A Tamper and Leakage Resilient von Neumann Architecture

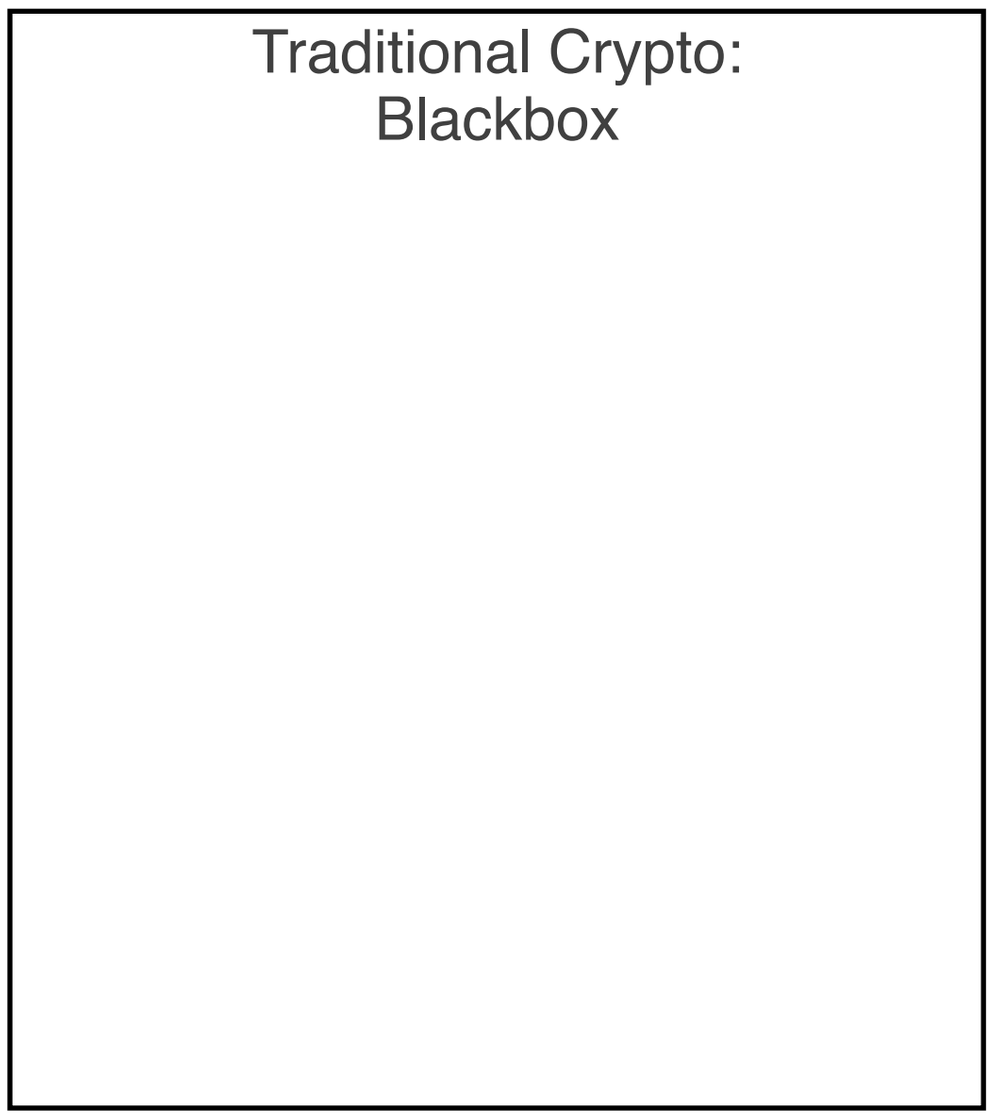
Pratyay Mukherjee
Aarhus University

joint work with

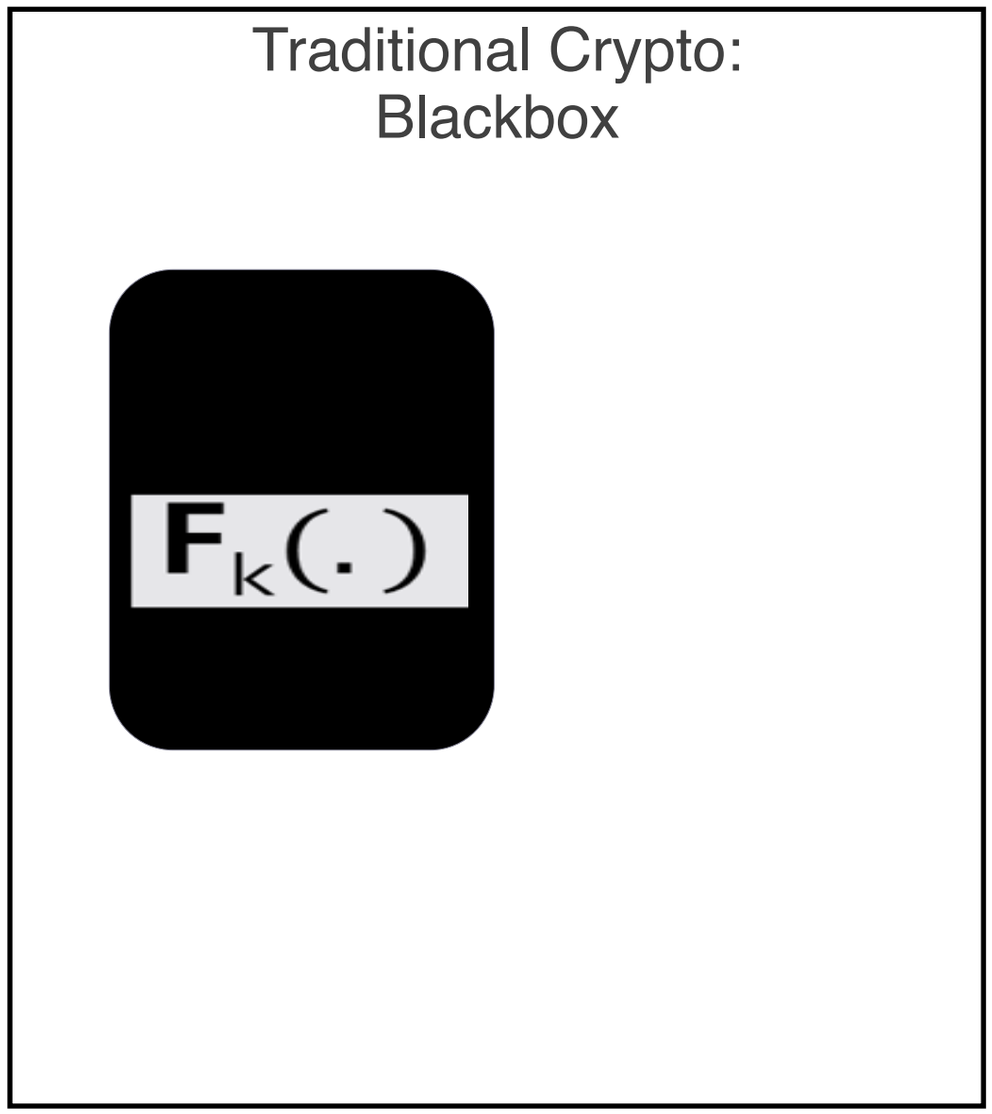
Sebastian Faust (EPFL), Jesper Buus Nielsen (Aarhus),
Daniele Venturi (La Sapienza, Rome)

