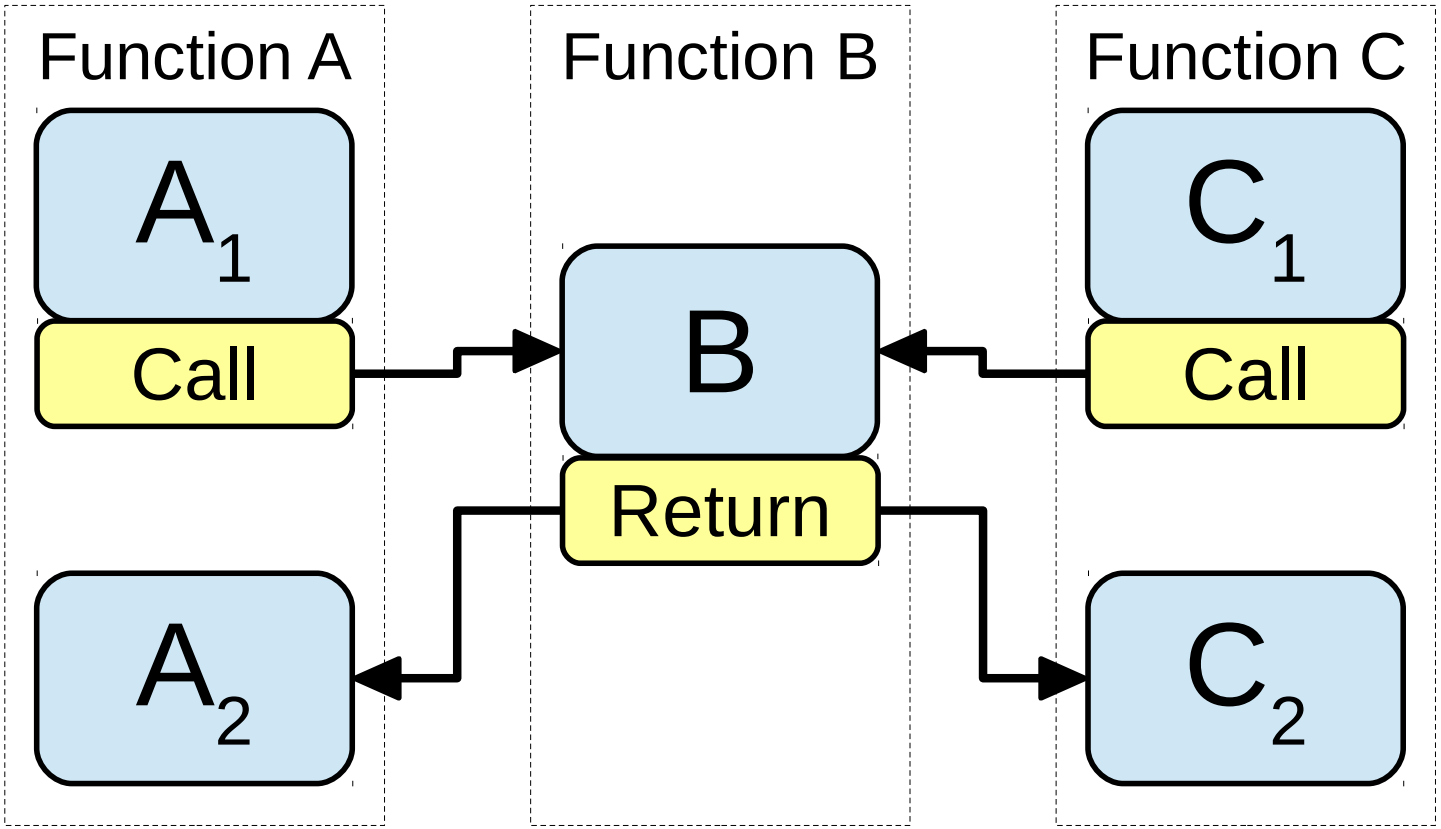
Patching Branches



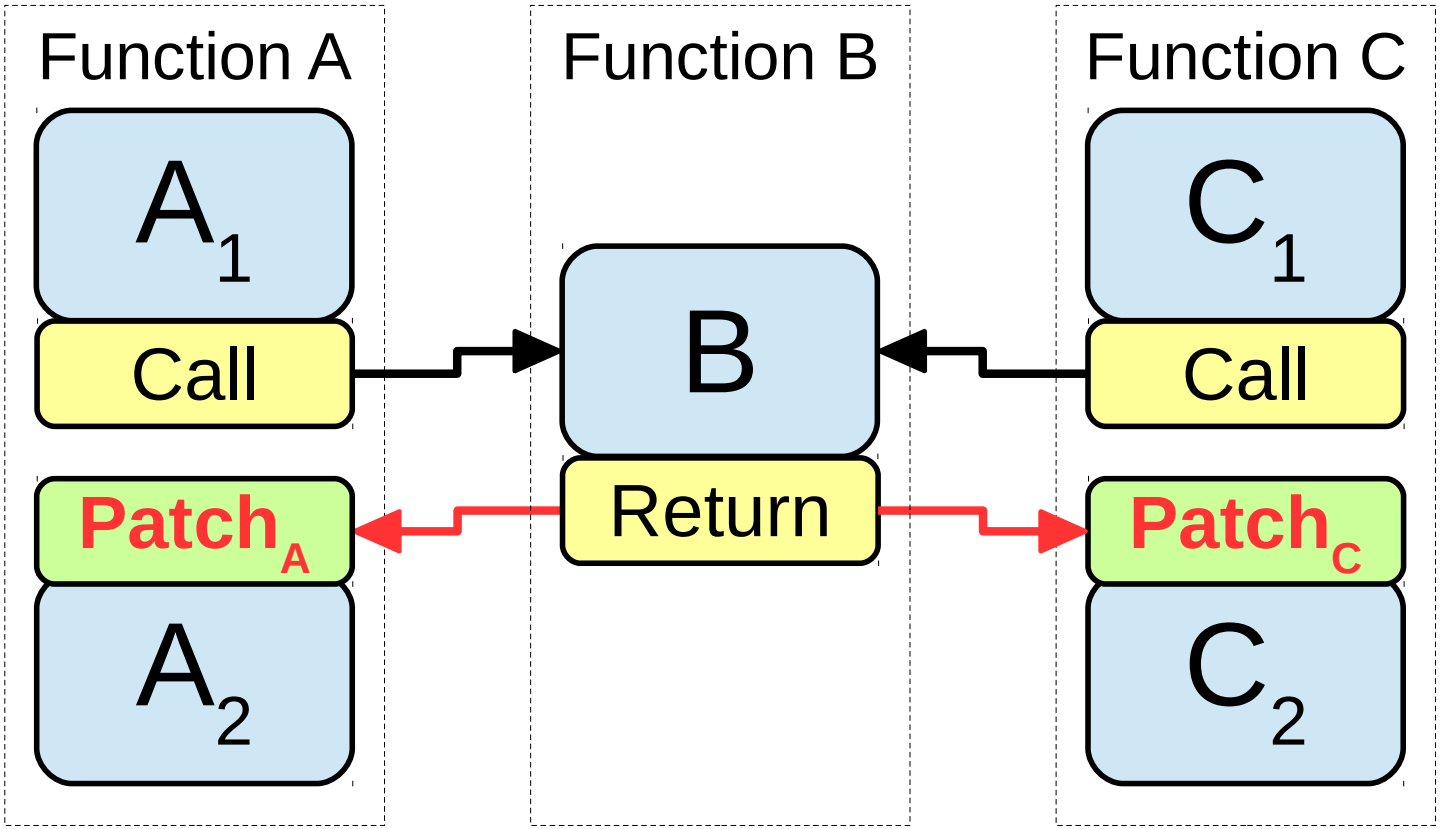
Patching Branches



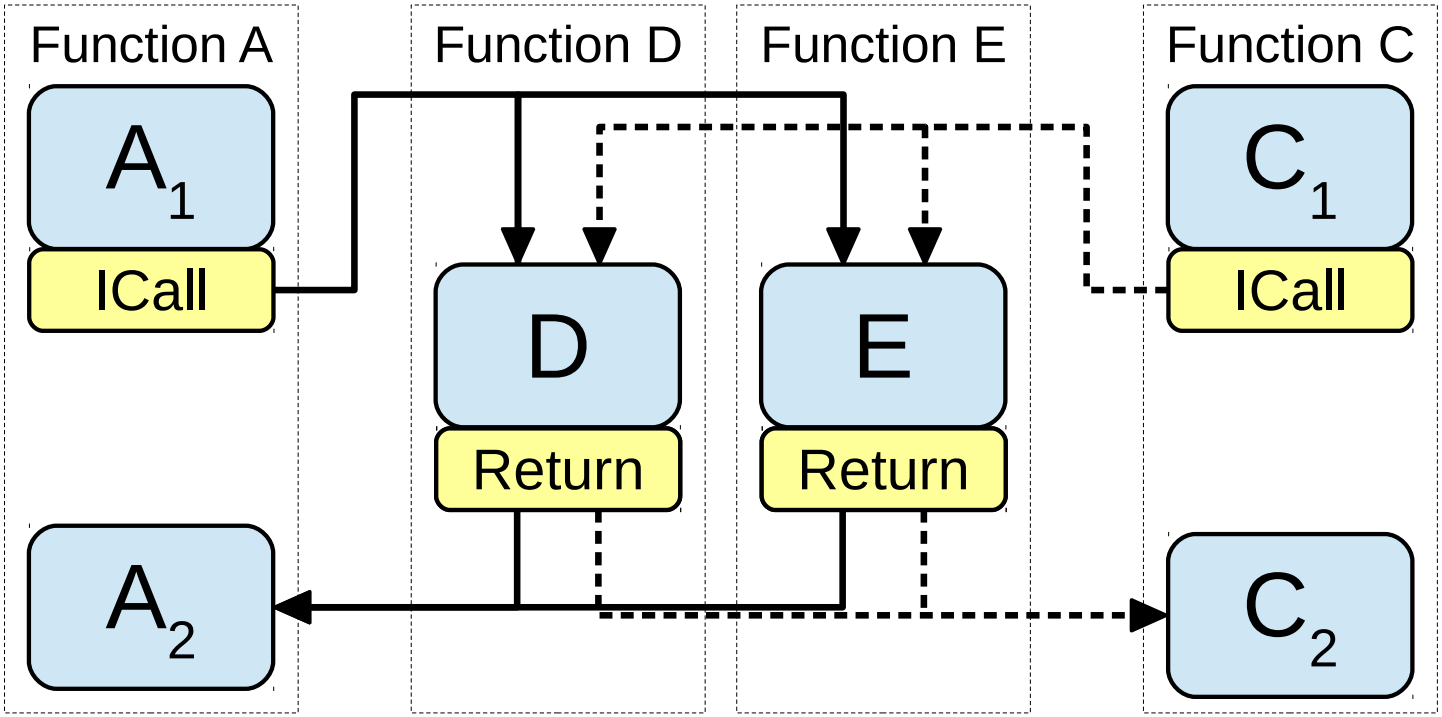
Patching Direct Function Calls



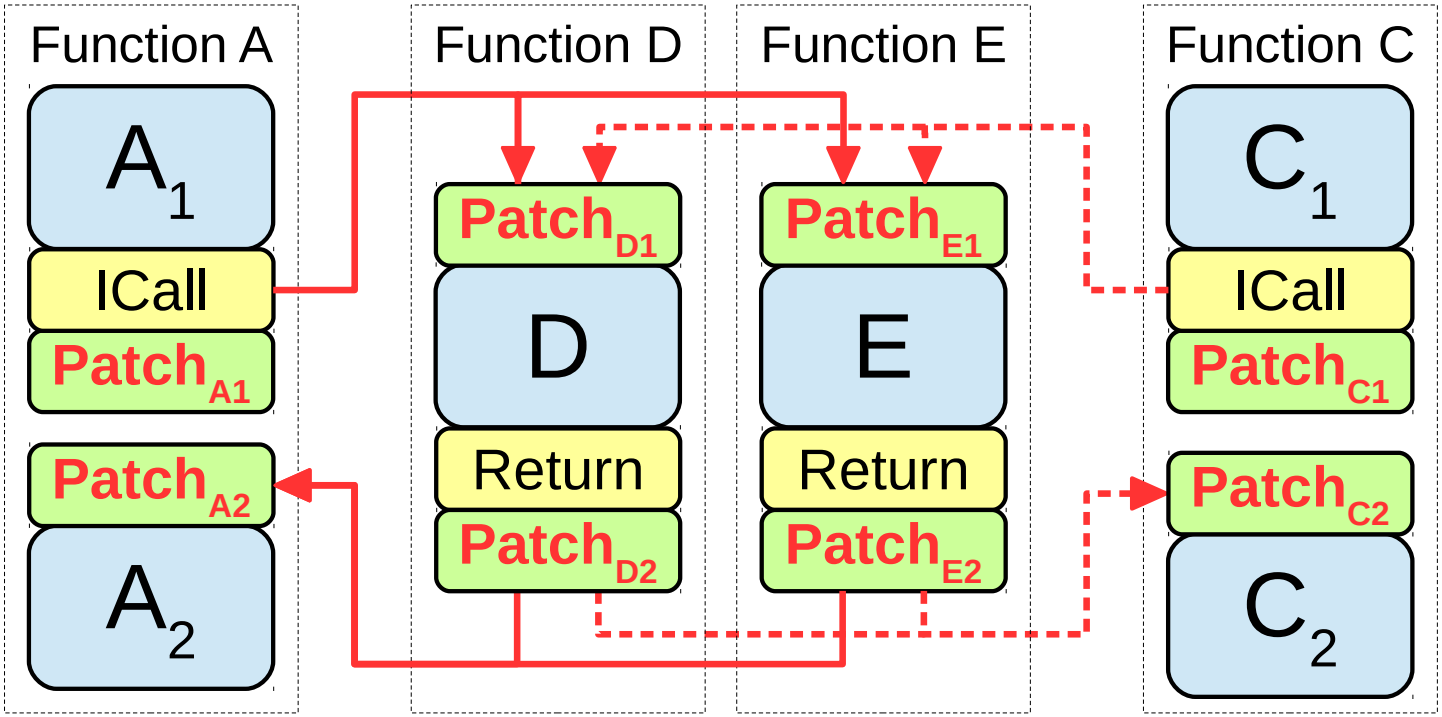
Patching Direct Function Calls



Patching Indirect Function Calls

