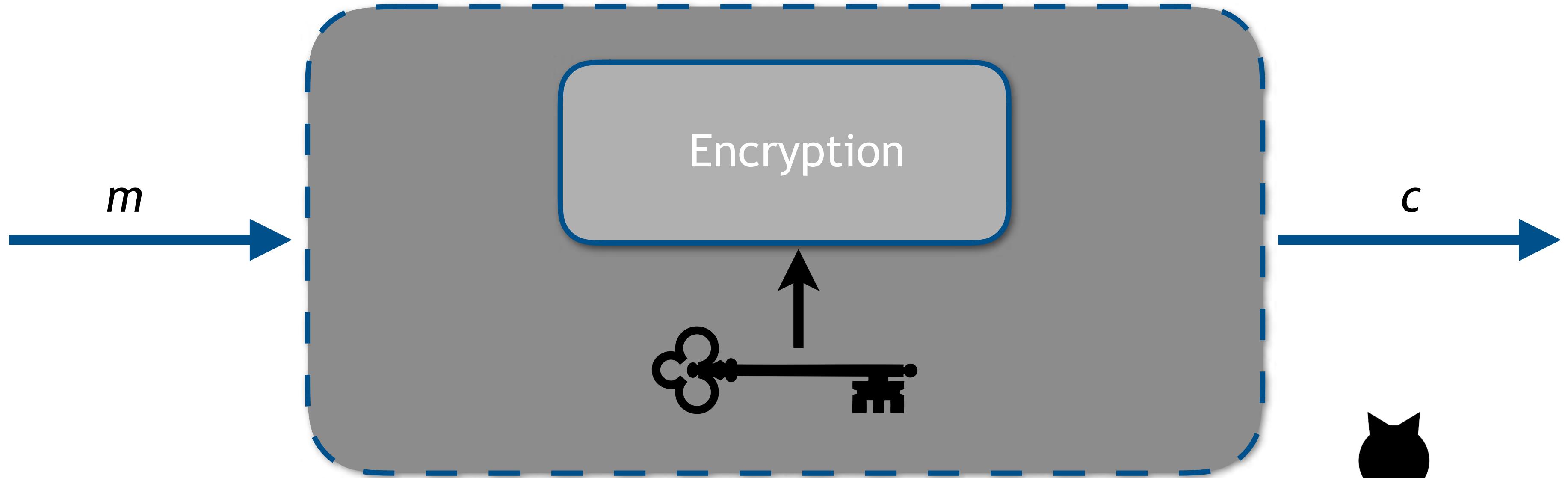
White-box Cryptography: Security Goals and Foundations