Physical Attack on implementations

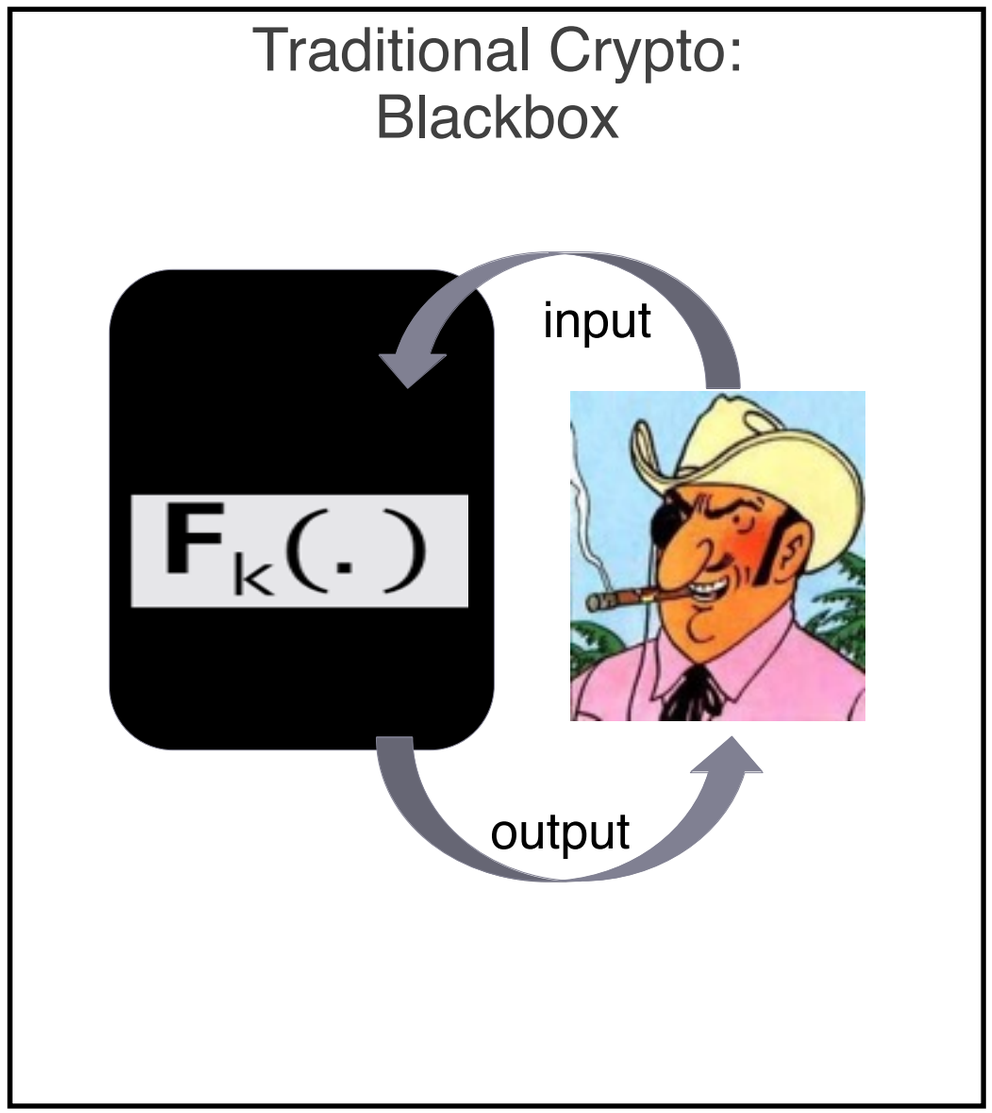
Physical Attack on implementations



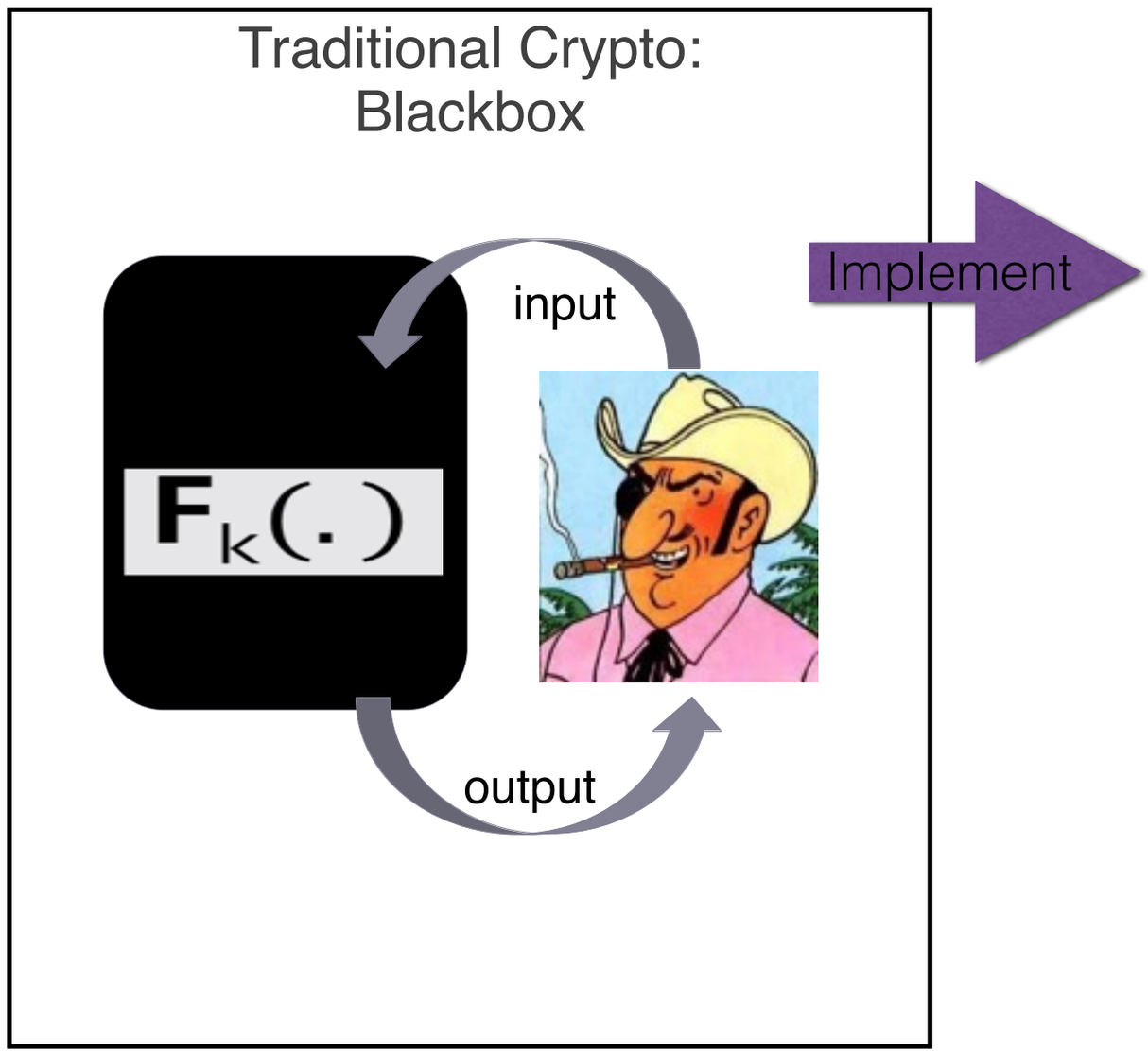
Physical Attack on implementations



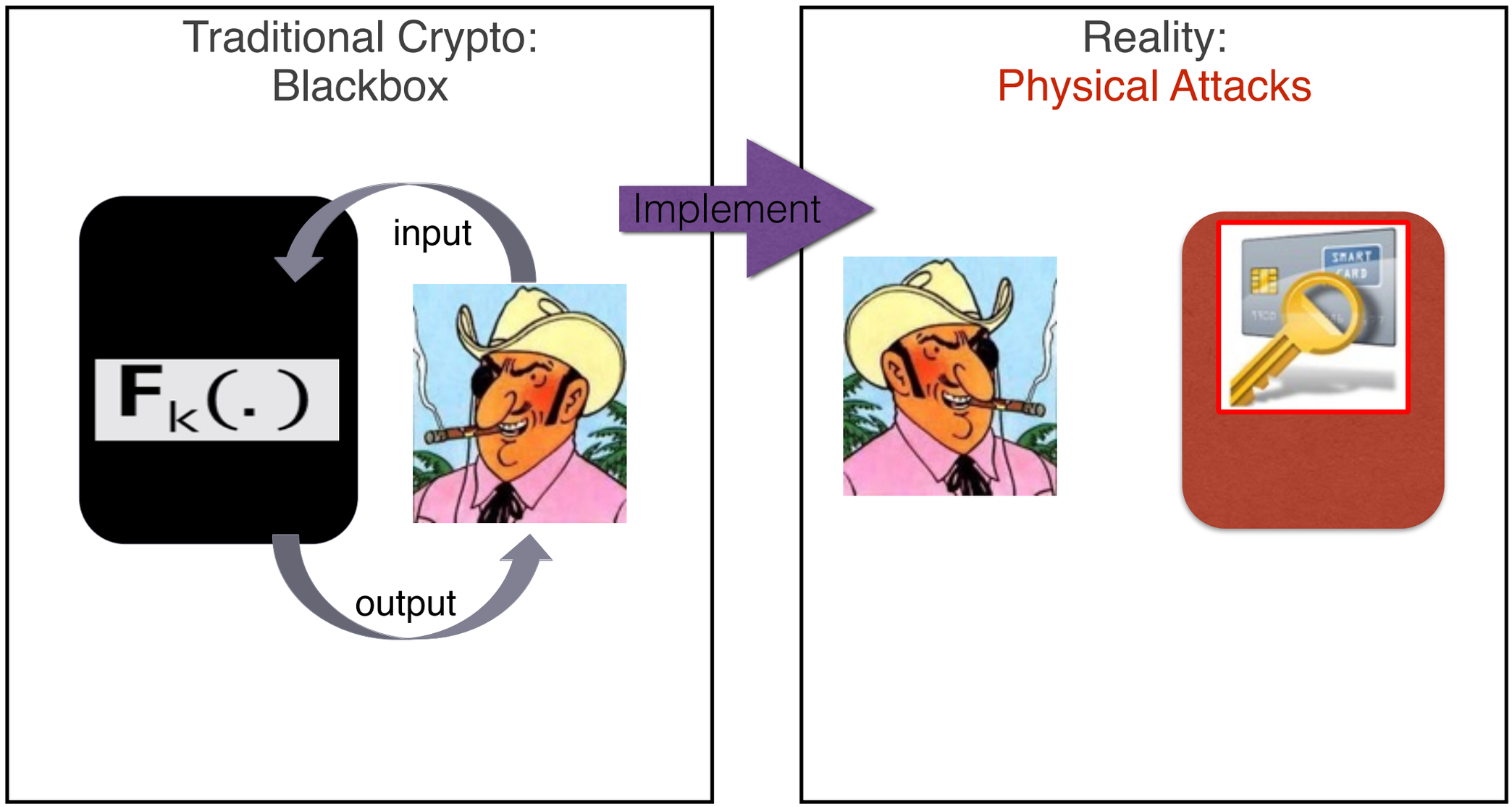
Physical Attack on implementations



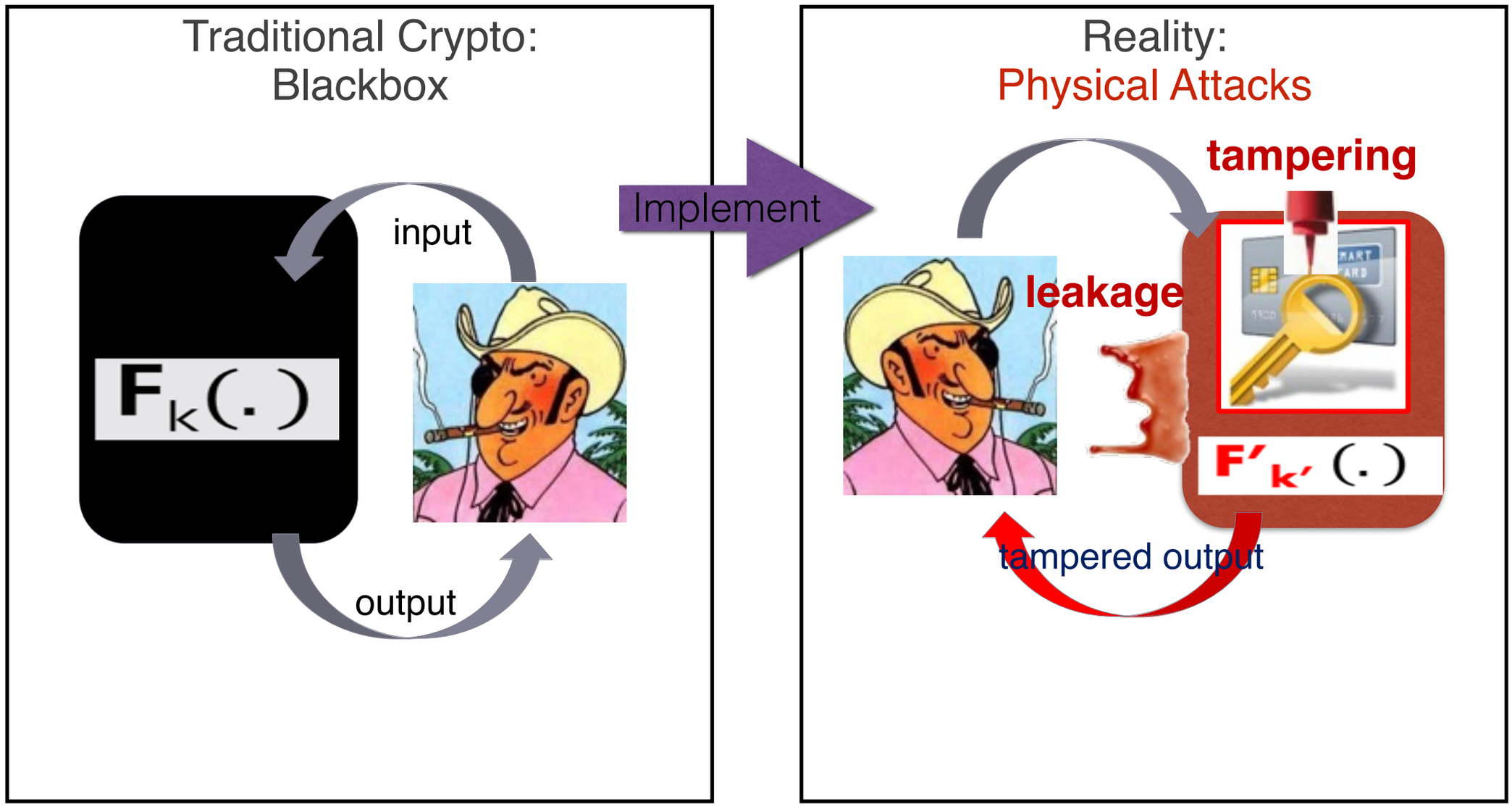
Physical Attack on implementations



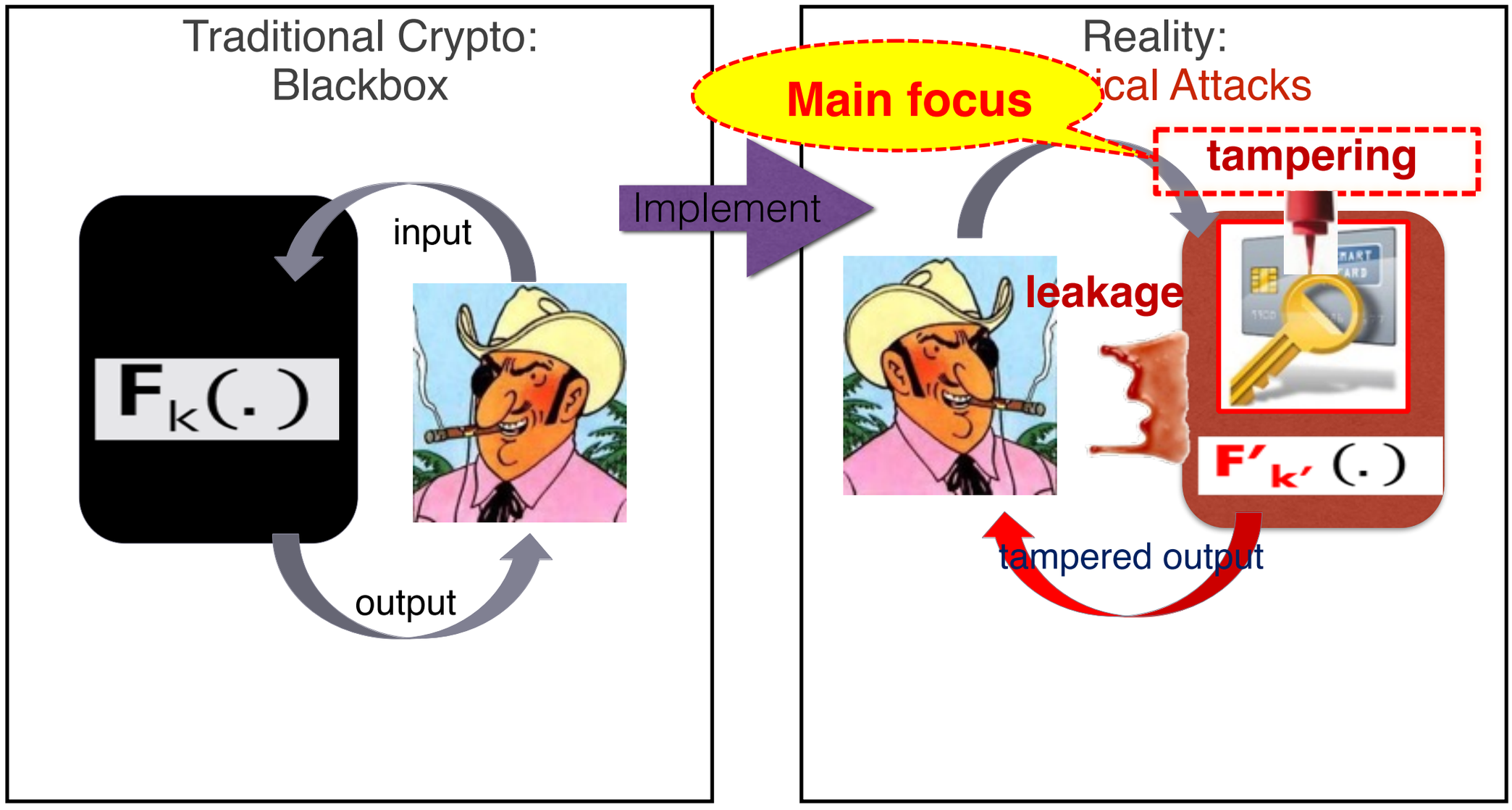
Physical Attack on implementations



Physical Attack on implementations



Physical Attack on implementations



Why care about tampering?

Why care about tampering?

BDL'01: Inject single (random) fault to the signing-key of some type of RSA-sig

Factor RSA-modulus !

Why care about tampering?

BDL'01: Inject single (random) fault to the signing-key of some type of RSA-sig

Factor RSA-modulus !

More…

Anderson and Kuhn '96

Skorobogatov et al. '02

Coron et al. '09

……and many more…….

Why care about tampering?

BDL'01: Inject single (random) fault to the signing-key of some type of RSA-sig

Factor RSA-modulus !

More…

Anderson and Kuhn '96

Skorobogatov et al. '02

Coron et al. '09

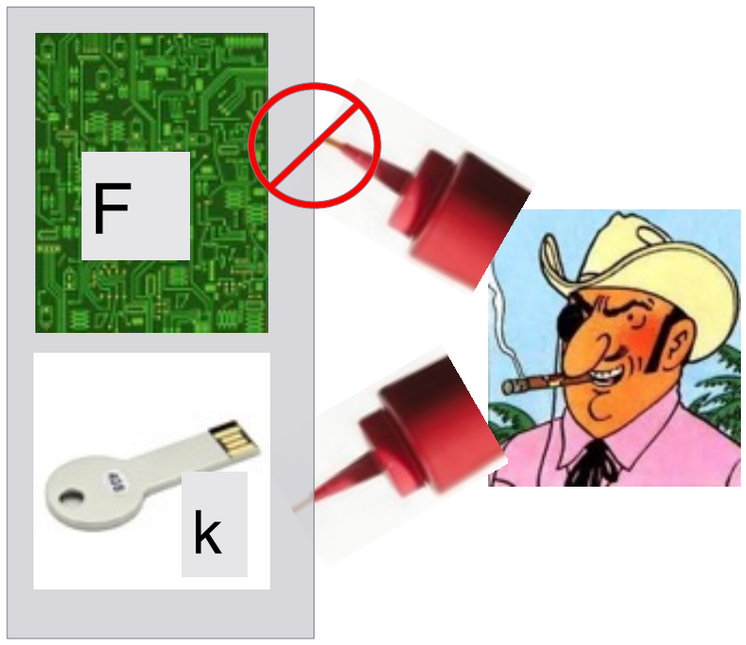
……and many more…….

Devastating attacks on Provably Secure Crypto-systems!

Theoretical Models of Tampering

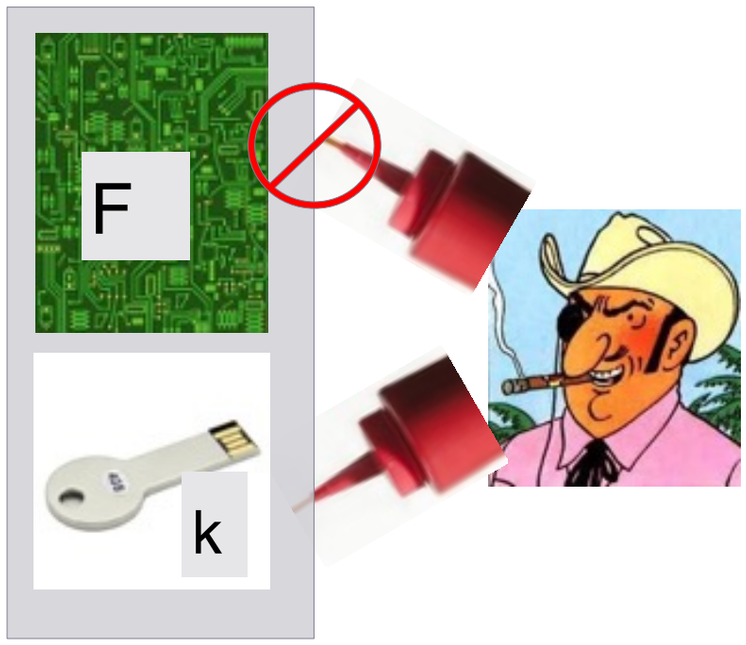
Theoretical Models of Tampering

Memory only tampering
(GLMMR ’04)



Theoretical Models of Tampering

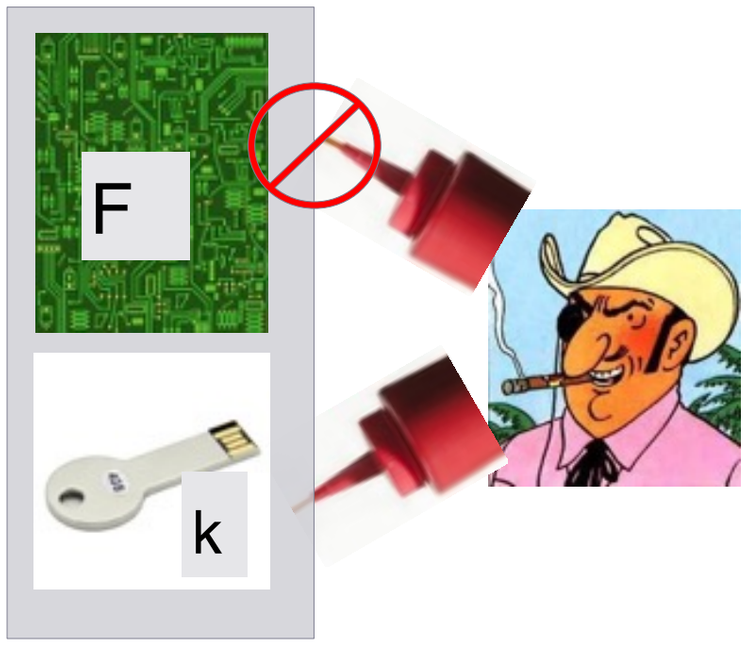
Memory only tampering
(GLMMR ’04)