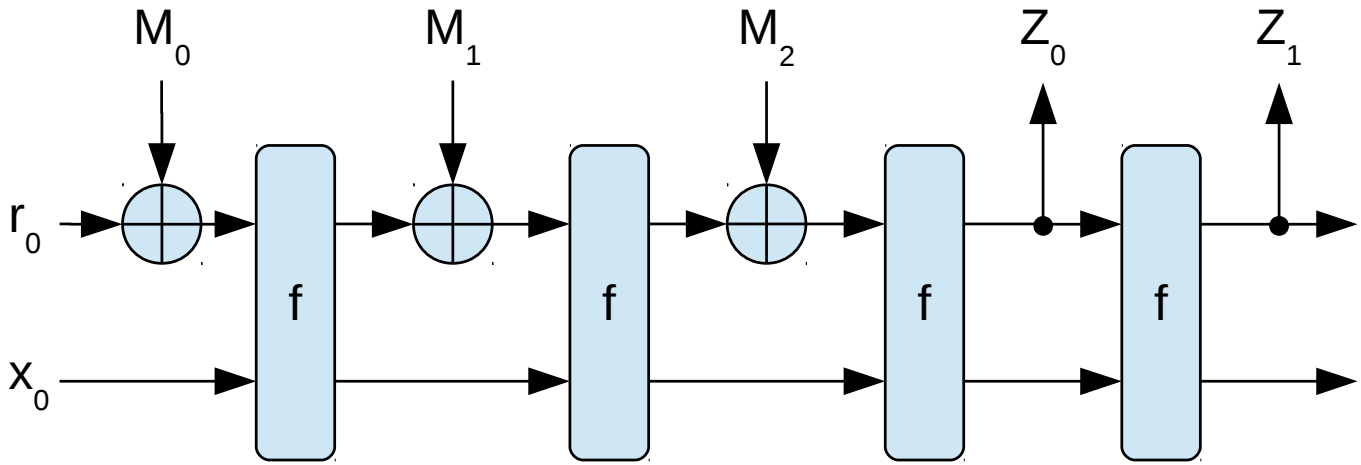


Patching Indirect Function Calls

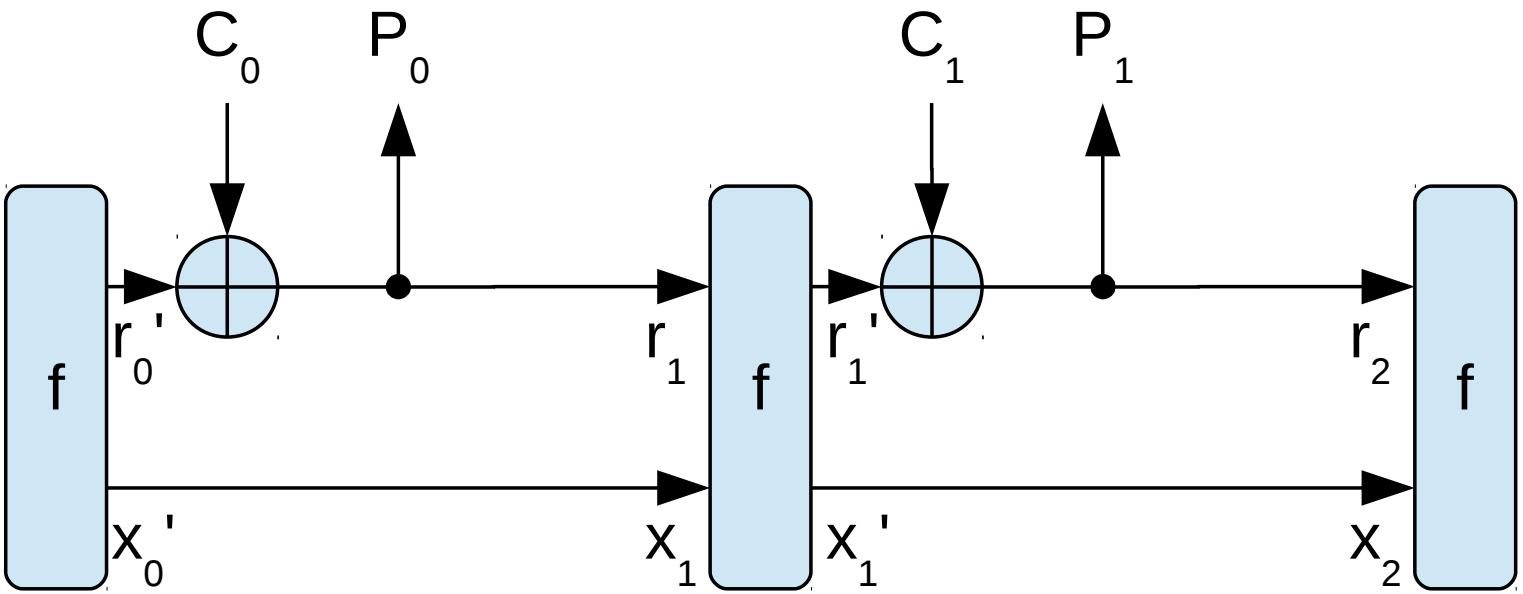


Sponge Constructions for SCFP

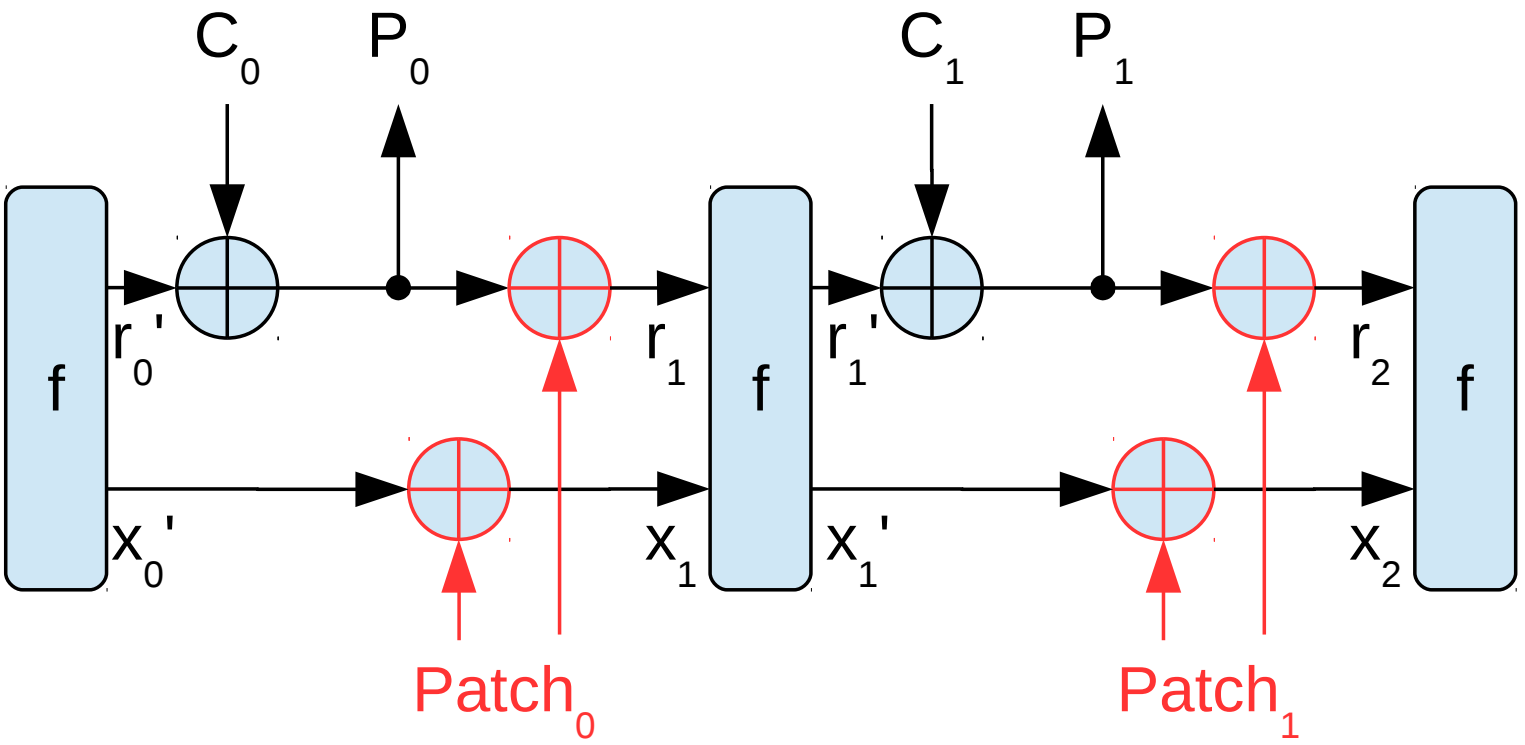
- Mode of operation developed by Bertoni et al. [Ber+07]