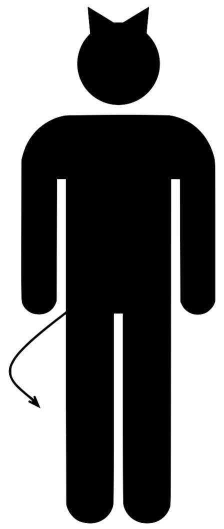
Estuardo Alpírez Bock and Chris Brzuska

Aalto University

ISC Winter School 2021

Grey-box attack scenario

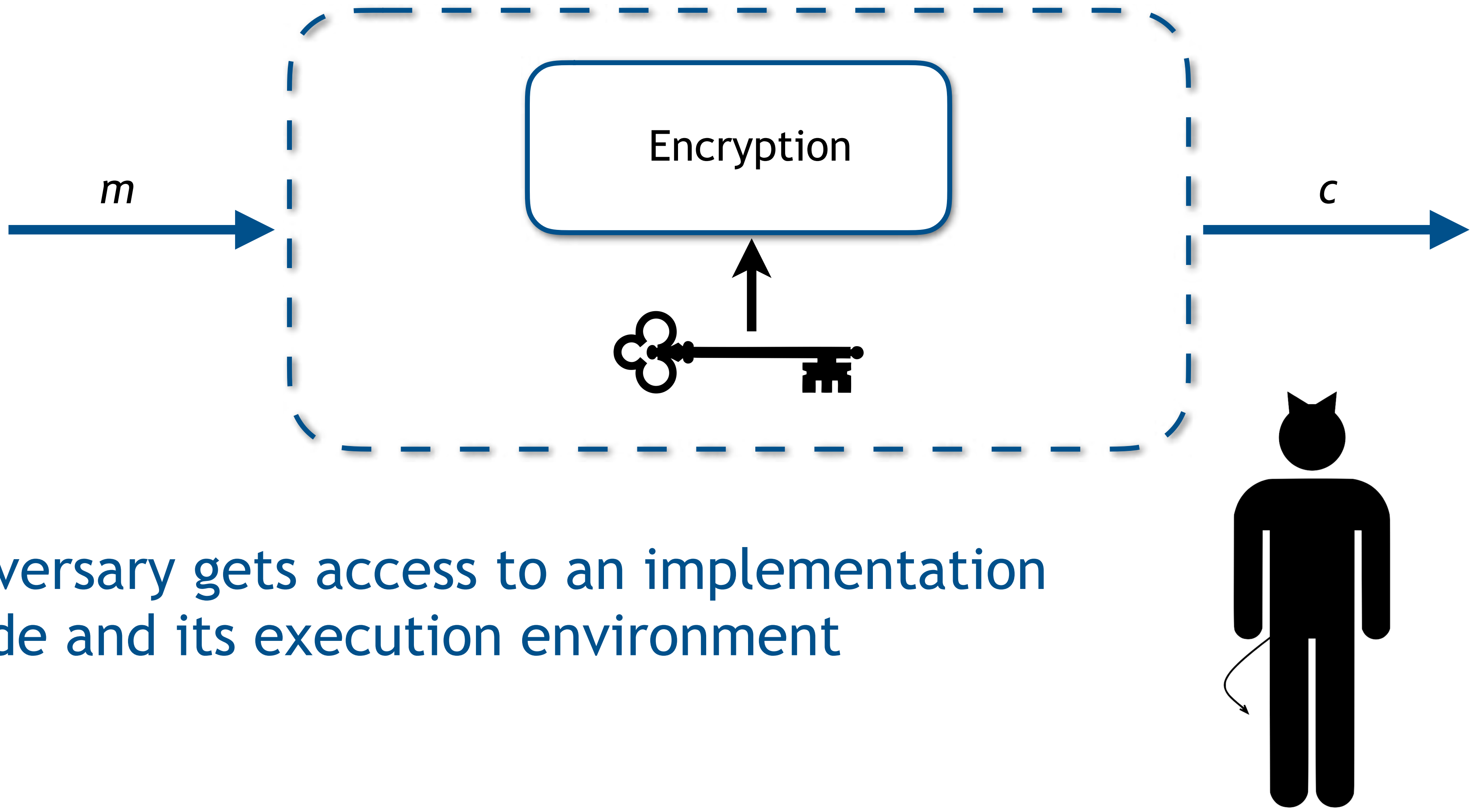




The adversary observes physical parameters generated by the device.

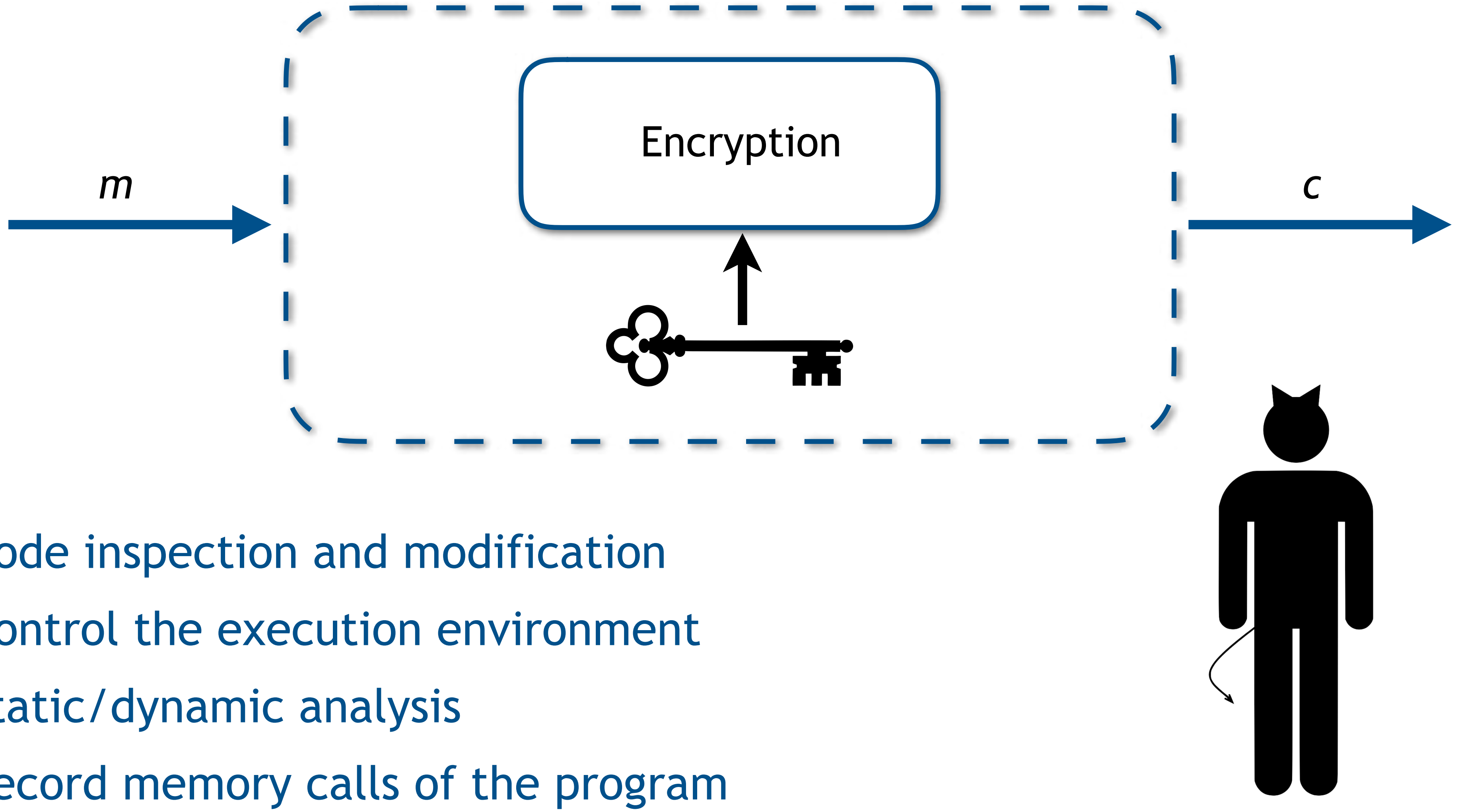
- Such physical parameters might show key-dependencies
- The adversary can perform statistical analyses on the parameters
- The adversary can also modify the hardware