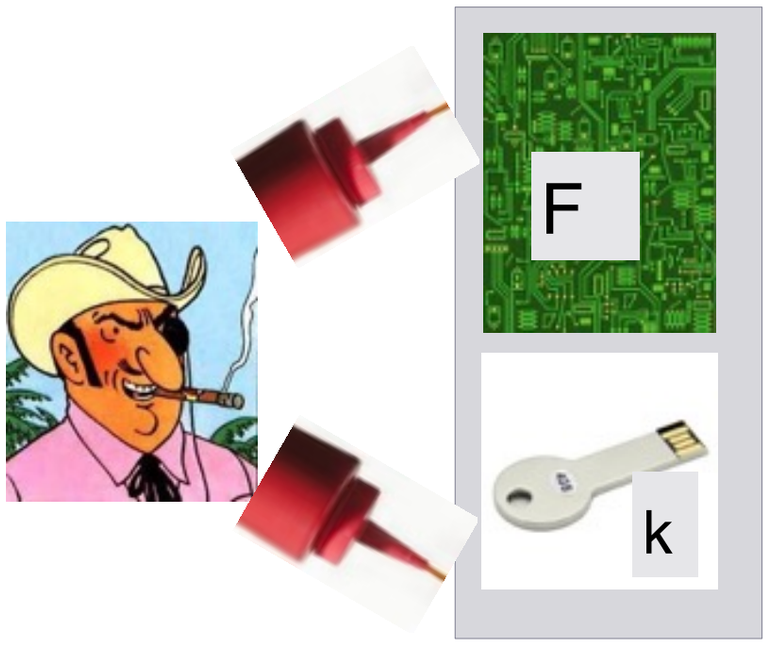
Tamper with both memory and computation
(IPSW ’06)

Theoretical Models of Tampering

Memory only tampering
(GLMMR ’04)

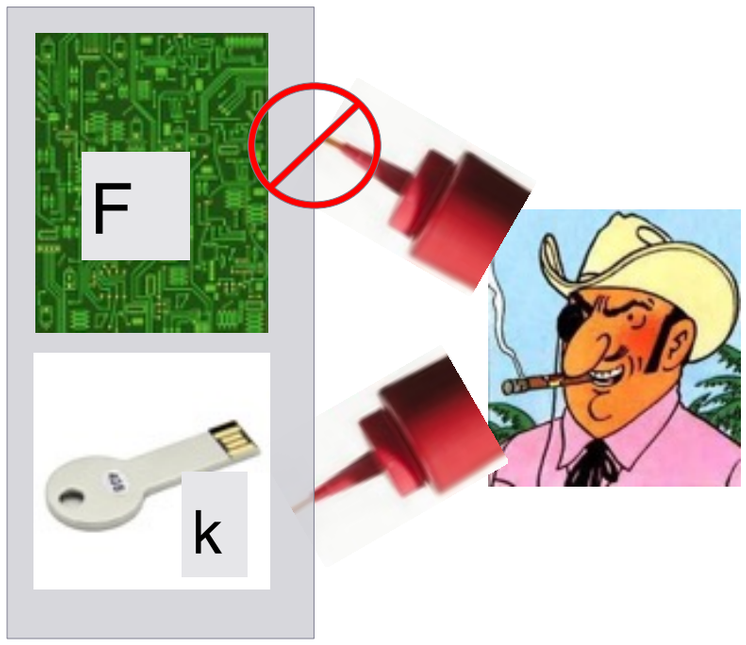


Tamper with both memory and computation
(IPSW ’06)



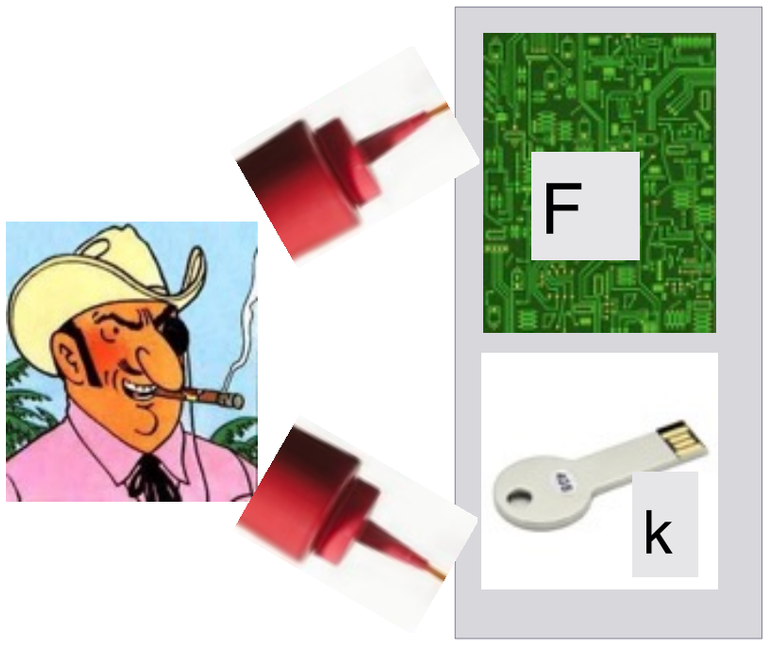
Theoretical Models of Tampering

Memory only tampering
(GLMMR ’04)



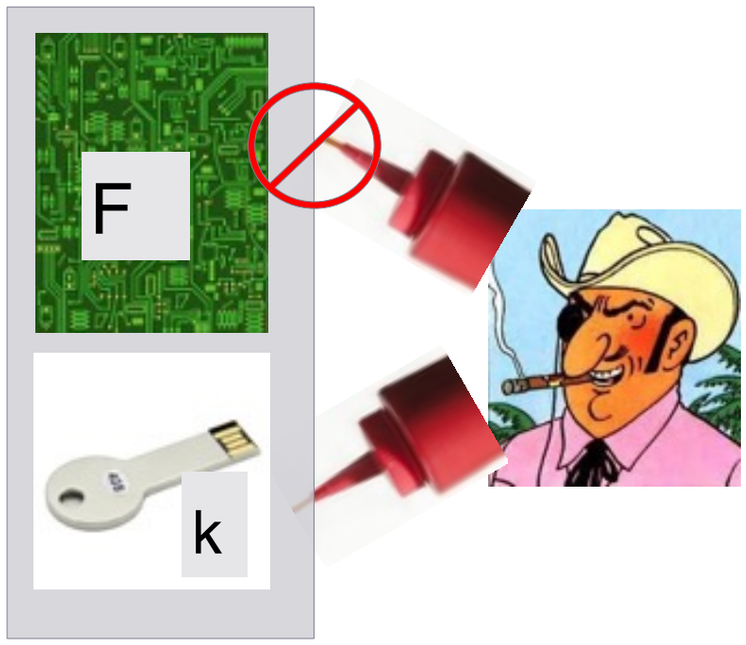
- Easier analysis.

Tamper with both memory and computation
(IPSW ’06)



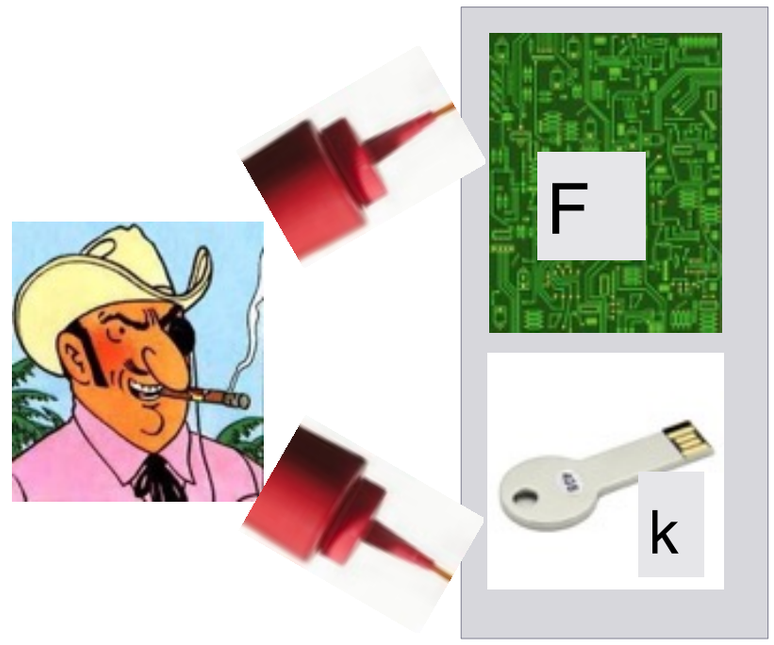
Theoretical Models of Tampering

Memory only tampering
(GLMMR ’04)



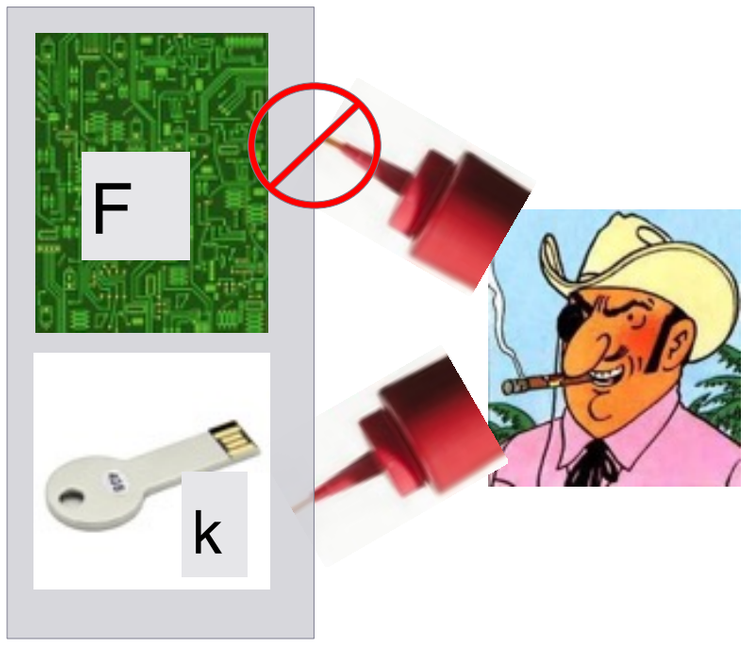
- Easier analysis.
- Lot of positive results

Tamper with both memory and computation
(IPSW ’06)



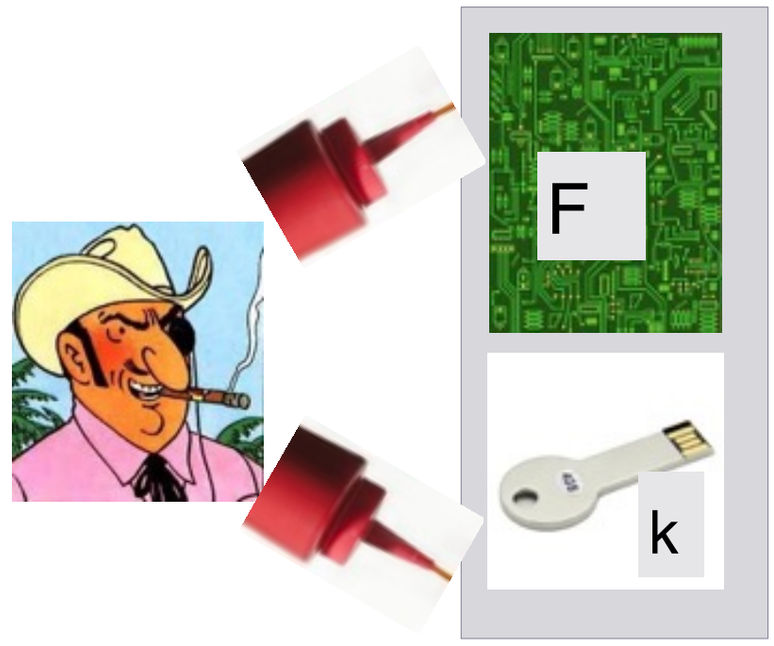
Theoretical Models of Tampering

Memory only tampering
(GLMMR ’04)



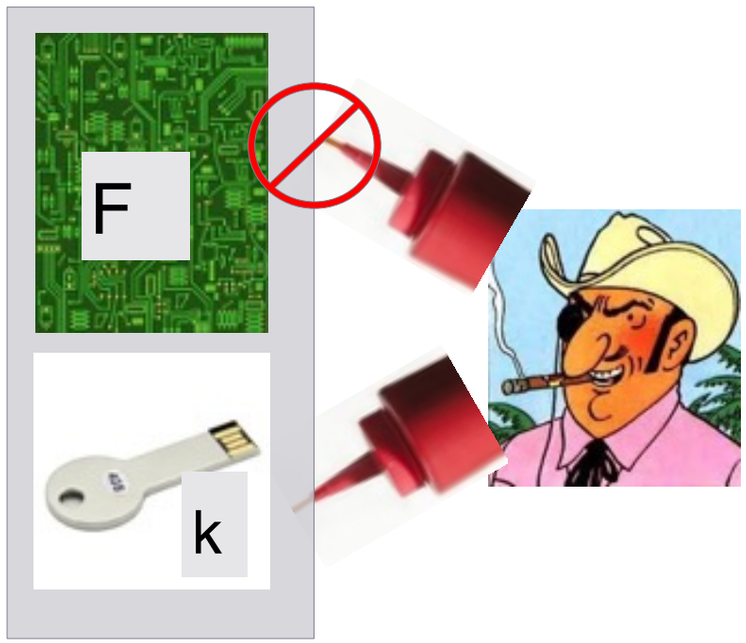
- Easier analysis.
- Lot of positive results
- In practice hard to guarantee.

Tamper with both memory and computation
(IPSW ’06)



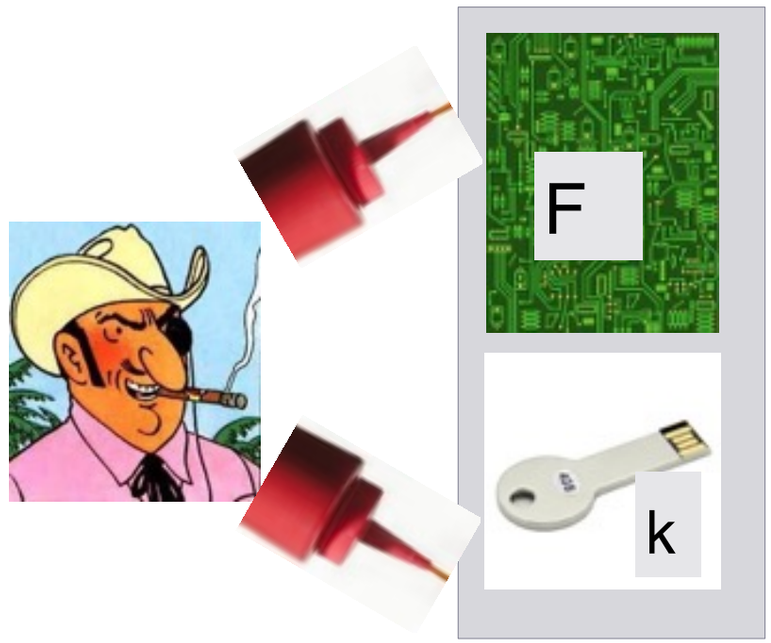
Theoretical Models of Tampering

Memory only tampering
(GLMMR ’04)



- Easier analysis.
- Lot of positive results
- In practice hard to guarantee.

Tamper with both memory and computation
(IPSW ’06)



This talk

Earlier approach: protect circuits against tampering

Earlier approach: protect circuits against tampering

Approach: Model computation as circuits: build circuit compiler which transforms any C to C’ which is protected against tampering

Earlier approach: protect circuits against tampering

Approach: Model computation as circuits: build circuit compiler which transforms any C to C' which is protected against tampering

First initiative by IPSW06: protects against constant number of wire faults

Earlier approach: protect circuits against tampering

Approach: Model computation as circuits: build circuit compiler which transforms any C to C' which is protected against tampering

First initiative by IPSW06: protects against constant number of wire faults

Current best: DK14:

- Protects against tampering of (1/poly)-fraction of all wires.
- Analysis complicated: involves tools like PCP.

Earlier approach: protect circuits against tampering

Approach: Model computation as circuits: build circuit compiler which transforms any C to C' which is protected against tampering

First initiative by IPSW06: protects against constant number of wire faults

Current best: DK14:

- Protects against tampering of (1/poly)-fraction of all wires.
- Analysis complicated: involves tools like PCP.