- Based on a fixed-length permutation + padding rule
- Keccak standardized as SHA-3 and SHAKE
- Hash functions, XOFs, MACs, stream and AEAD ciphers



- Original construction by Bertoni et al. [Ber+11]



- Original construction by Bertoni et al. [Ber+11]

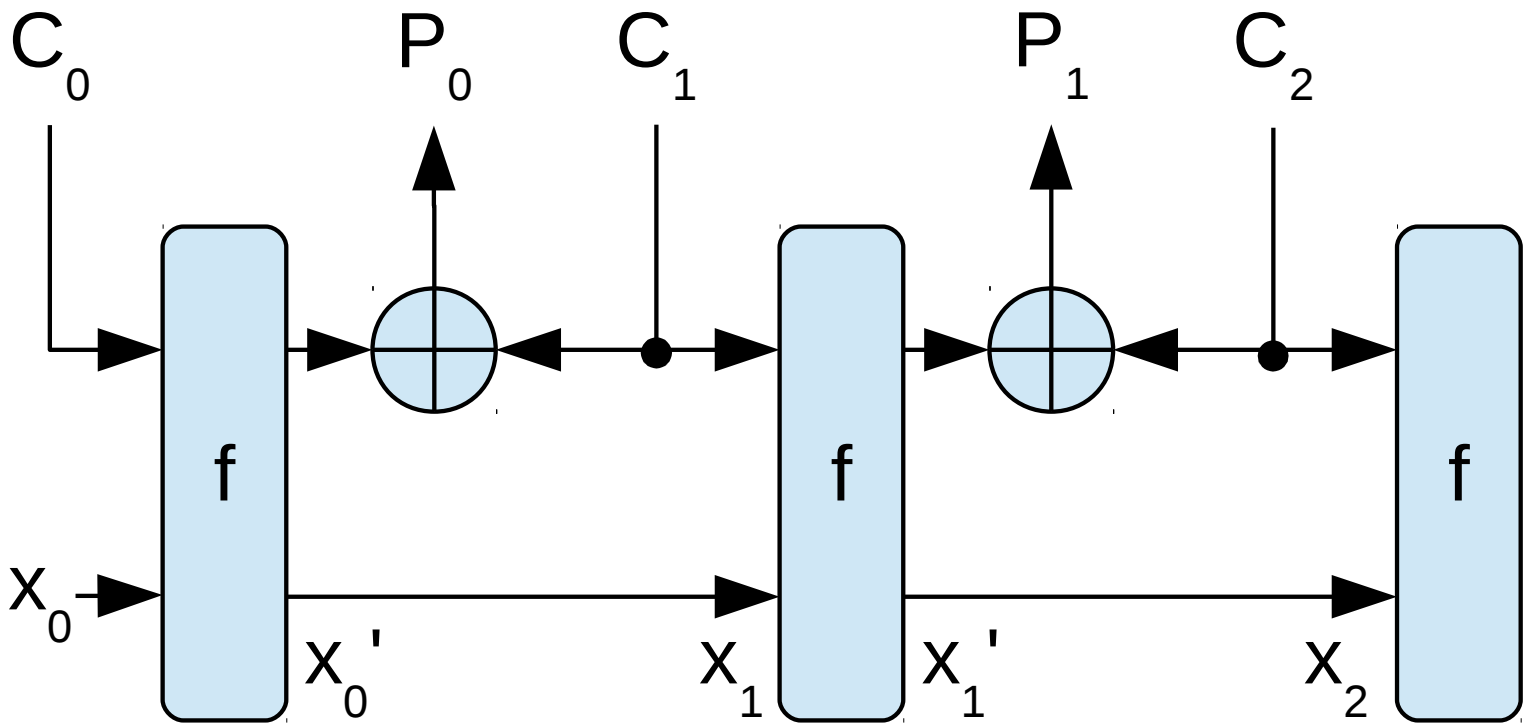


SpongeWrap-like Decryption Mode

- Unmodified duplex construction when Patch = 0
- Full state has to be patched for branches
- Patch can be considered as associated data (AD)
- Absorbing AD into the capacity is allowed for keyed sponges [MRV15; SY15]
- Errors propagate directly from ciphertext to plaintext