White-box attack scenario



Adversary gets access to an implementation code and its execution environment

White-box attack scenario



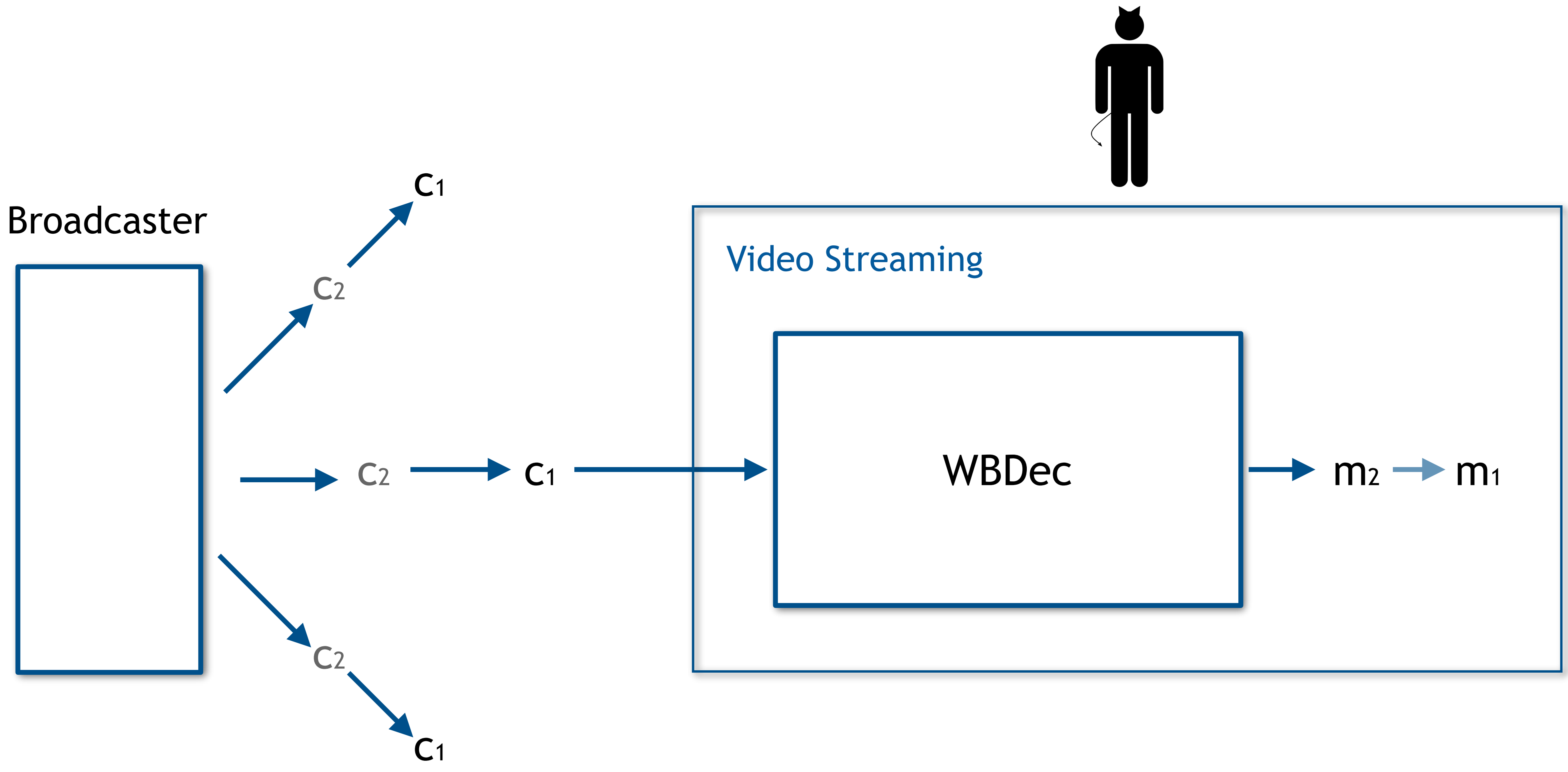
- Code inspection and modification
- Control the execution environment
- Static/dynamic analysis
- Record memory calls of the program