Our goal:

- simpler framework
- stronger tampering adversary

Our approach

Our approach

An alternative model of computation:
von Neumann Architecture (vNA)/ Random Access Machine (RAM)

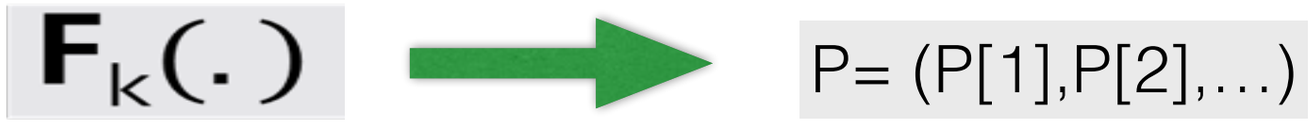
Our approach

An alternative model of computation:
von Neumann Architecture (vNA)/ Random Access Machine (RAM)

$F_k(.)$

Our approach

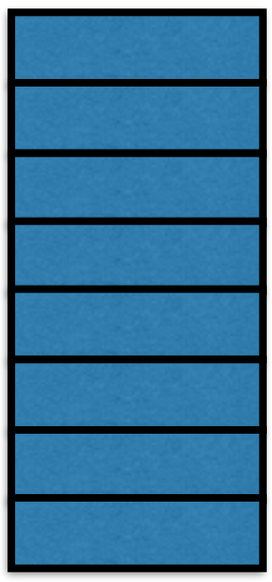
An alternative model of computation:
von Neumann Architecture (vNA)/ Random Access Machine (RAM)



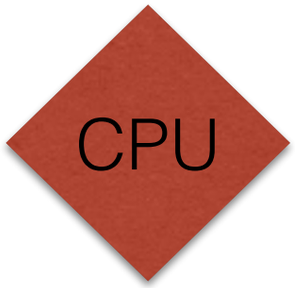
Our approach

An alternative model of computation:
von Neumann Architecture (vNA)/ Random Access Machine (RAM)

$F_k(.)$ → P= (P[1],P[2],…)

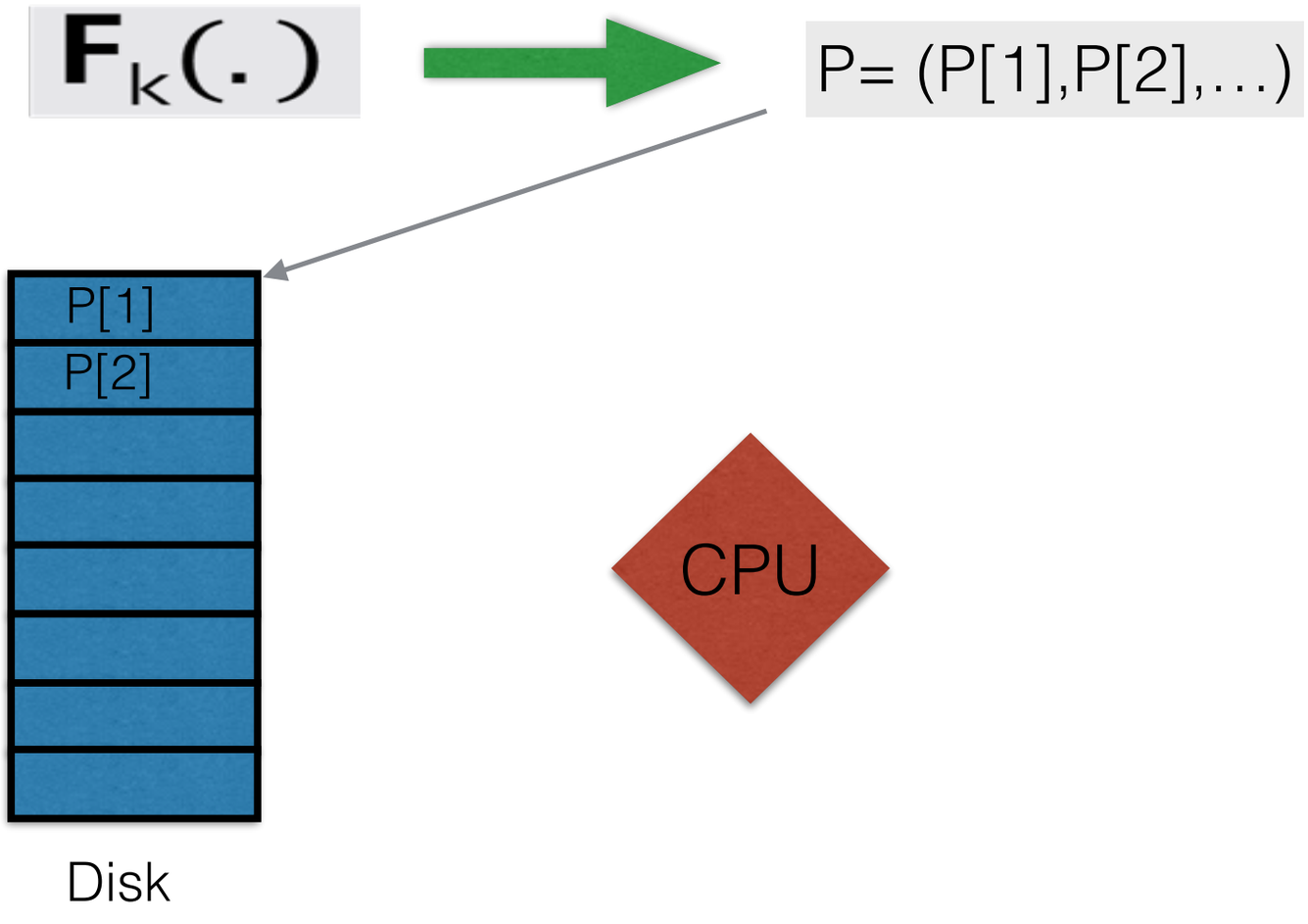


Disk



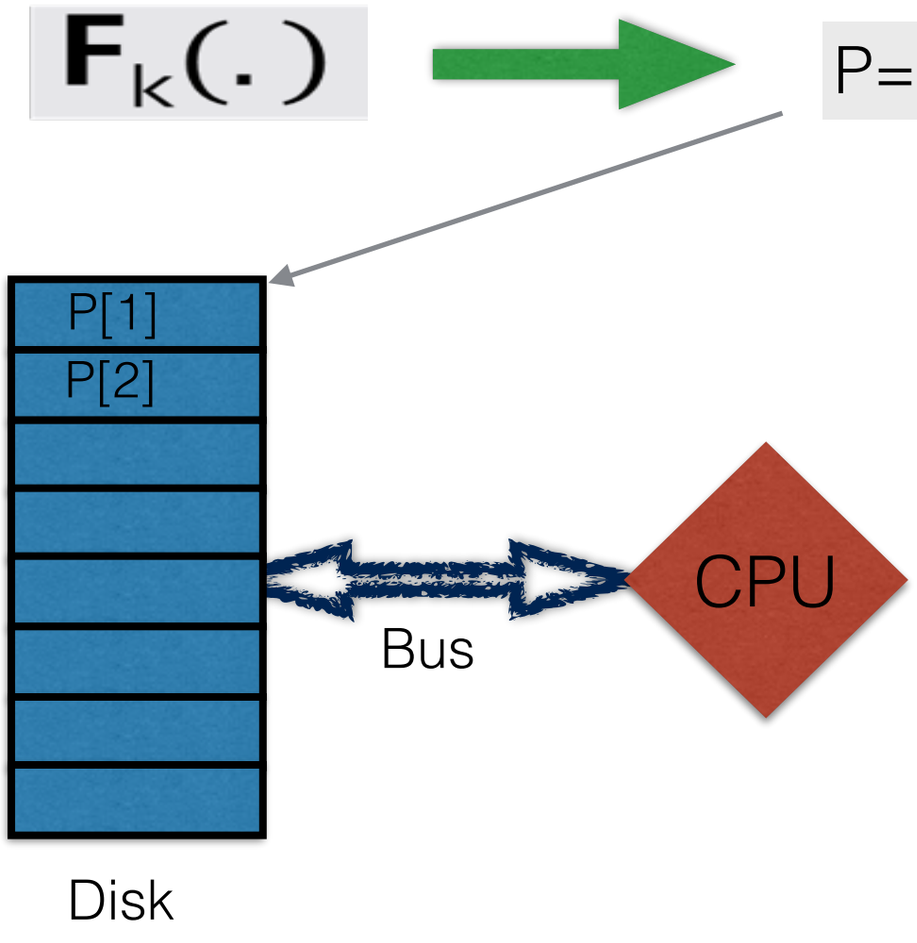
Our approach

An alternative model of computation:
von Neumann Architecture (vNA)/ Random Access Machine (RAM)



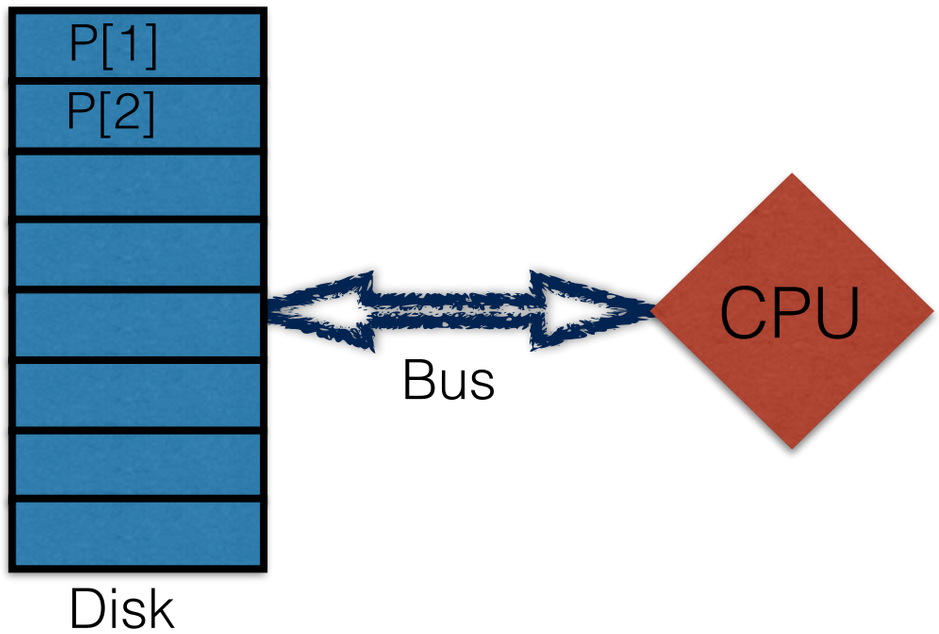
Our approach

An alternative model of computation:
von Neumann Architecture (vNA)/ Random Access Machine (RAM)