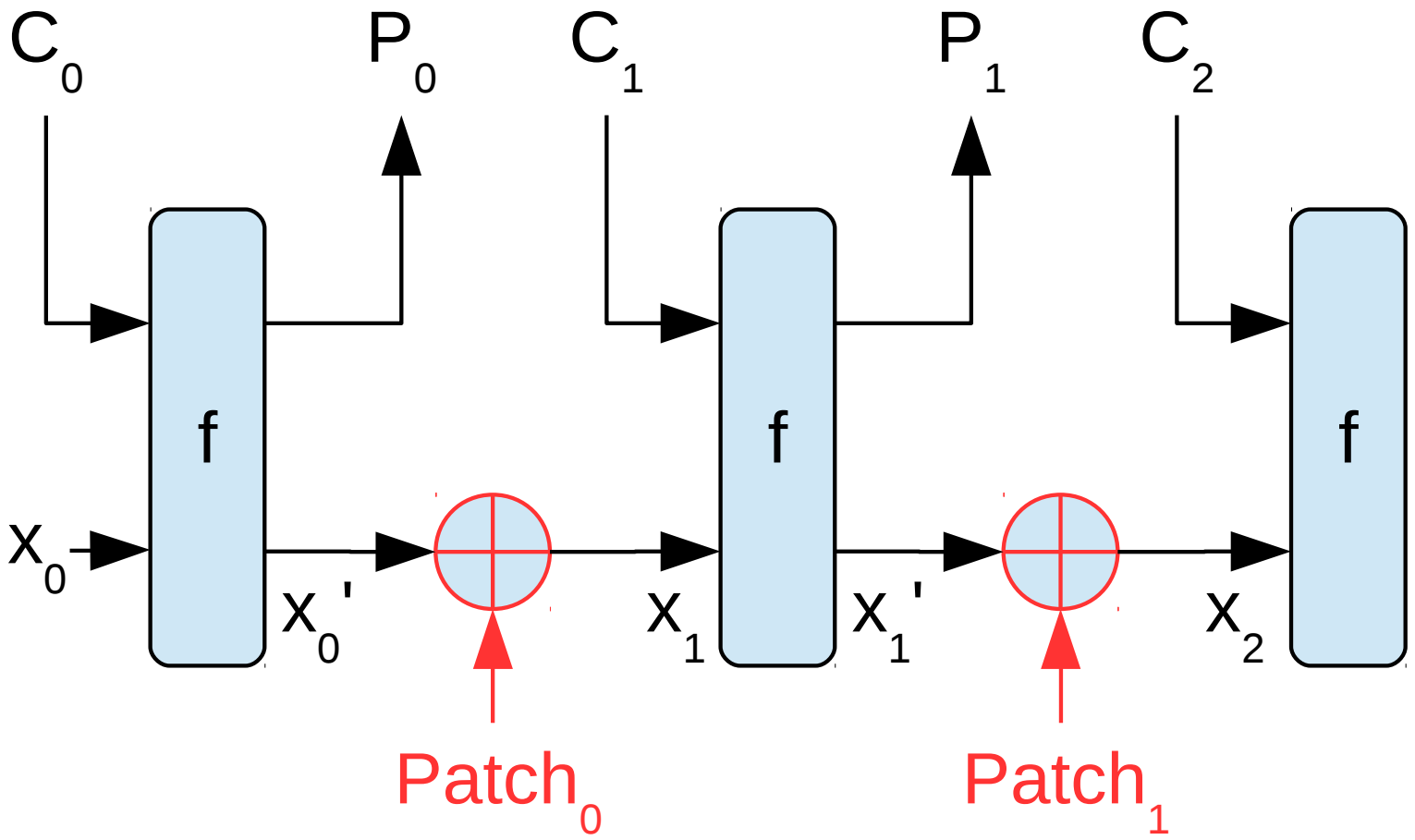
APE Decryption Mode

- Original construction by Andreeva et al. [And+14]



APE-like Decryption Mode

- Original construction by Andreeva et al. [And+14]



APE-like Decryption Mode

- Removes direct dependency of P_i on C_{i+1}
- Permutation between P_i and C_i
- Only the capacity has to be patched
- Non-inverse free regarding the permutation
- Encryption has to be performed backward

Instantiations and Security

- Extremely flexible due to the sponge-based design
- Security as well as overhead scales with the sponge
- Three different instantiations of the APE-like mode
- 32-bit instruction encodings

Instantiations and Security

- Extremely flexible due to the sponge-based design
- Security as well as overhead scales with the sponge
- Three different instantiations of the APE-like mode
- 32-bit instruction encodings

Name	Permutation	Parameters [bit]				
				Crypt.	Attack Compl.	
		x	s_p	Security	CIA	CRA
AEE	Keccak-p[200,12]	168	—	84	168	168

Instantiations and Security

- Extremely flexible due to the sponge-based design
- Security as well as overhead scales with the sponge
- Three different instantiations of the APE-like mode
- 32-bit instruction encodings

Name	Permutation	Parameters [bit]				
				Crypt.	Attack Compl.	
		x	s_p	Security	CIA	CRA
AEE	Keccak-p[200,12]	168	—	84	168	168
IE	Keccak-p[50,12]	16	—	8	16	16