$\longrightarrow$ WB Cryptography aims to provide security even under such attack threats

Outline

Introduction and use cases

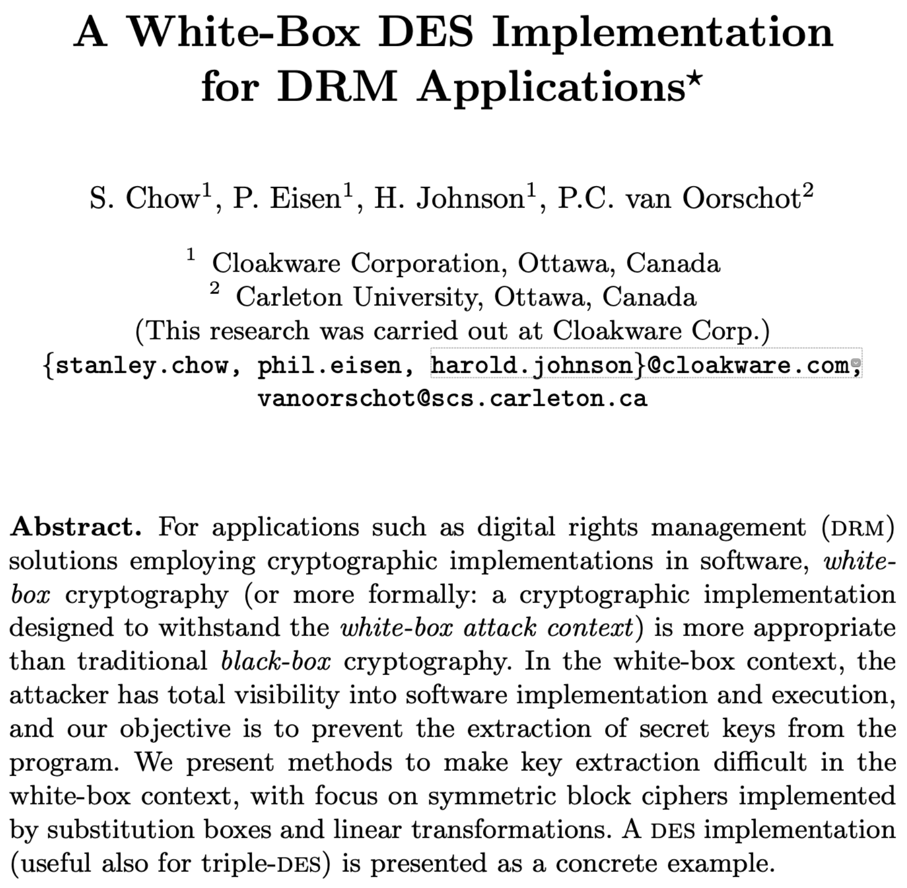
White-box crypto for DRM



- White-box crypto for mitigating piracy
- The owner of the application is considered to be the adversary

First white-box publications

- Chow, Eisen, Johnson, van Oorschot introduced the white-box attack scenario with two publications in 2002