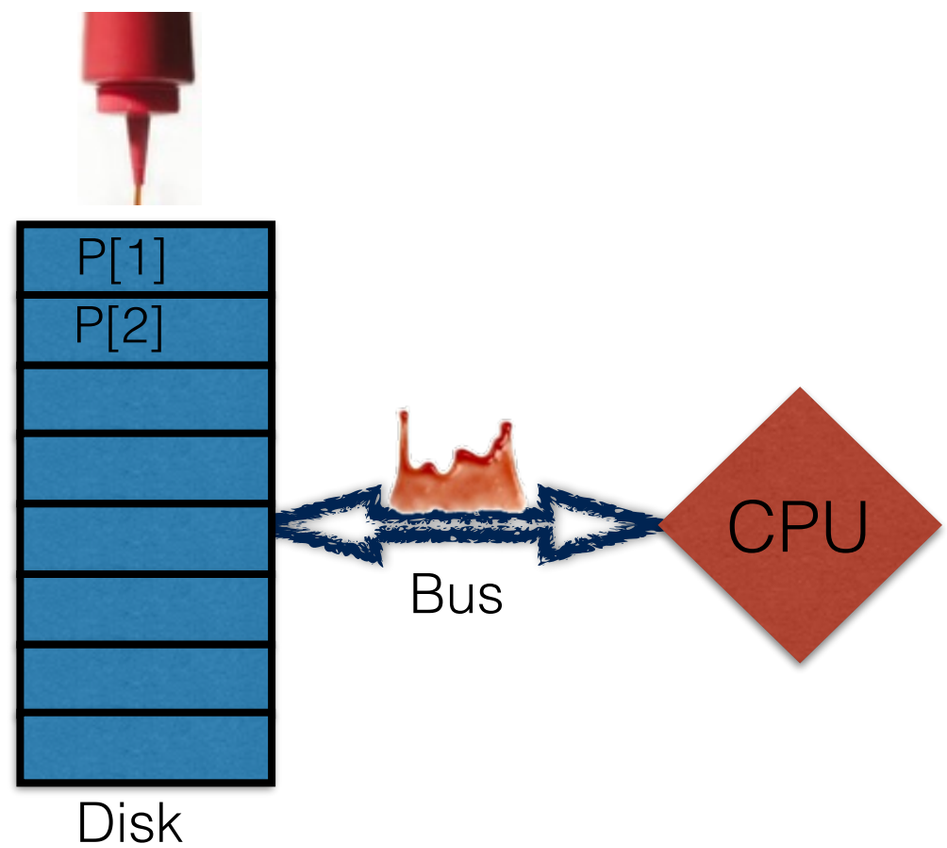


Tamper and leakage resilient vNA

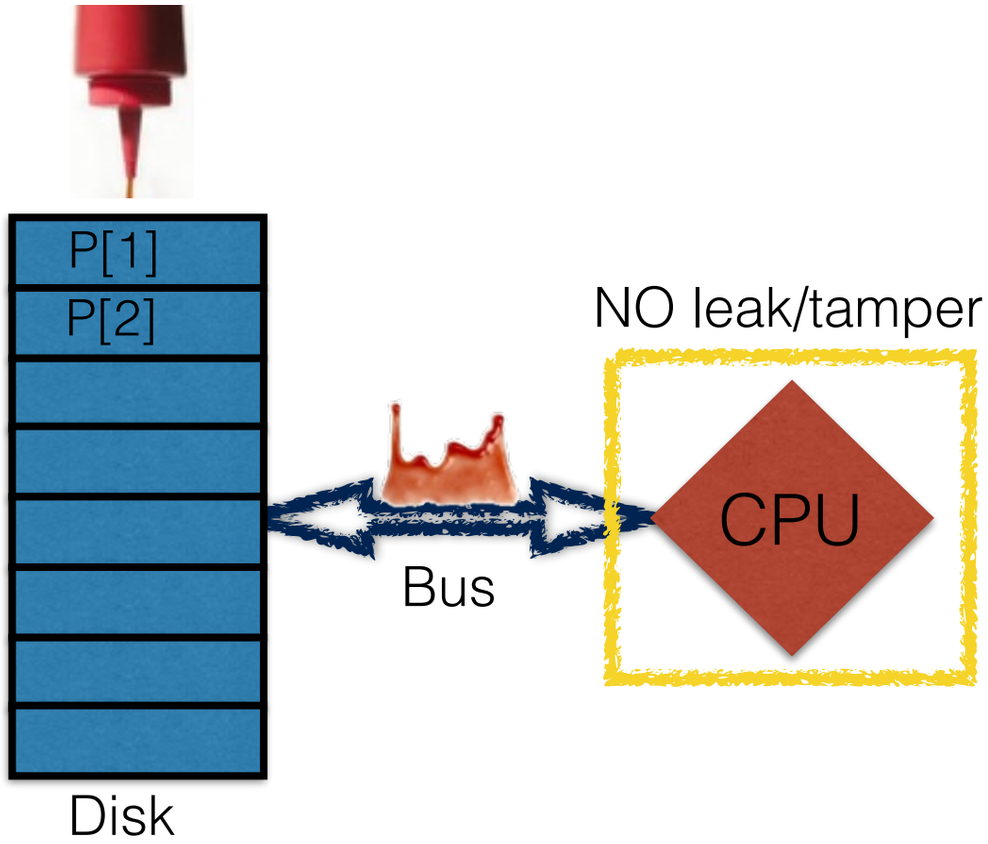
Tamper and leakage resilient vNA



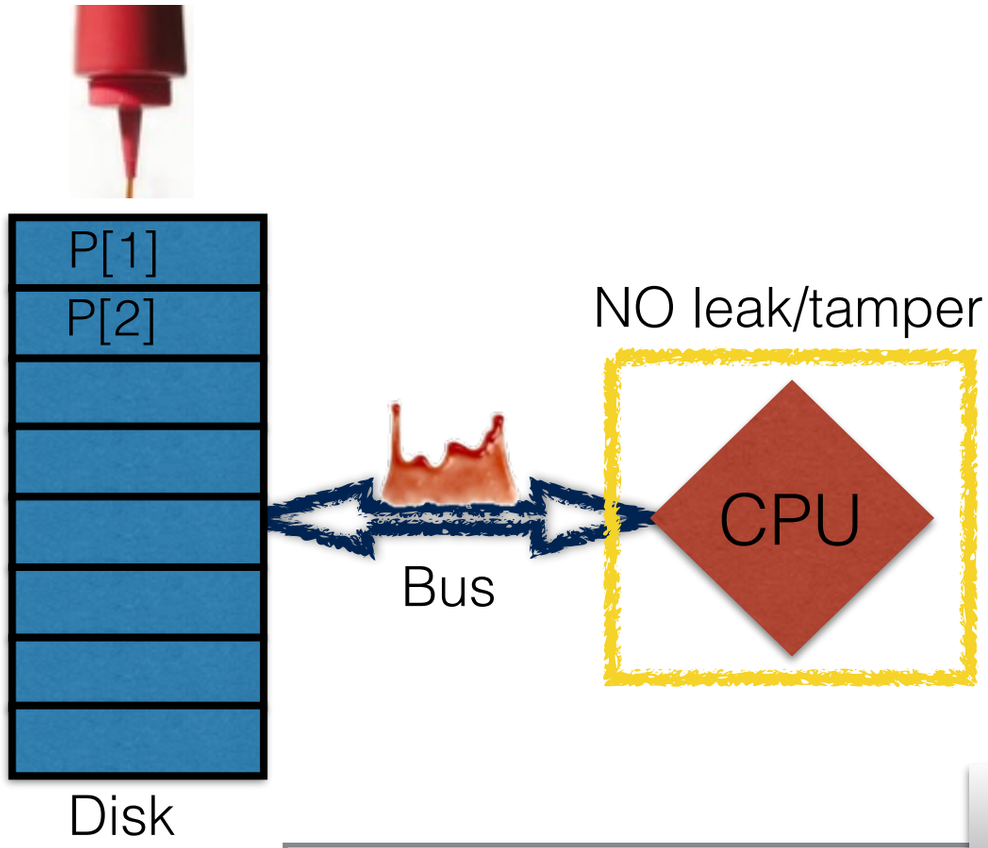
Tamper and leakage resilient vNA



Tamper and leakage resilient vNA



Tamper and leakage resilient vNA

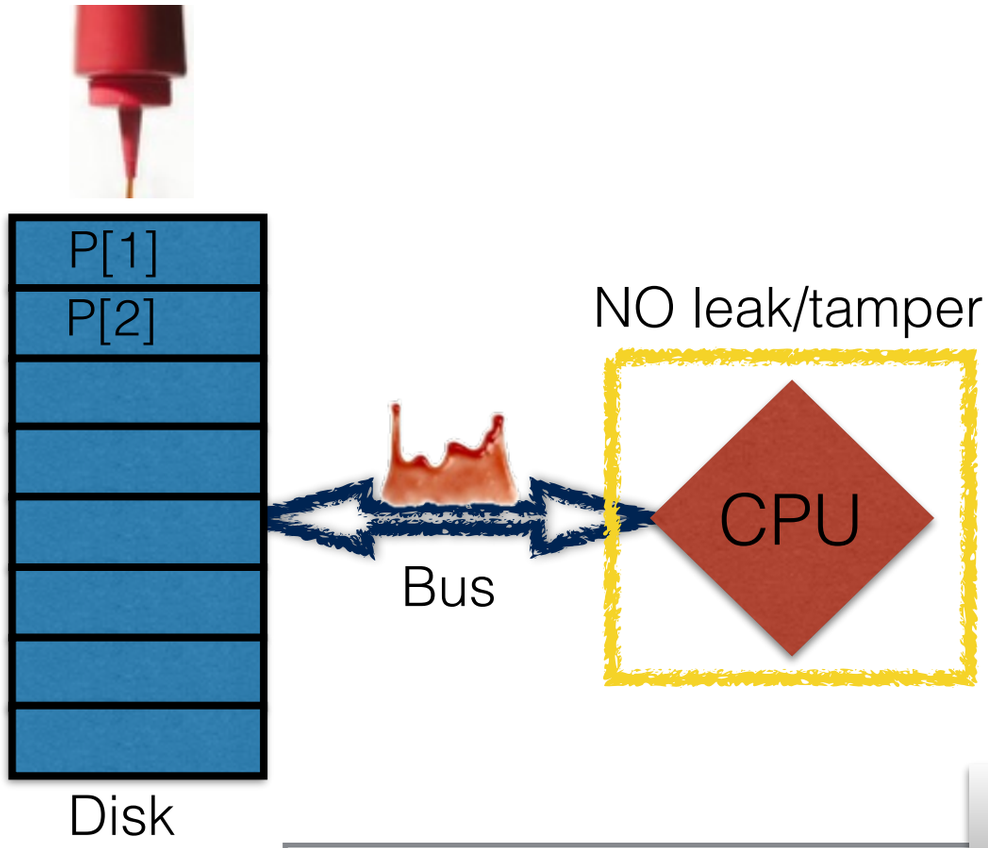


NO leak/tamper

CPU

- “small” and “universal”

Tamper and leakage resilient vNA



CPU

- “small” and “universal”
- Does not store any secret.
  - otherwise simple solution !
  - stores a untamperable special purpose bit.

Our results

Our results

A general framework to compute any keyed functionality in a tampering-leakage environment.

Our results

A general framework to compute any keyed functionality in a tampering-leakage environment.

Reducing the problem of shielding arbitrary complex computations to protecting a single, simple,universal component (CPU): Similar in spirit with leakage-resilient computations (GR10)

Our results

A general framework to compute any keyed functionality in a tampering-leakage environment.

Reducing the problem of shielding arbitrary complex computations to protecting a single, simple,universal component (CPU): Similar in spirit with leakage-resilient computations (GR10)

We construct a compiler which turns program P for ideal vNA to a program P' for vNA under tampering: using non-malleable codes (black-box)

Our results

A general framework to compute any keyed functionality in a tampering-leakage environment.

Reducing the problem of shielding arbitrary complex computations to protecting a single, simple,universal component (CPU): Similar in spirit with leakage-resilient computations (GR10)

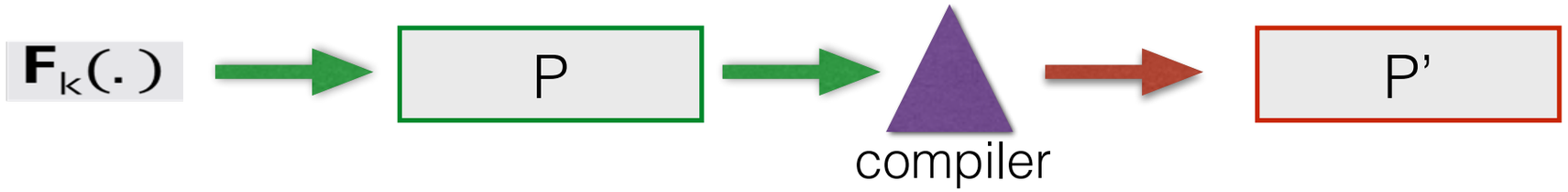
We construct a compiler which turns program P for ideal vNA to a program P' for vNA under tampering: using non-malleable codes (black-box)

DLSZ15: A concurrent and independent work

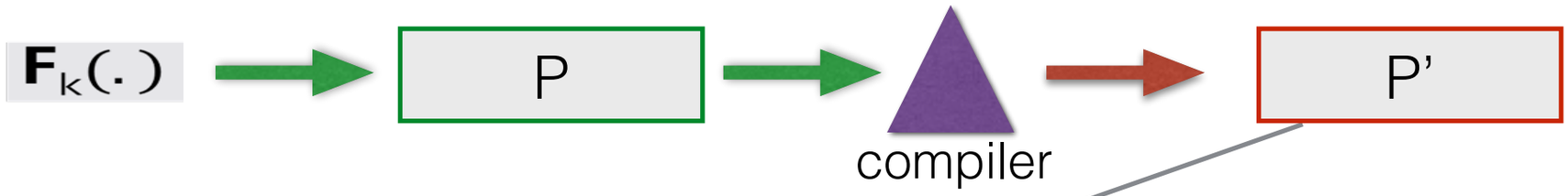
Locally Decodable and Updatable Non-Malleable Codes and Their Applications