Instantiations and Security

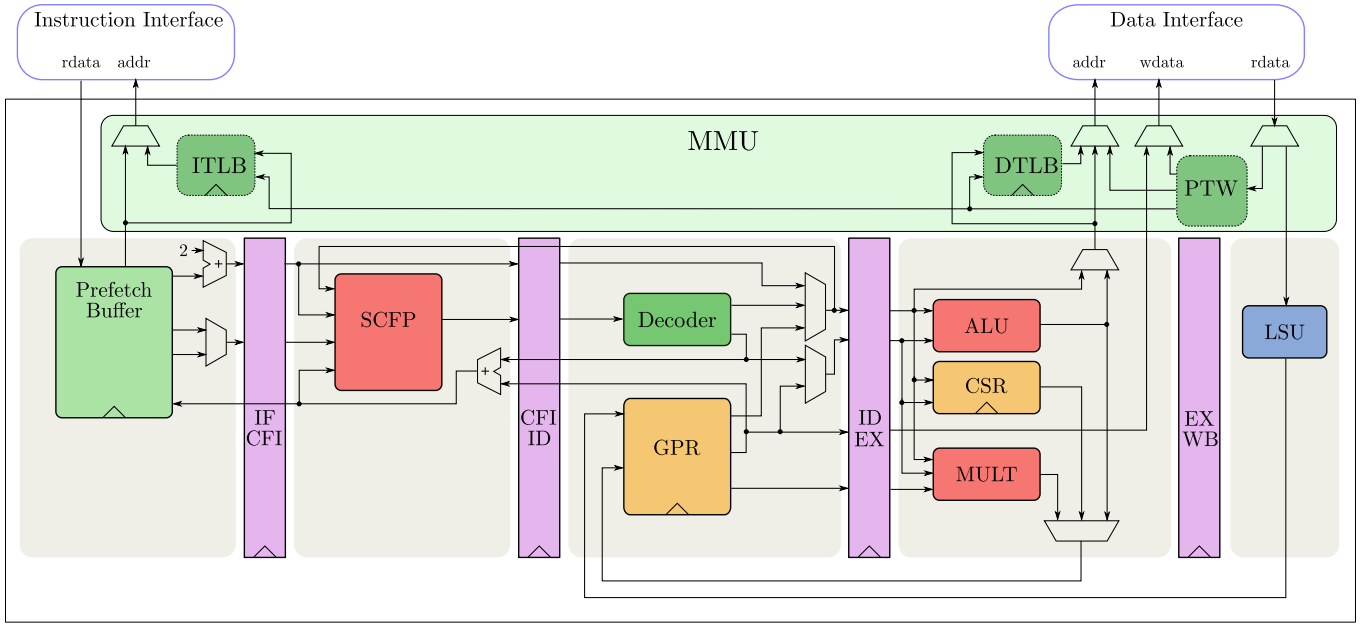
- Extremely flexible due to the sponge-based design
- Security as well as overhead scales with the sponge
- Three different instantiations of the APE-like mode
- 32-bit instruction encodings

Name	Permutation	Parameters [bit]				
				Crypt.	Attack Compl.	
		x	s_p	Security	CIA	CRA
AEE	Keccak-p[200,12]	168	—	84	168	168
IE	Keccak-p[50,12]	16	—	8	16	16
AEE-Light	PRINCE	32	96	16	128	32