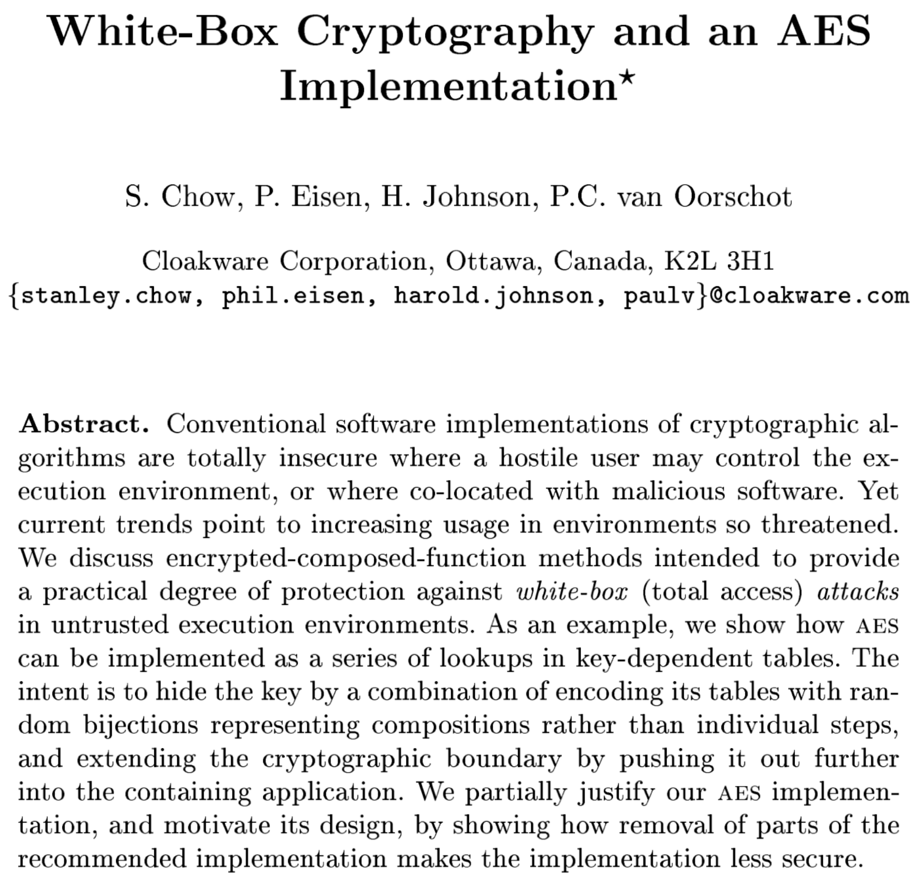


A White-Box DES Implementation for DRM Applications[⋆]

S. Chow[1], P. Eisen[1], H. Johnson[1], P.C. van Oorschot[2]

[1] Cloakware Corporation, Ottawa, Canada
[2] Carleton University, Ottawa, Canada
(This research was carried out at Cloakware Corp.)
{stanley.chow, phil.eisen, harold.johnson}@cloakware.com,
vanoorschot@scs.carleton.ca

Abstract. For applications such as digital rights management (DRM) solutions employing cryptographic implementations in software, *white-box* cryptography (or more formally: a cryptographic implementation designed to withstand the *white-box attack context*) is more appropriate than traditional *black-box* cryptography. In the white-box context, the attacker has total visibility into software implementation and execution, and our objective is to prevent the extraction of secret keys from the program. We present methods to make key extraction difficult in the white-box context, with focus on symmetric block ciphers implemented by substitution boxes and linear transformations. A DES implementation (useful also for triple-DES) is presented as a concrete example.



White-Box Cryptography and an AES Implementation[⋆]

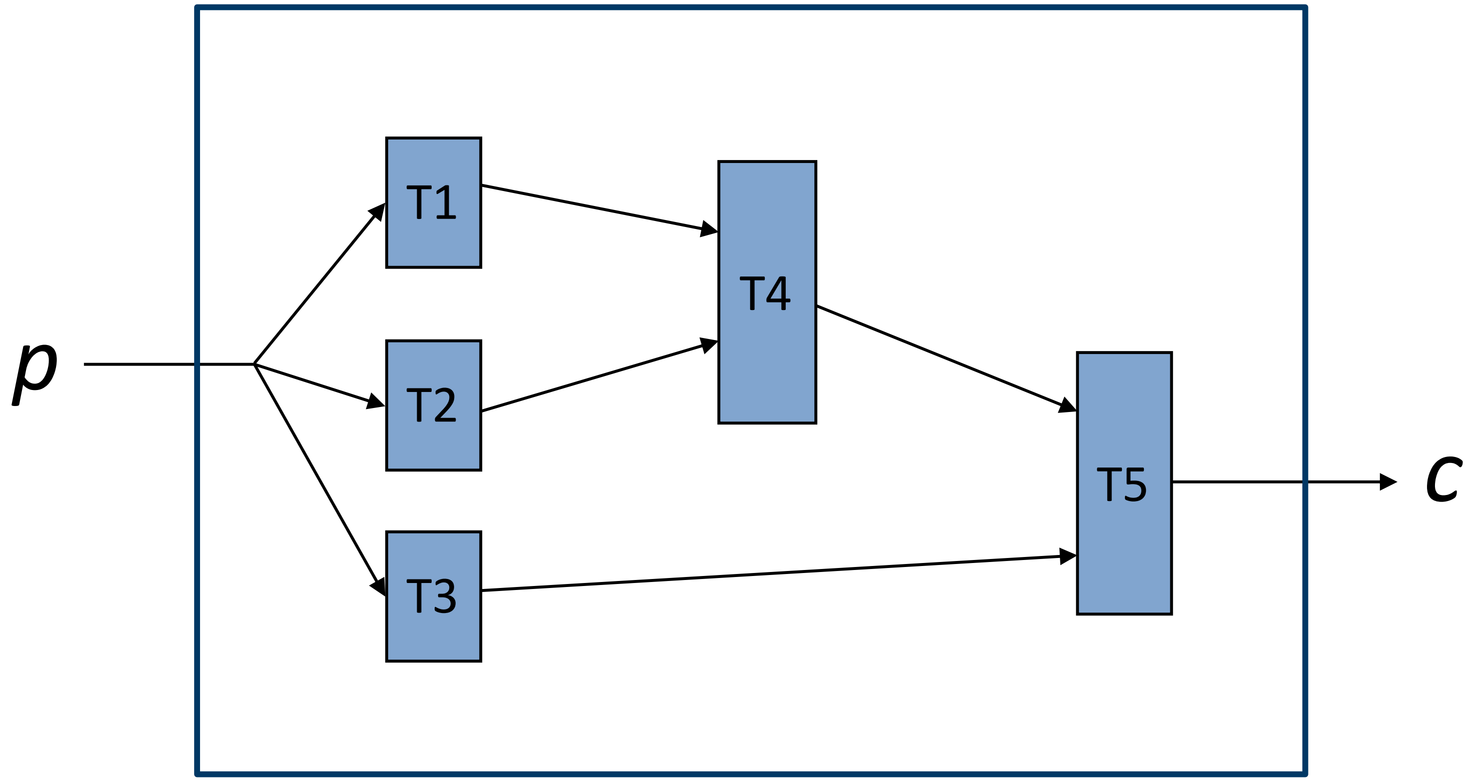
S. Chow, P. Eisen, H. Johnson, P.C. van Oorschot