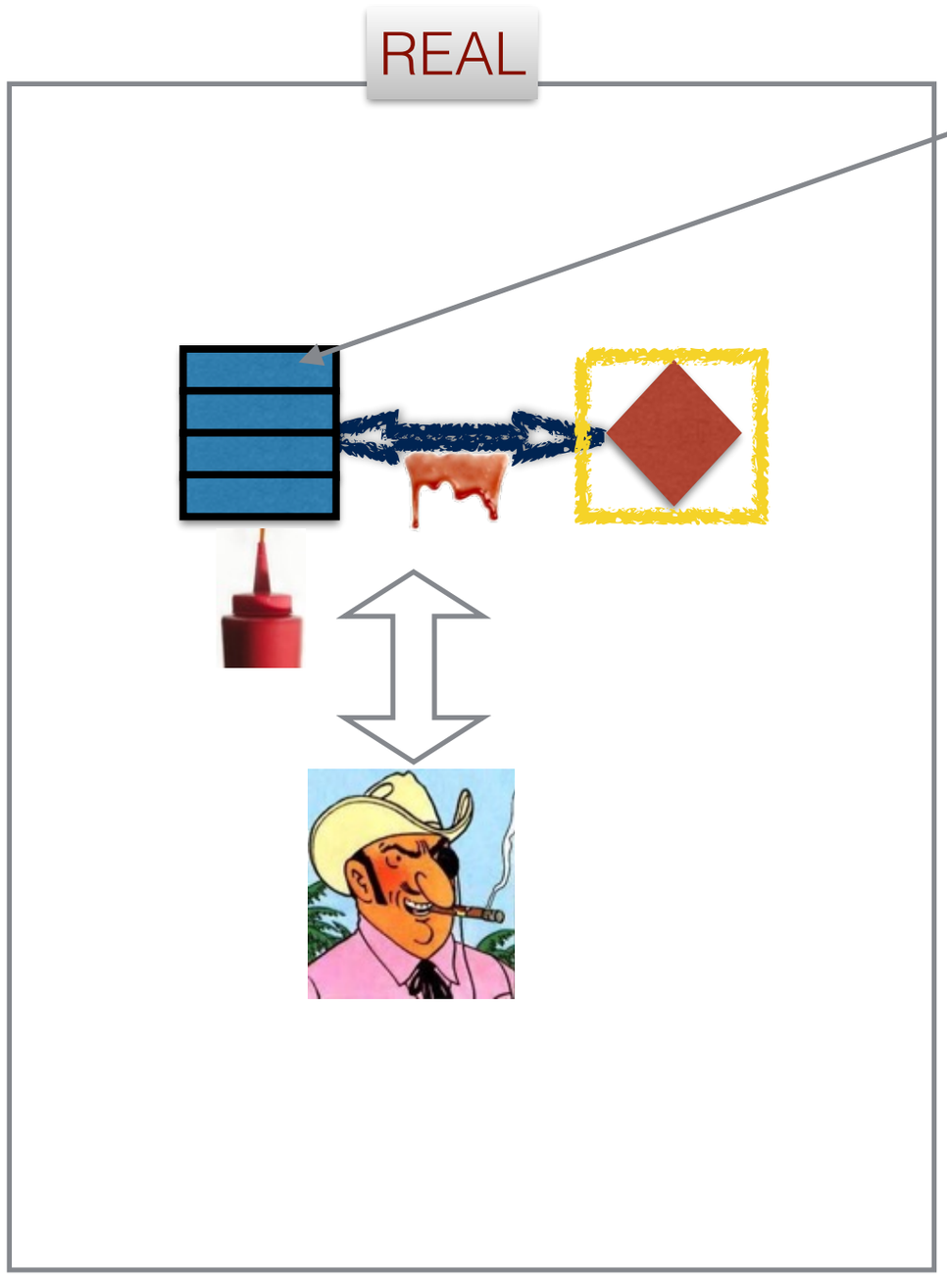
The security notion

The security notion

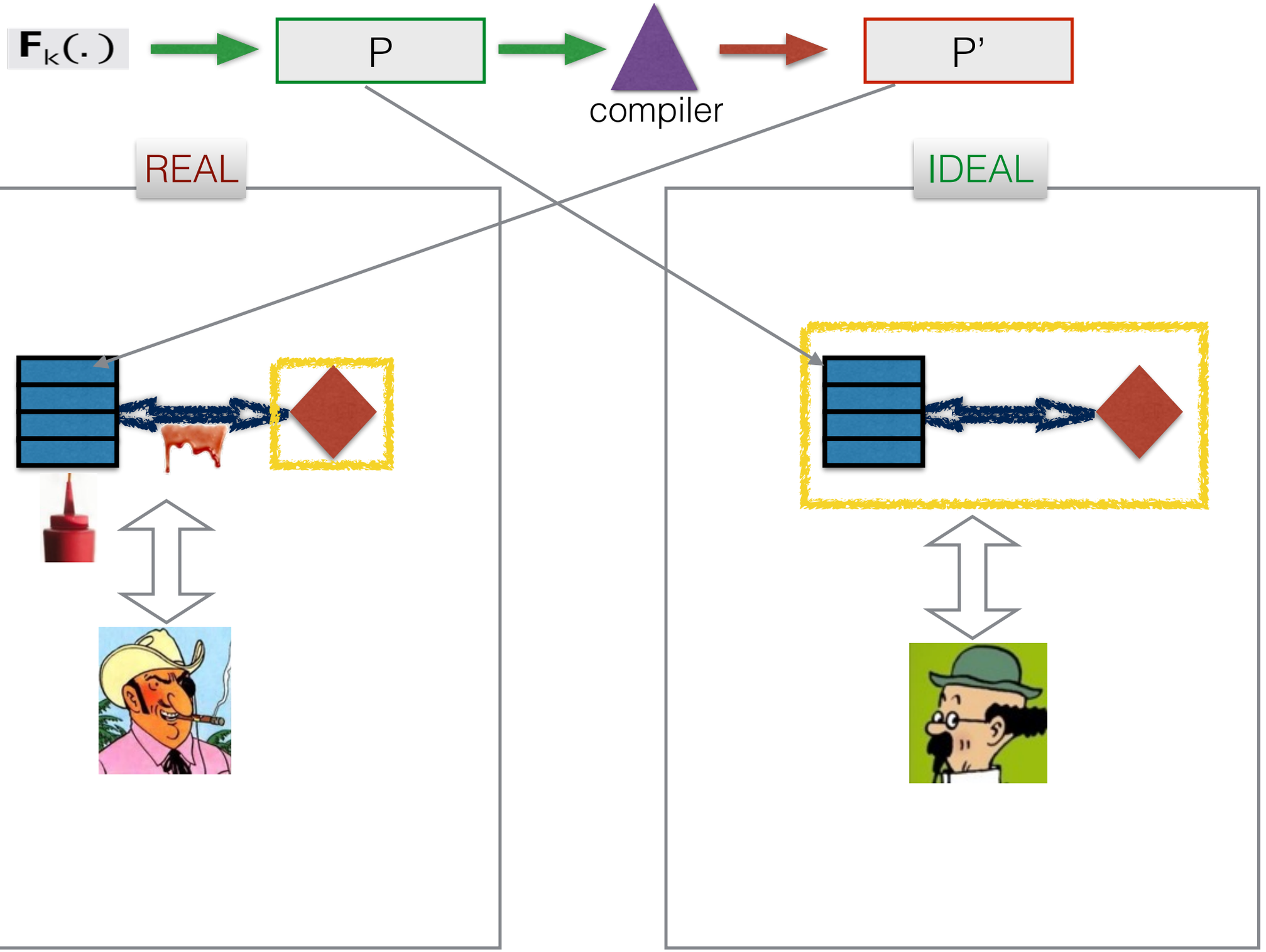


The security notion

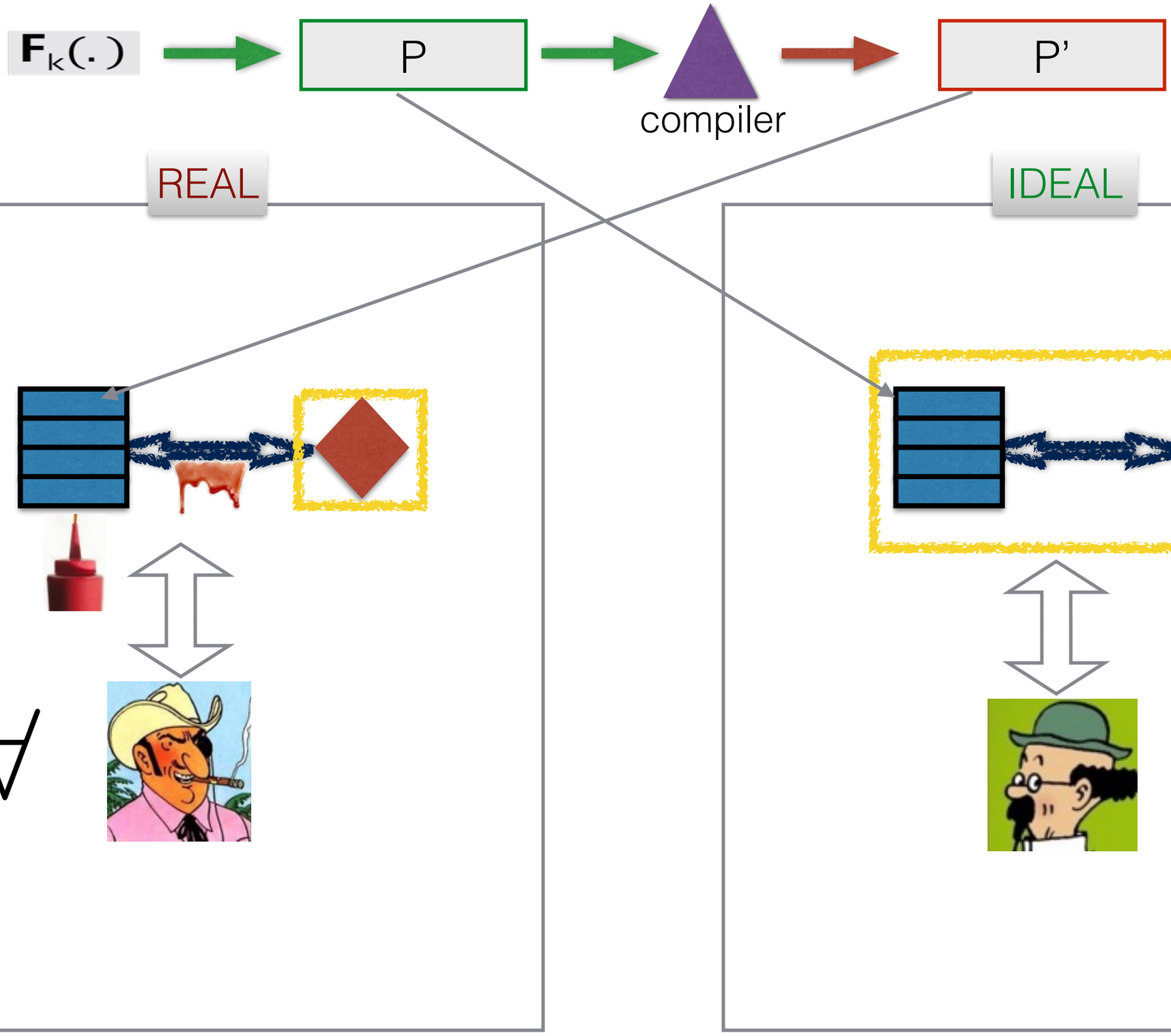




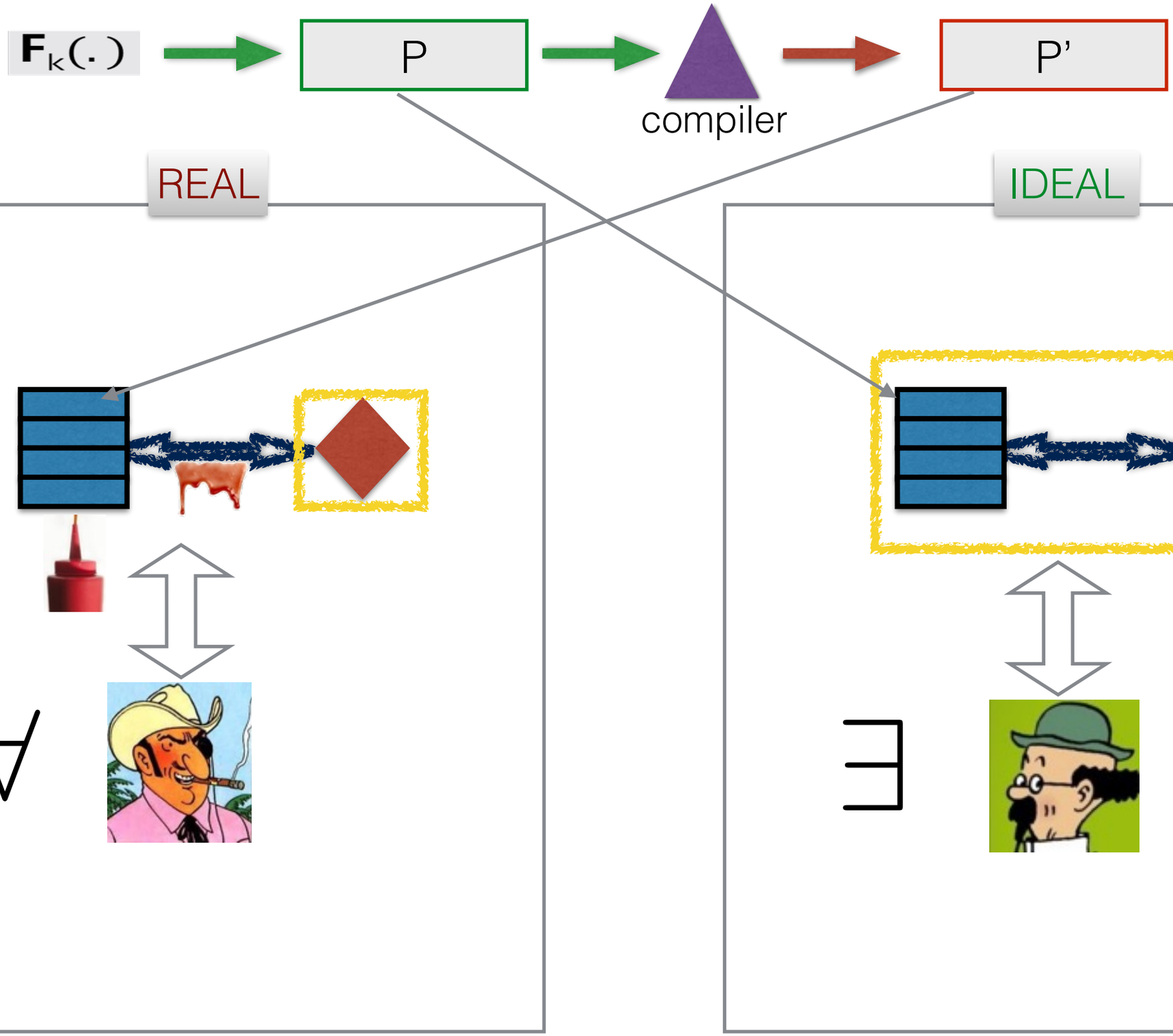
The security notion



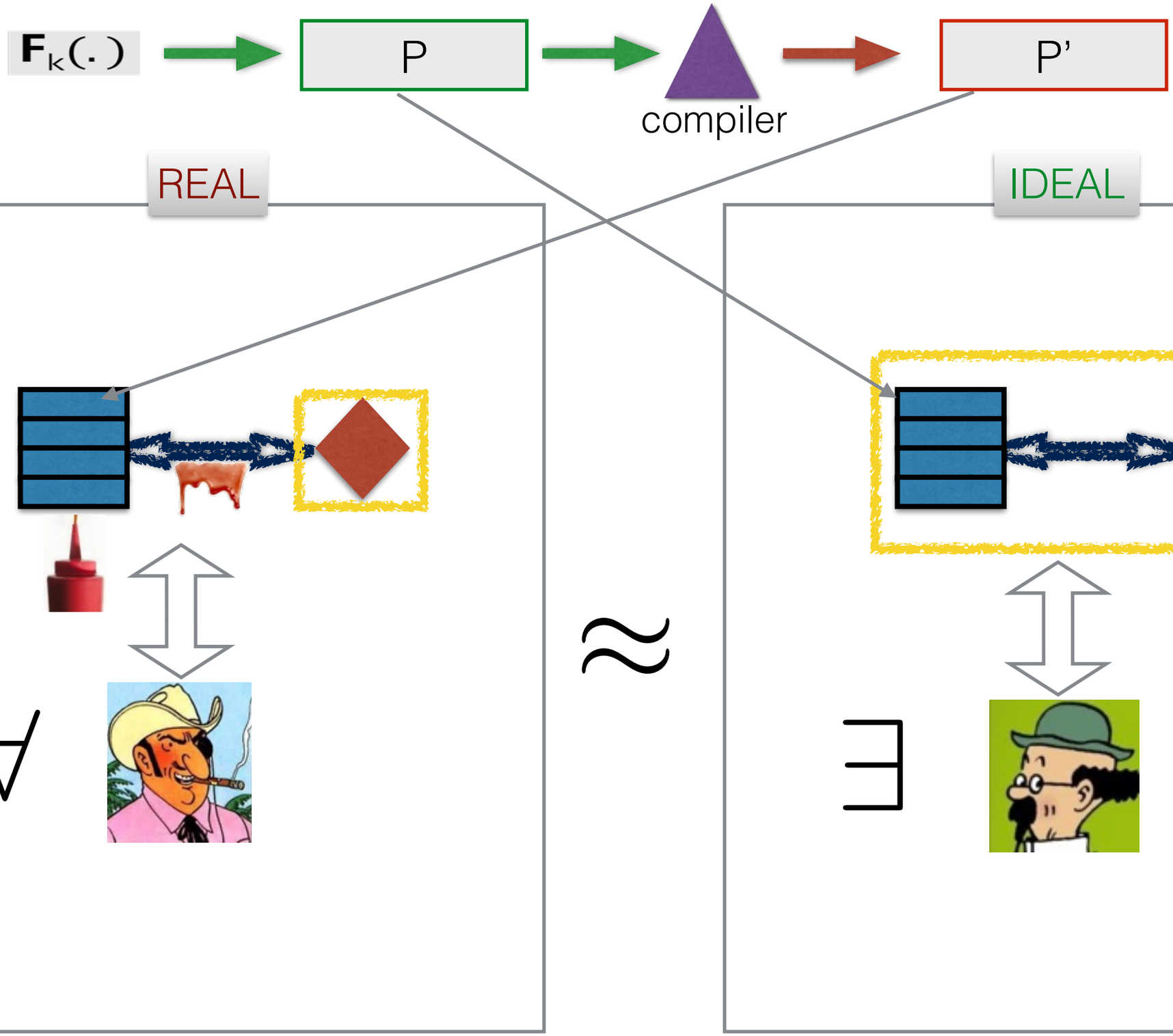
The security notion



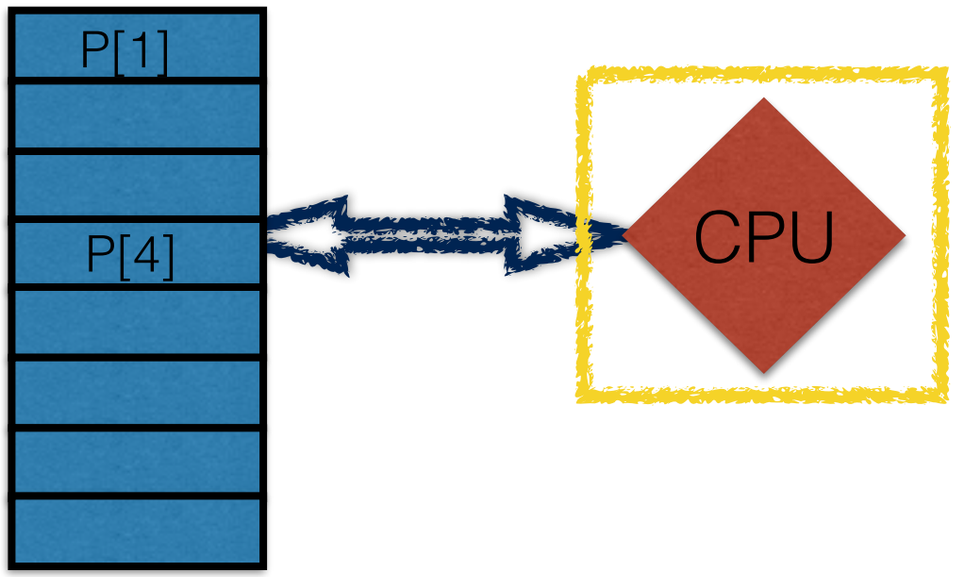
The security notion



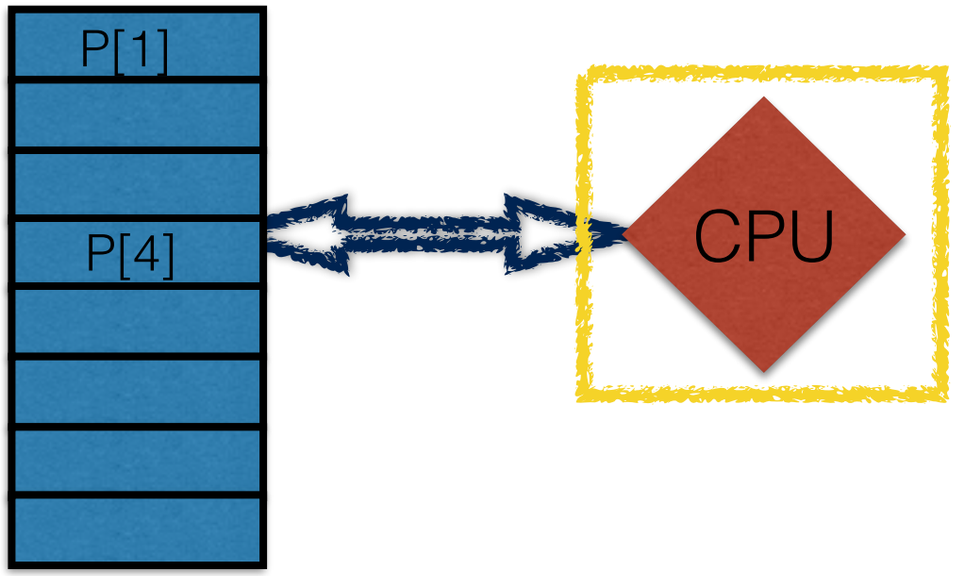
The security notion



Modular approach: Hybrid world

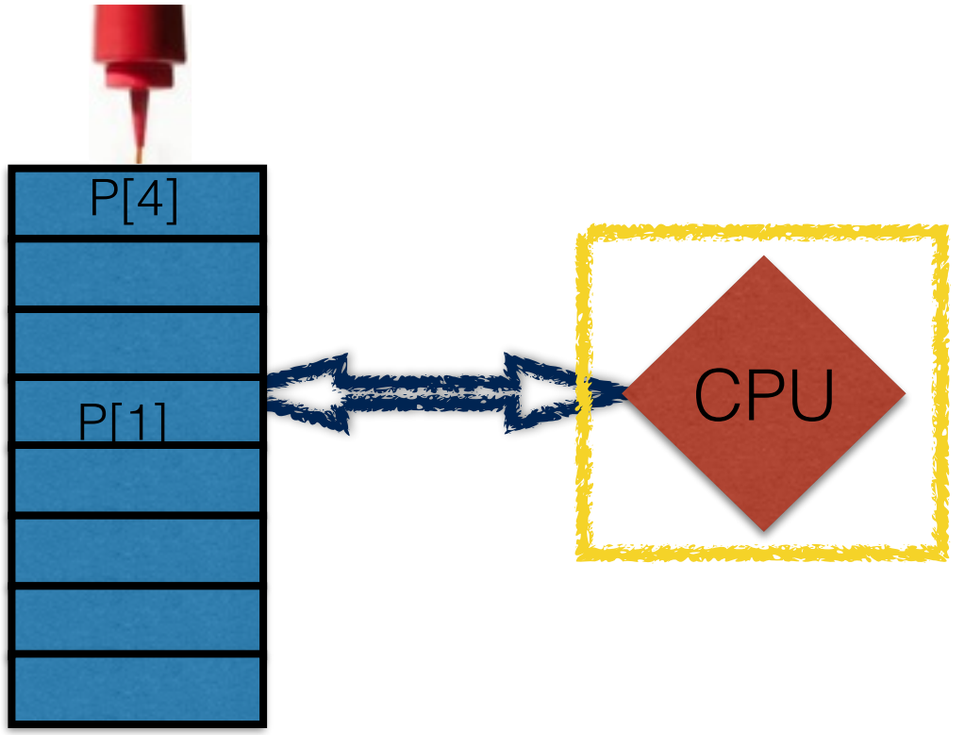


Modular approach: Hybrid world



No leakage

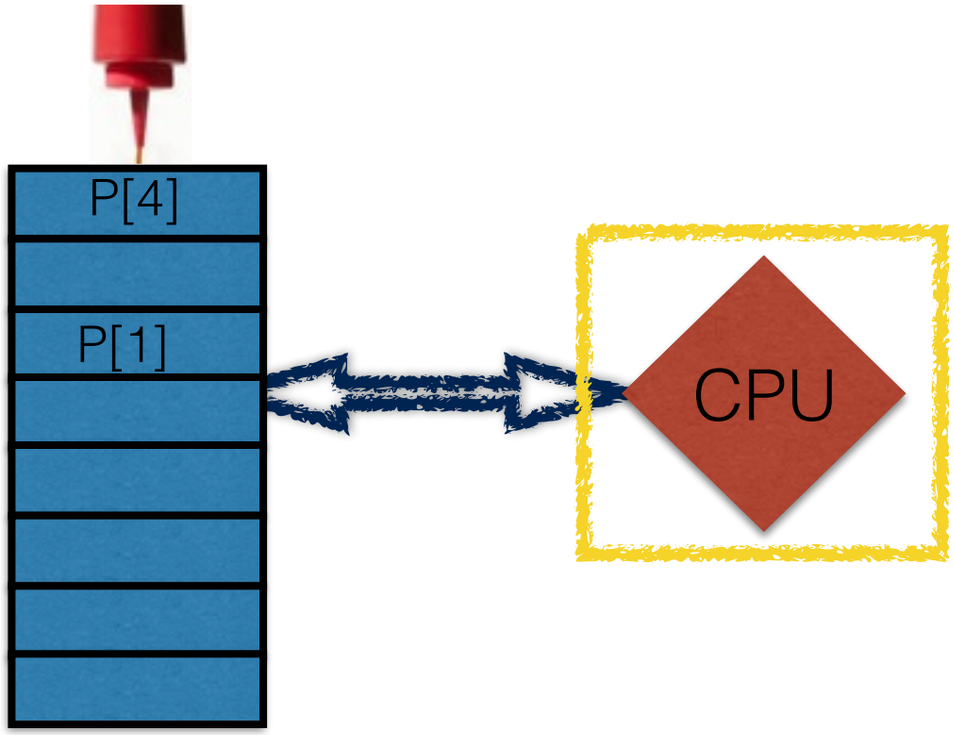
Modular approach: Hybrid world



No leakage

Mild tampering:
1. COPY within disk

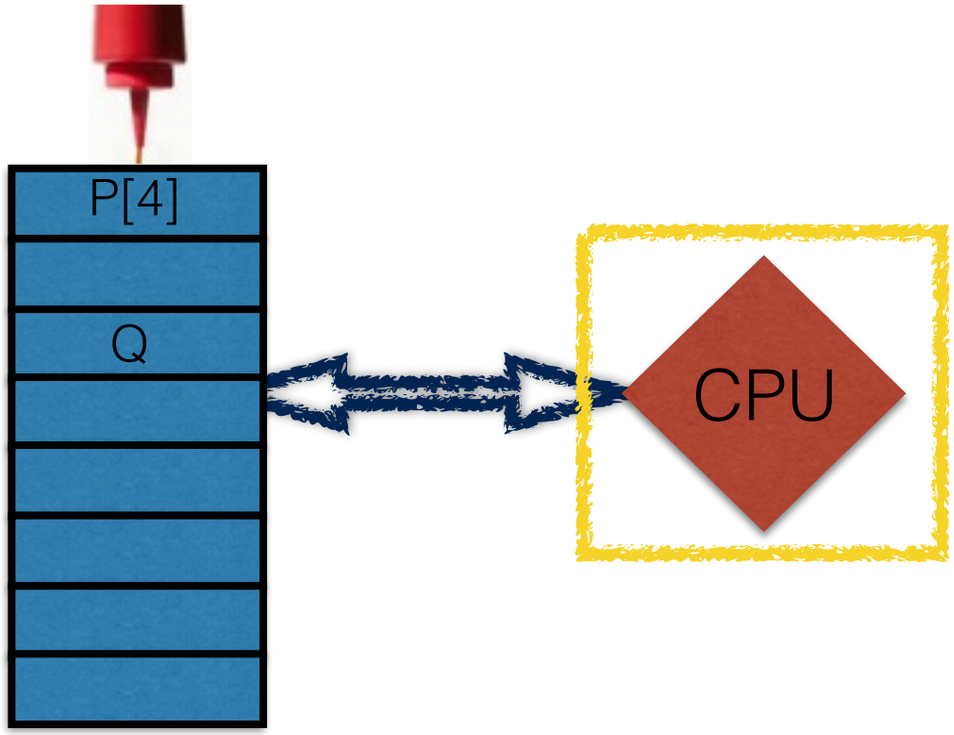
Modular approach: Hybrid world



No leakage

Mild tampering:
1. COPY within disk

Modular approach: Hybrid world



No leakage

Mild tampering:
1. COPY within disk
2. Overwrite

Our construction: Two steps

Our construction: Two steps

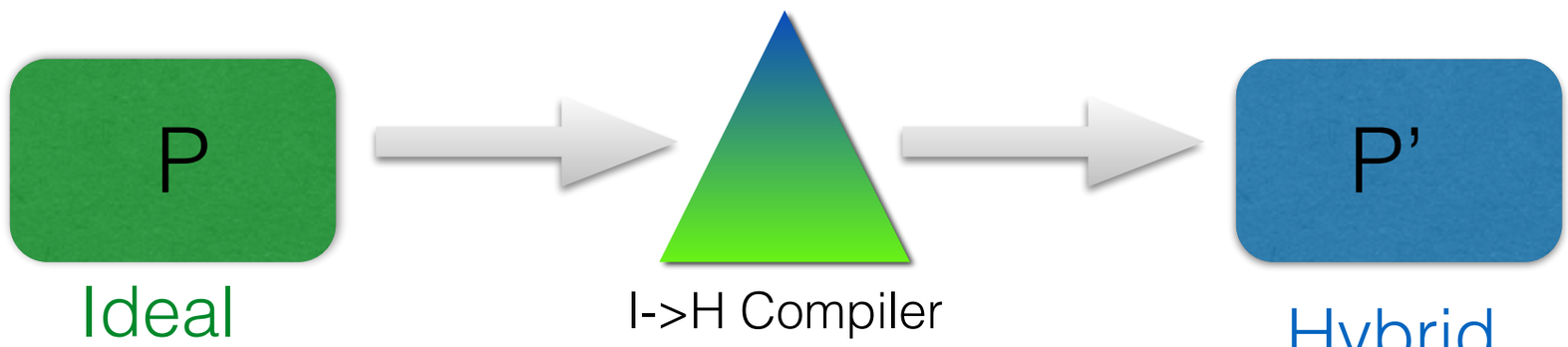
Our construction: Two steps

P= (P[1],P[2],…)

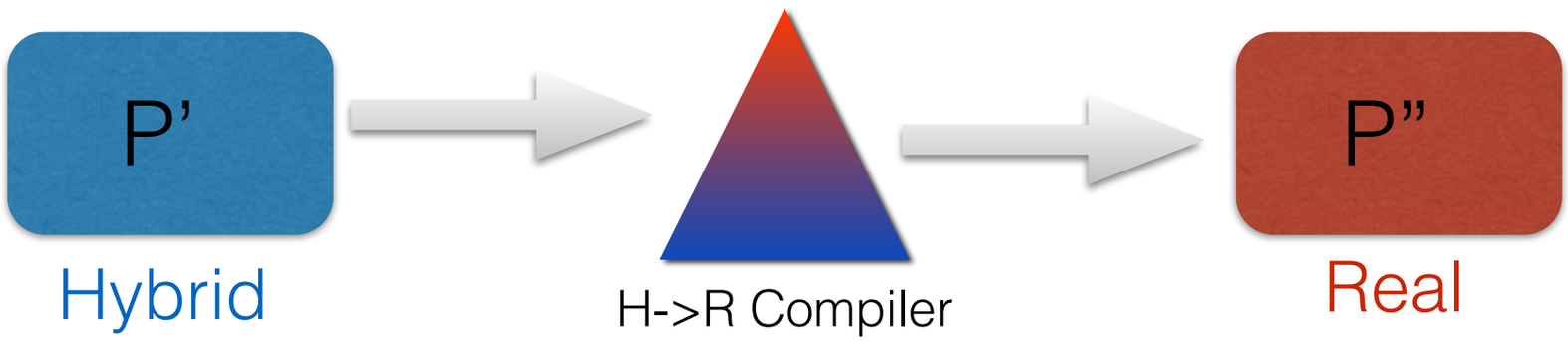
Step-1

Our construction: Two steps

Step-1



Step-2



Step-1: Ideal to Hybrid compiler

Step-1: Ideal to Hybrid compiler

Step-1: Ideal to Hybrid compiler

Recall : in hybrid model there is only limited tampering:
1. COPY 2. OVERWRITE

Step-1: Ideal to Hybrid compiler

Recall : in hybrid model there is only limited tampering:
1. COPY 2. OVERWRITE

Augment each P[i] by appending:

- A secret level L

Step-1: Ideal to Hybrid compiler

Recall : in hybrid model there is only limited tampering:
1. COPY 2. OVERWRITE

Augment each P[i] by appending:
- A secret level L

- Binds the instructions and secrets
- Guarantees that if the adversary overwrites instructions, it has to overwrite secrets.
  - Otherwise one can just overwrites instructions to output secrets.

Step-1: Ideal to Hybrid compiler

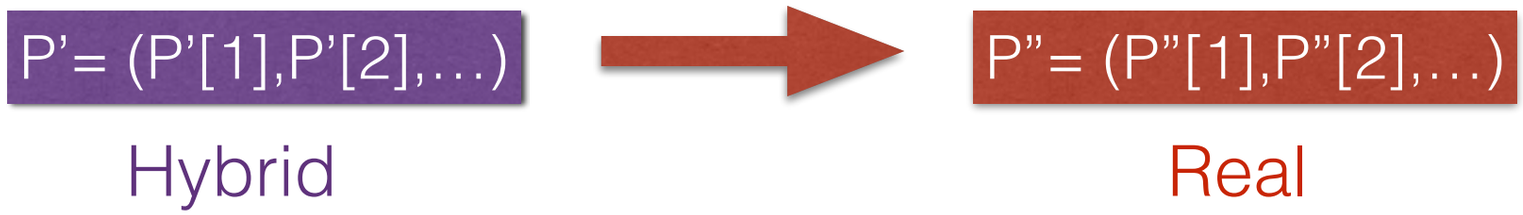
Recall : in hybrid model there is only limited tampering:
1. COPY 2. OVERWRITE

Augment each P[i] by appending:
- A secret level L
- The position i

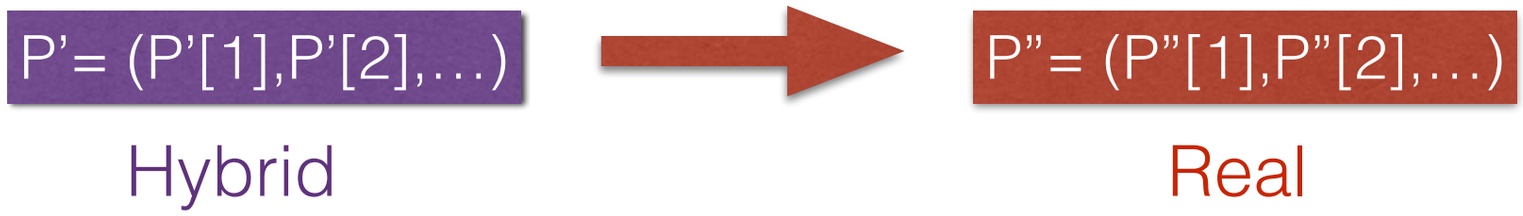
Protect against copying from one location to another

Step-2: Hybrid to Real compiler

Step-2: Hybrid to Real compiler

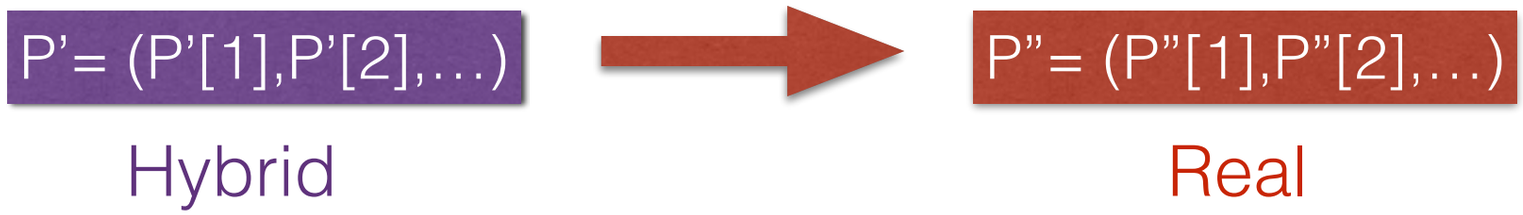


Step-2: Hybrid to Real compiler



Idea: Encode each P'[i] such that the adversary can only copy or overwrite

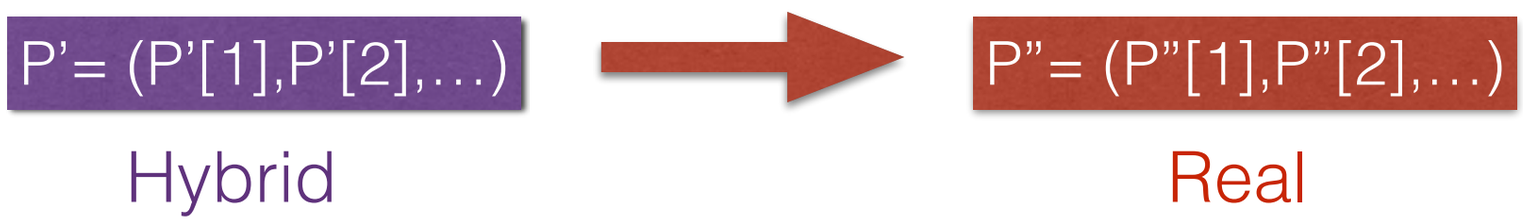
Step-2: Hybrid to Real compiler



Idea: Encode each P'[i] such that the adversary can only copy or overwrite

Encode using Non-malleable codes

Step-2: Hybrid to Real compiler

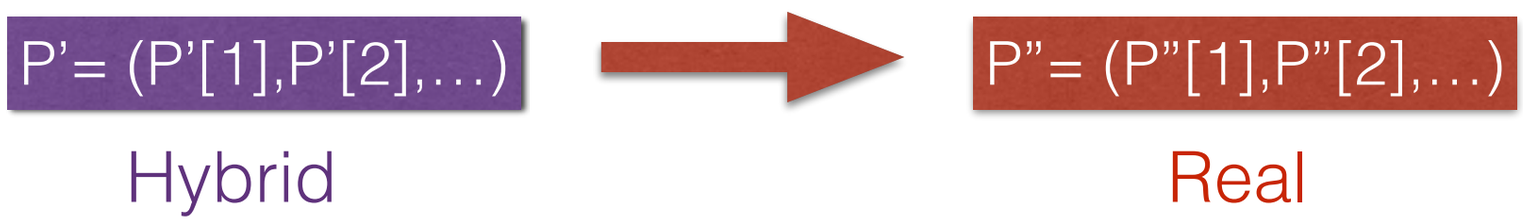


Idea: Encode each P'[i] such that the adversary can only copy or overwrite

Encode using Non-malleable codes

Guarantees that if the adversary does not copy the encoding then only can overwrite