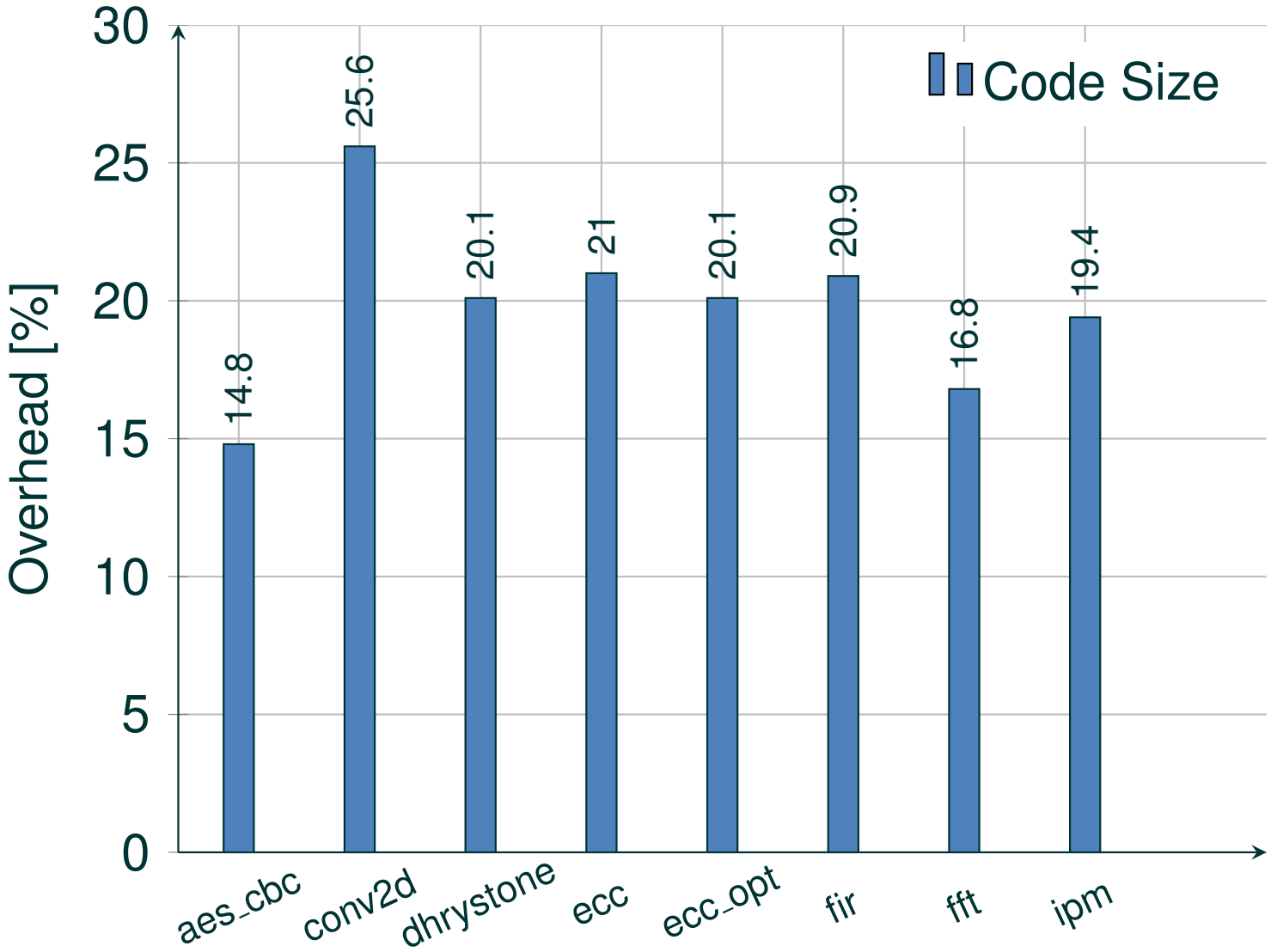
Evaluation and Summary

Implementation

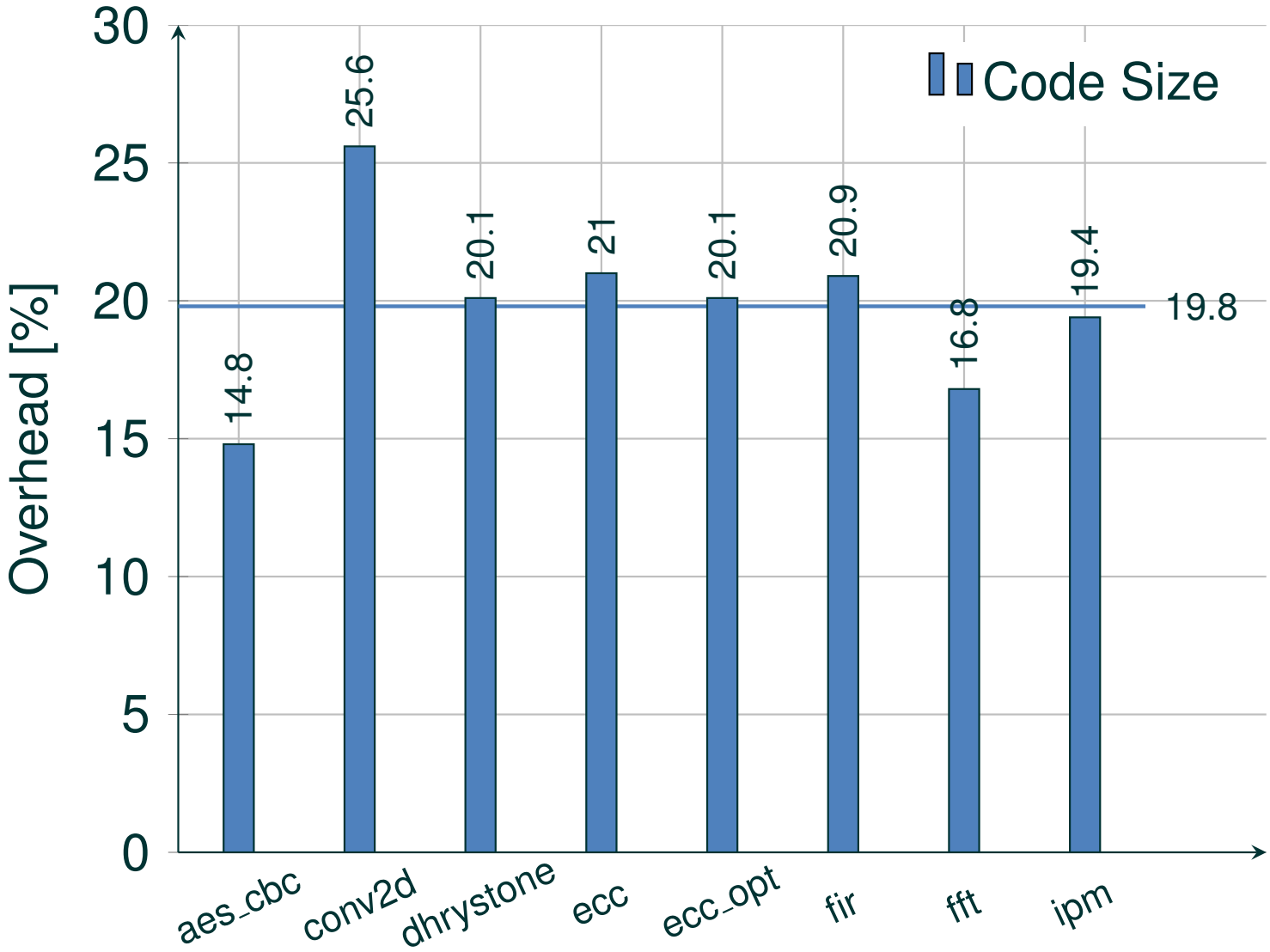
- Added SCFP the RI5CY [ETH17] RISC-V processor core
- 4 or 5 pipeline stages
- AEE-Light with PRINCE in APE-like mode
- ∼30 kGE of area for SCFP at 100 MHz in UMC65
- Benchmarks using a very simple software toolchain