Cloakware Corporation, Ottawa, Canada, K2L 3H1
{stanley.chow, phil.eisen, harold.johnson, paulv}@cloakware.com

Abstract. Conventional software implementations of cryptographic algorithms are totally insecure where a hostile user may control the execution environment, or where co-located with malicious software. Yet current trends point to increasing usage in environments so threatened. We discuss encrypted-composed-function methods intended to provide a practical degree of protection against *white-box* (total access) *attacks* in untrusted execution environments. As an example, we show how AES can be implemented as a series of lookups in key-dependent tables. The intent is to hide the key by a combination of encoding its tables with random bijections representing compositions rather than individual steps, and extending the cryptographic boundary by pushing it out further into the containing application. We partially justify our AES implementation, and motivate its design, by showing how removal of parts of the recommended implementation makes the implementation less secure.

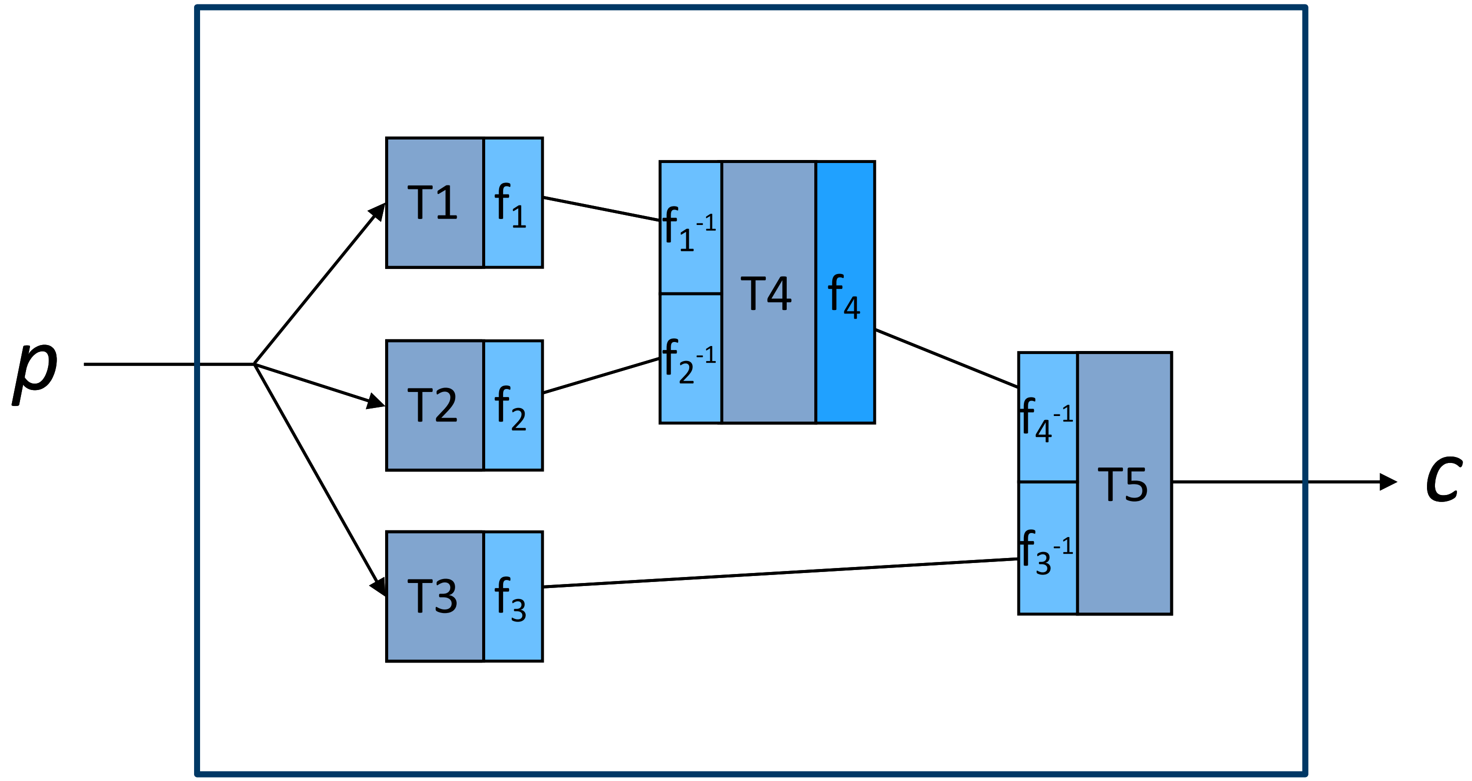
Chow et al.'s framework

- Presented approach: implement the AES (or DES) as a network of key-dependent look-up tables