Step-2: Hybrid to Real compiler



Idea: Encode each P'[i] such that the adversary can only copy or overwrite

Encode using Non-malleable codes

Guarantees that if the adversary does not copy the encoding then only can overwrite

Non-malleable Codes (DPW10)

An encoding (ENC, DEC) is non-malleable w.r.t. a function family $\mathcal{F}$ if the following holds

$$\forall m, \forall f \in \mathcal{F} \qquad c \leftarrow ENC(m) \qquad c' := f(c) \qquad m' := DEC(c')$$

Then m is equal or unrelated to m'

A suitable instantiation of Non-malleable codes

A suitable instantiation of Non-malleable codes

Since the same encoding can be tampered multiple times we will need continuous non-malleable codes

A suitable instantiation of Non-malleable codes

Since the same encoding can be tampered multiple times we will need continuous non-malleable codes

Continuous Non-malleable Codes: introduced by FMNV14

A suitable instantiation of Non-malleable codes

Since the same encoding can be tampered multiple times we will need continuous non-malleable codes

Continuous Non-malleable Codes: introduced by FMNV14

- Construction works for split-state $\mathcal{F}$: codeword has two parts which are independently temperable

A suitable instantiation of Non-malleable codes

Since the same encoding can be tampered multiple times we will need continuous non-malleable codes

Continuous Non-malleable Codes: introduced by FMNV14

- Construction works for split-state $\mathcal{F}$: codeword has two parts which are independently temperable
- Needs Common Random String

A suitable instantiation of Non-malleable codes

Since the same encoding can be tampered multiple times we will need continuous non-malleable codes

Continuous Non-malleable Codes: introduced by FMNV14

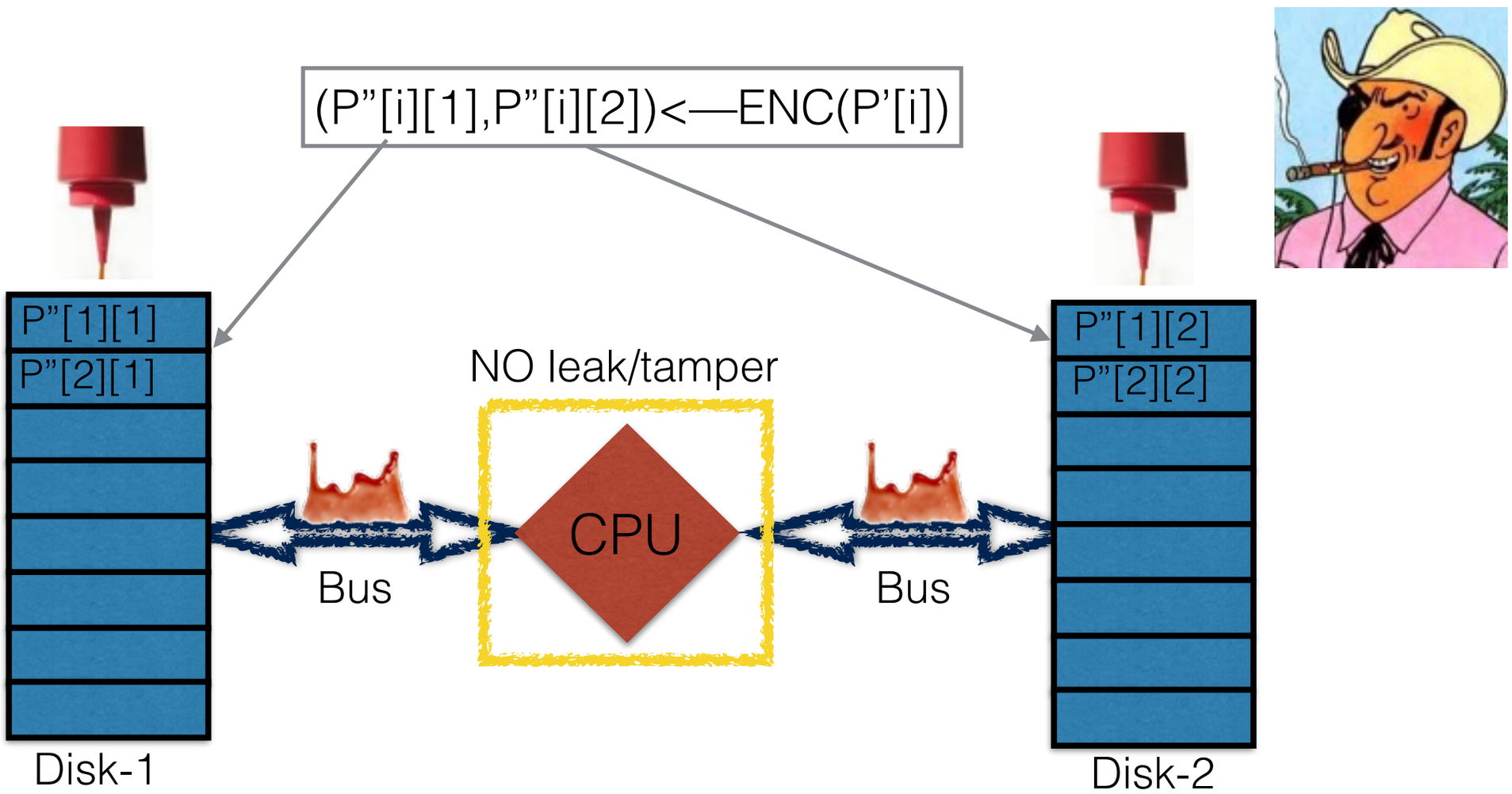
- Construction works for split-state $\mathcal{F}$: codeword has two parts which are independently temperable
- Needs Common Random String
- Needs self-destruct

Implementing vNA via FMNV14 code

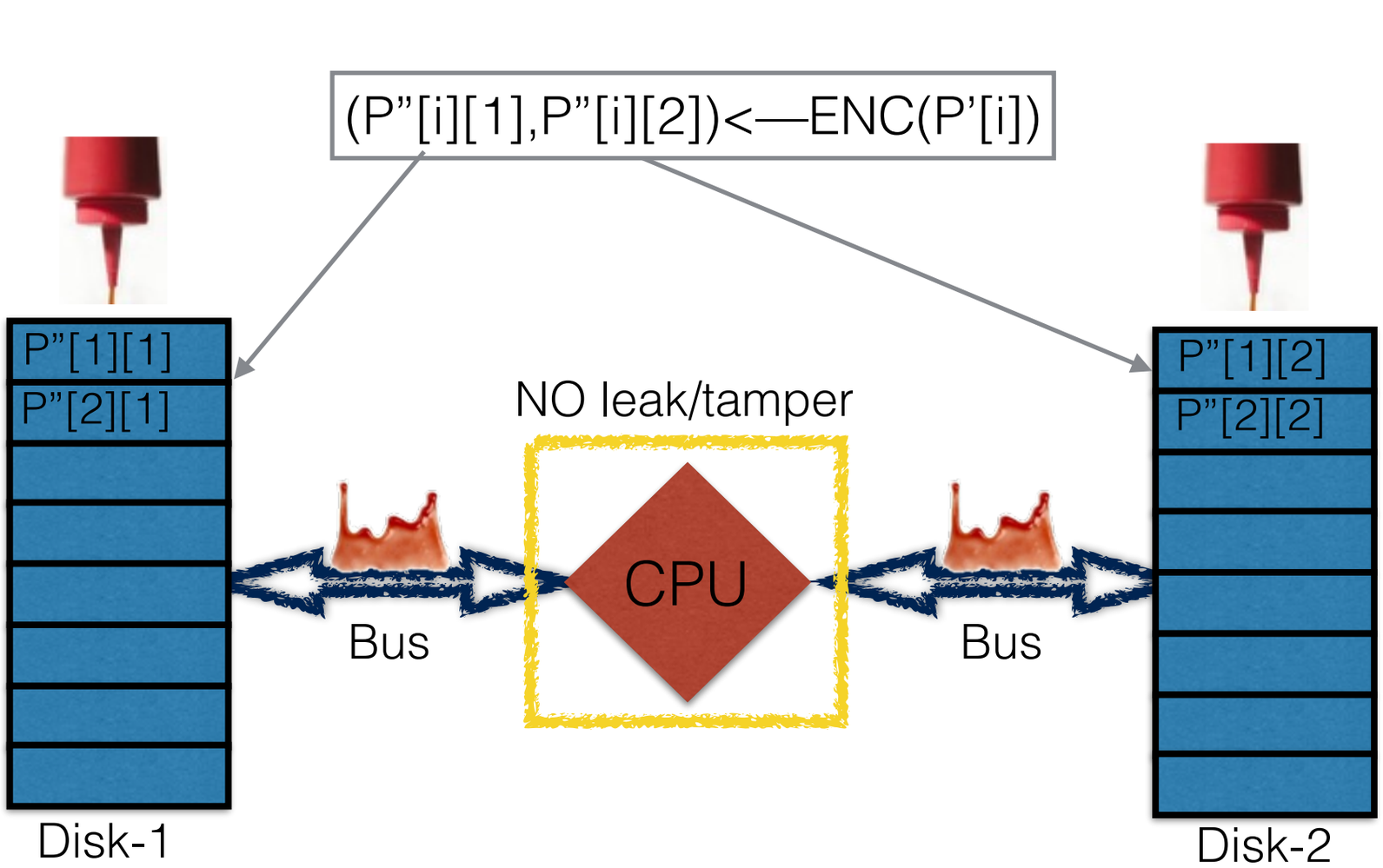
Implementing vNA via FMNV14 code

(P”[i][1],P”[i][2])<—ENC(P’[i])

Implementing vNA via FMNV14 code



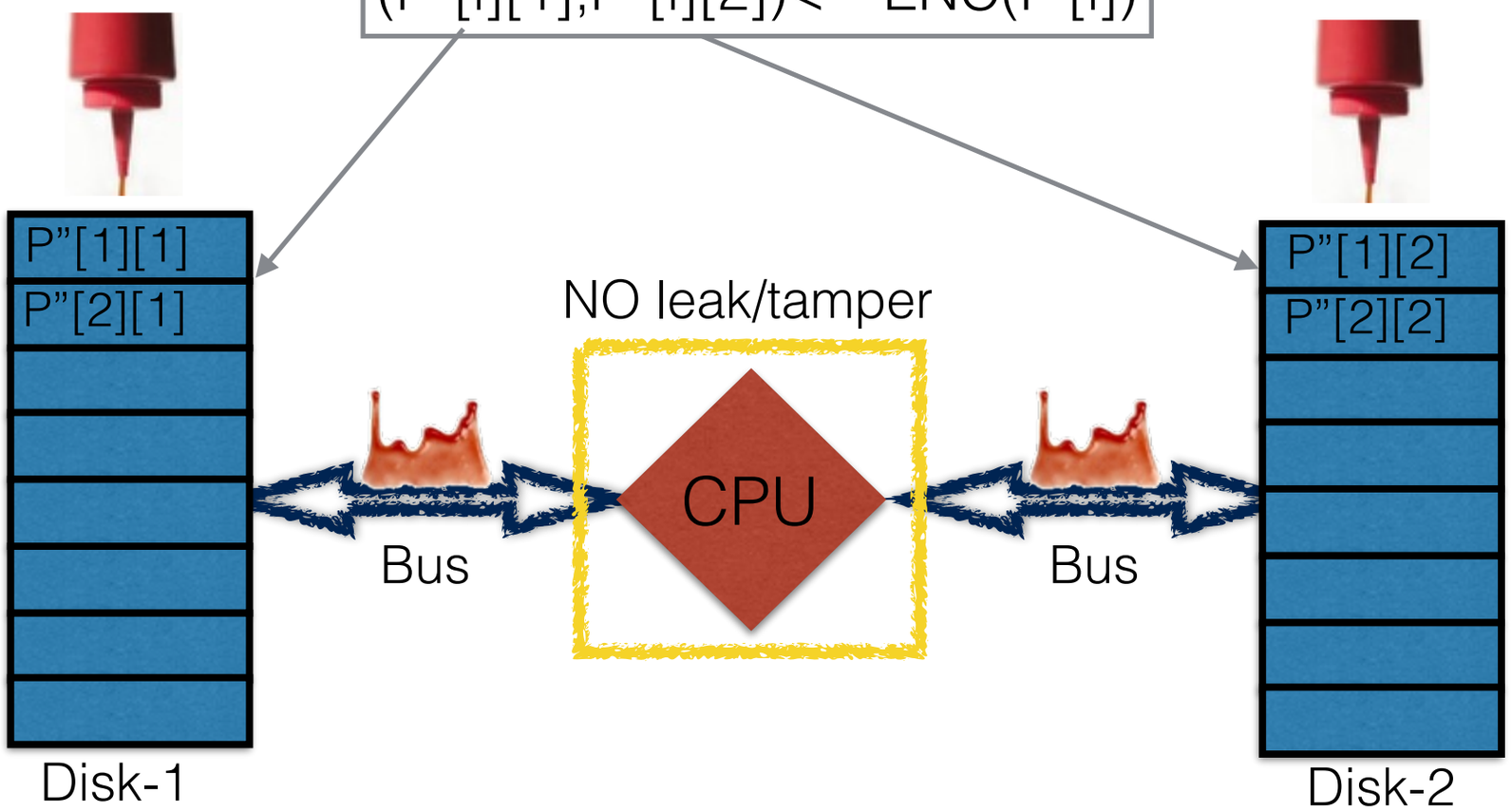
Implementing vNA via FMNV14 code



CPU

- CRS is part of description and hardwired.
- stores one untamperable special purpose bit.

Implementing vNA via FMNV14 code



CPU

- CRS is part of description and hardwired.
- stores one untamperable special purpose bit.

the self-destruct flag

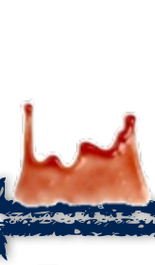
Implementing vNA via FMNV14 code

(P”[i][1],P”[i][2])<—ENC(P’[i])

P”[1][1]
P”[2][1]

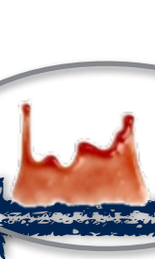
P”[1][2]
P”[2][2]

NO leak/tamper



Bus

CPU



Bus

- P’[i] accessed bounded no. of times.
- CNMC protect against bounded leakage

Disk-1

Disk-2

the self-destruct flag

CPU

- CRS is part of description and hardwired.
- stores one untamperable special purpose bit.

Conclusion

Conclusion

- We propose a new framework of protecting computations against tampering (and leakage).

Conclusion

- We propose a new framework of protecting computations against tampering (and leakage).
- Our architecture can withhold significantly stronger tampering: (split-state) than the circuit-oriented approach.

Conclusion

- We propose a new framework of protecting computations against tampering (and leakage).
- Our architecture can withhold significantly stronger tampering: (split-state) than the circuit-oriented approach.
- Since our transformation from Hybrid to Real uses the CNMC in blackbox way, better construction will improve the overall construction.

Conclusion

- We propose a new framework of protecting computations against tampering (and leakage).
- Our architecture can withhold significantly stronger tampering: (split-state) than the circuit-oriented approach.
- Since our transformation from Hybrid to Real uses the CNMC in blackbox way, better construction will improve the overall construction.
- Future direction:

Conclusion

- We propose a new framework of protecting computations against tampering (and leakage).
- Our architecture can withhold significantly stronger tampering: (split-state) than the circuit-oriented approach.
- Since our transformation from Hybrid to Real uses the CNMC in blackbox way, better construction will improve the overall construction.
- Future direction:
  - CPU size depends linearly on sec-param since it has to execute DECODE of CNMC. Can we get constant?

Thank You !