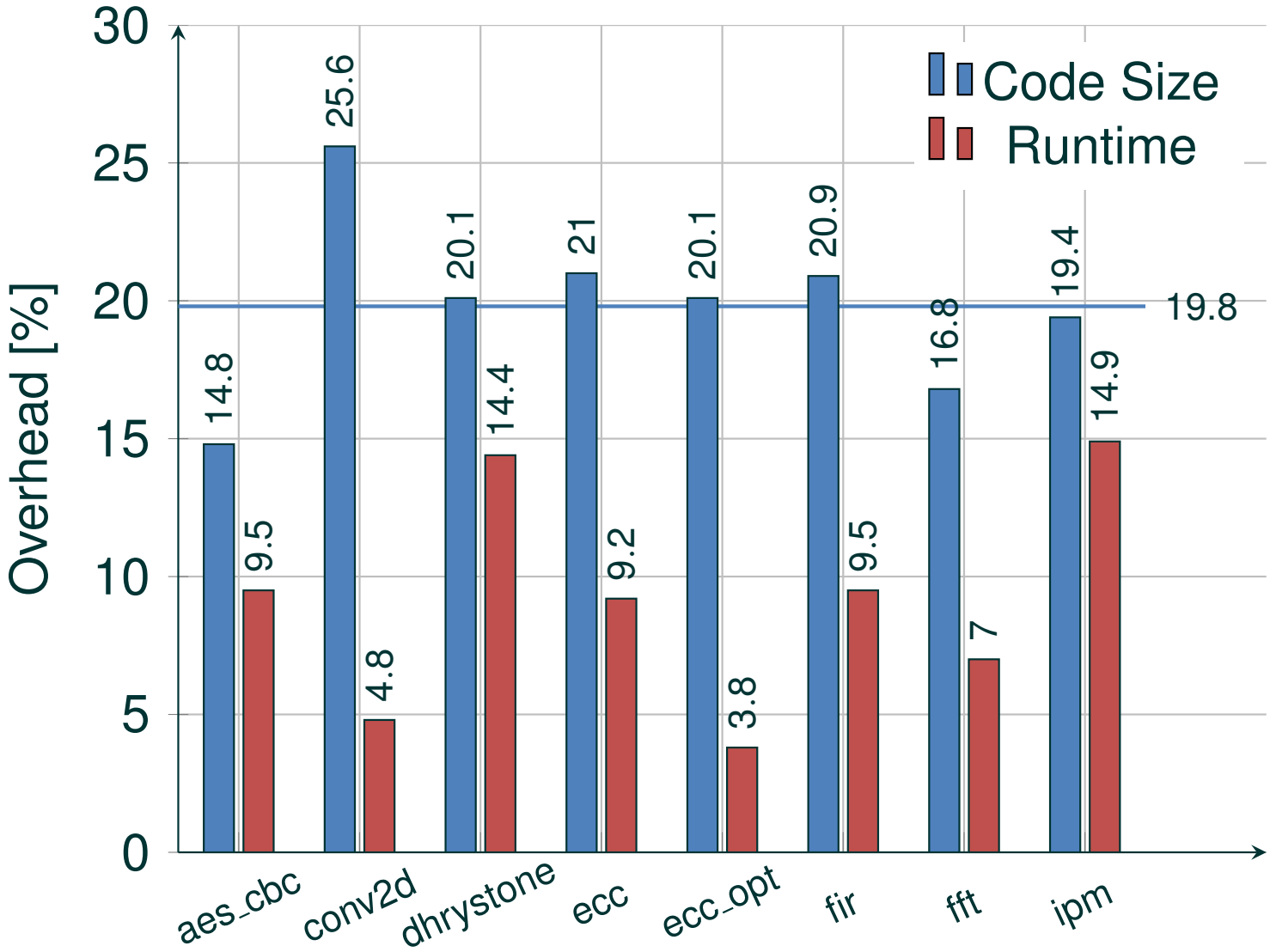
Overhead Evaluation



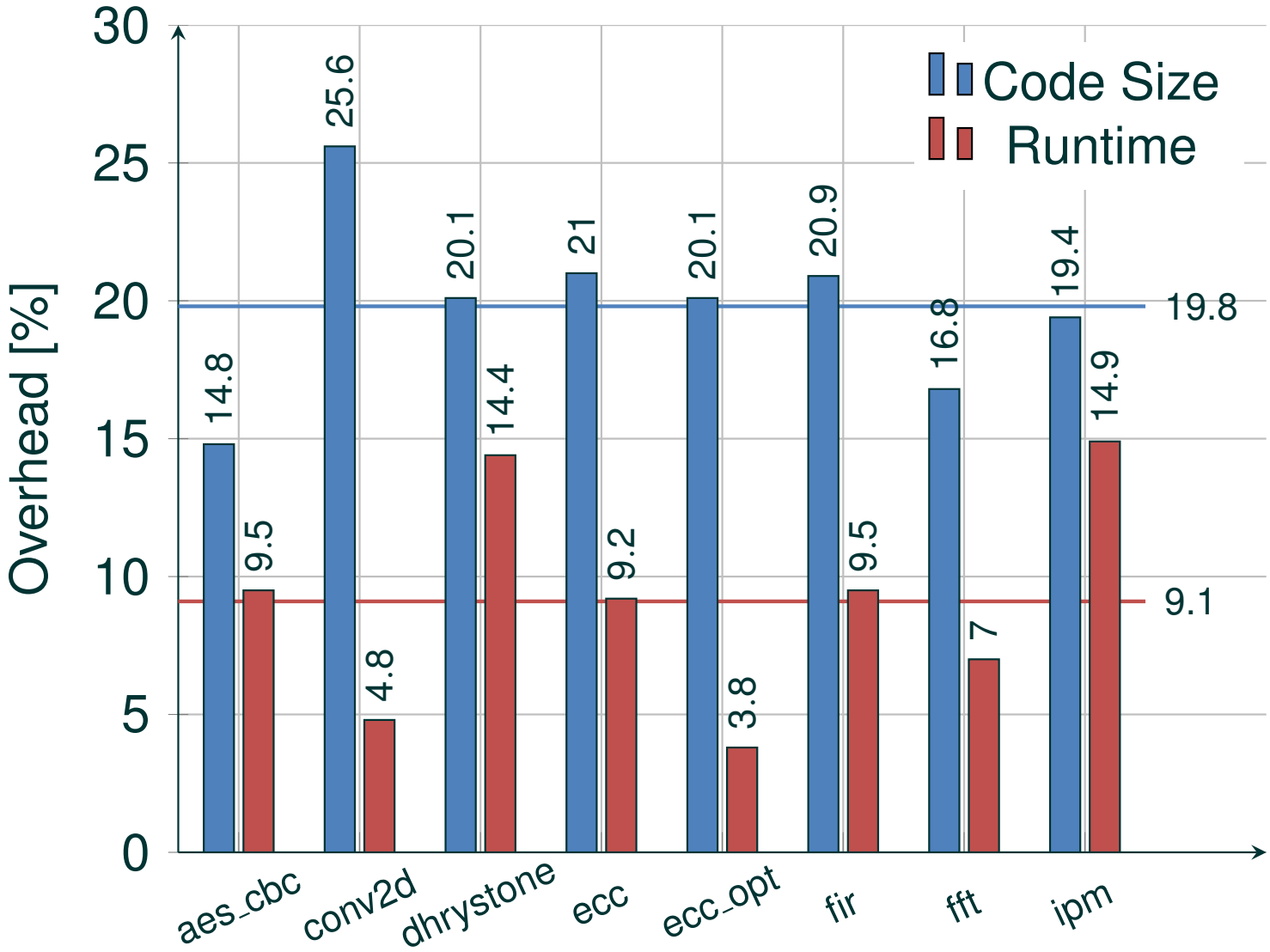
Overhead Evaluation



Overhead Evaluation



Overhead Evaluation



Summary

- Physical attacks have to be considered
- Sponge-based Control-Flow Protection
  - Hardware supported CFI scheme
  - Encrypts the instruction stream with instruction granularity
- Presented and analyzed two suitable sponge constructions
- Discussed three SCFP instantiations (IE, AEE, AEE-Light)
- Implemented AEE-Light into a RISC-V processor
  - 9.1 % runtime overhead
  - 19.8 % code size overhead

Sponge-Based Control-Flow Protection for IoT Devices

Werner, Unterluggauer, Schaffenrath, Mangard

25th April 2018, London

Graz University of Technology

References

Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. “APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography”. In: *Fast Software Encryption – FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. LNCS. Springer, 2014, pp. 168–186. DOI: 10.1007/978-3-662-46706-0_9.

Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. “The Sorcerer’s Apprentice Guide to Fault Attacks”. In: *IACR Cryptology ePrint Archive* 2004 (2004), p. 100. URL: http://eprint.iacr.org/2004/100.

Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. “On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)”. In: *Advances in Cryptology – EUROCRYPT 97*. Ed. by Walter Fumy. Vol. 1233. LNCS. Springer, 1997, pp. 37–51. DOI: 10.1007/3-540-69053-0_4.

Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. “Sponge functions”. In: *ECRYPT Hash Workshop*. 2007. URL: https://keccak.team/files/SpongeFunctions.pdf (visited on 01/19/2016).

Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. “Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications”. In: *Selected Areas in Cryptography – SAC 2011*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. LNCS. Springer, 2011, pp. 320–337. DOI: 10.1007/978-3-642-28496-0_19.

References iii

Ruan de Clercq, Ronald De Keulenaer, Bart Coppens, Bohan Yang, Pieter Maene, Koen De Bosschere, Bart Preneel, Bjorn De Sutter, and Ingrid Verbauwhede. “SOFIA: Software and control flow integrity architecture”. In: *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*. Ed. by Luca Fanucci and Jürgen Teich. IEEE, 2016, pp. 1172–1177. URL: http://ieeexplore.ieee.org/document/7459489/.

ETH Zurich. *RI5CY Source Repository*. 2017. URL: https://github.com/pulp-platform/riscv (visited on 03/26/2018).

Bart Mennink, Reza Reyhanitabar, and Damian Vizár. “Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption”. In: *Advances in Cryptology – ASIACRYPT 2015*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, 2015, pp. 465–489. DOI: 10.1007/978-3-662-48800-3_19.

Yu Sasaki and Kan Yasuda. “How to Incorporate Associated Data in Sponge-Based Authenticated Encryption”. In: *Topics in Cryptology – CT-RSA 2015*. Ed. by Kaisa Nyberg. Vol. 9048. LNCS. Springer, 2015, pp. 353–370. DOI: 10.1007/978-3-319-16715-2_19.