- Each look-up table corresponds to a step in the algorithm



Chow et al.'s framework

- The contents of the look-up table can be *obscured* via randomised encodings



Chow et al.'s framework

Chow et al.'s framework presents a relatively efficient method for obfuscating AES and DES designs

The framework was embraced by the industry

Attacks have been published on this framework and it is probably not implemented exactly as described in the literature, but rather on variant ways

[1] Chow, Eisen, Johnson, van Oorschot: A white-box DES implementation for DRM applications, ACM 2002

[2] Chow, Eisen, Johnson, van Oorschot: White-box cryptography and an AES implementation, SAC 2002

White-box crypto for payment applications

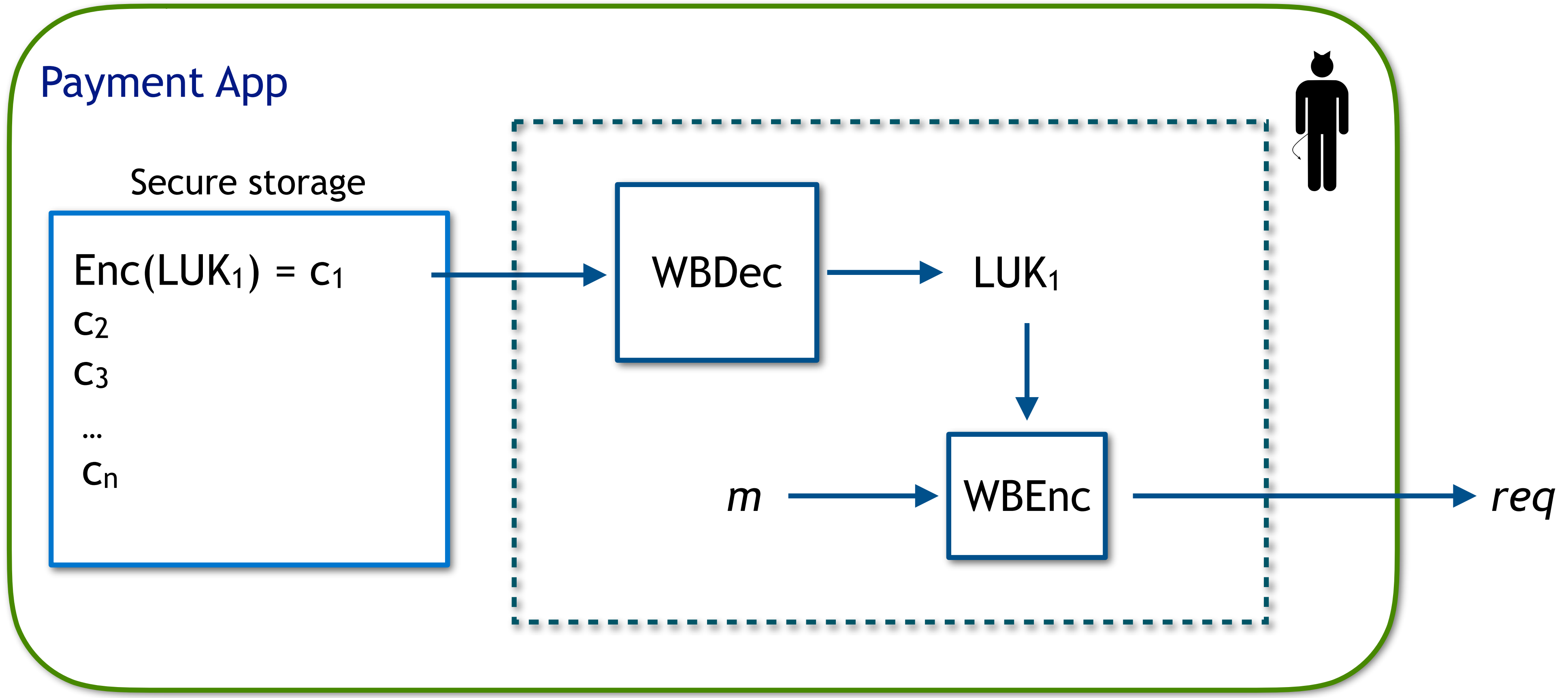
- In 2015, Android introduced host card emulation (HCE), which allows for software applications running on a phone’s CPU to communicate via Near Field Communication (NFC)
- White-box crypto was proposed as a software countermeasure technique to help protect mobile payment applications implemented in software [3,4]
- At this point, white-box crypto re-gained some popularity in the scientific community

[3] Smart Card Alliance Mobile and NFC Council. Host card emulation 101. white paper, 2014

[4] Emv mobile payment: Software-based mobile payment security requirements, 2019

White-box crypto for payment applications

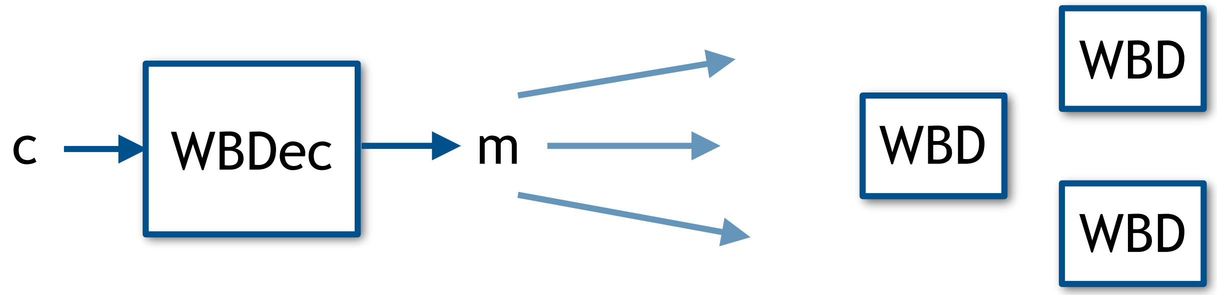
- Limited use keys (LUKs) used for encrypting a transaction request message



Security goals and notions

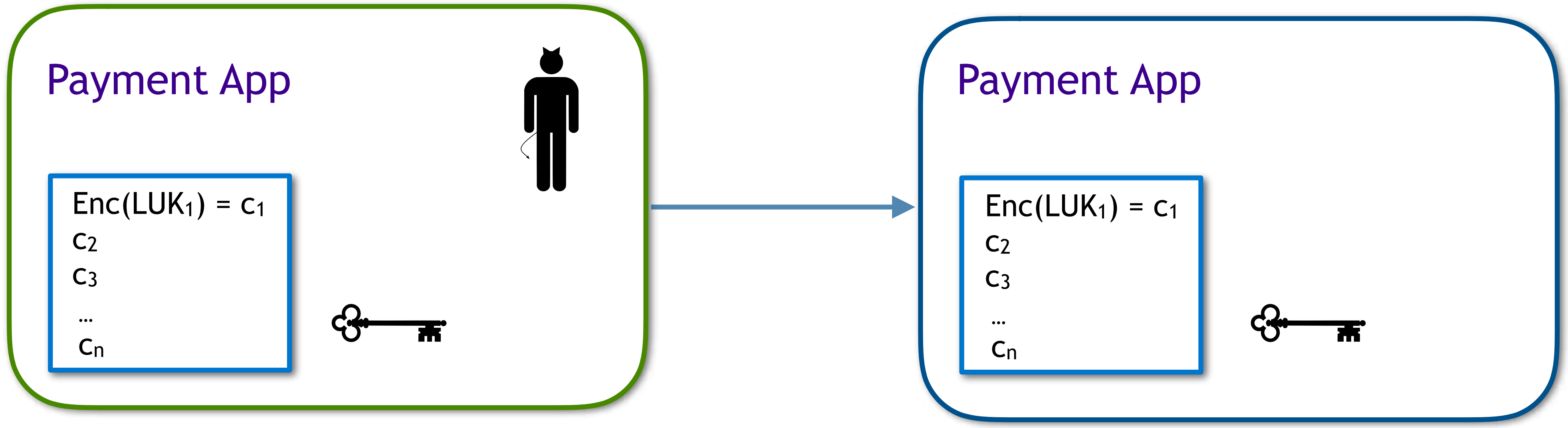
What are the goals of white-box crypto?

- Depending who we ask, the goal might be:
  - Hiding the key of a cipher (special purpose obfuscation)
    - Given access to implementation code, key extraction is a big threat
  - Hiding the key of an AES implementation (special purpose obfuscation)
    - Opinion motivated by the popular goal of white-boxing AES (Popularity of AES, first white-box paper by Chow et al., WhibOx competitions, etc.)
  - Mitigate redistribution attacks
    - Motivated by the use case of white-box crypto in DRM applications



White-box crypto for payment applications

- An adversary can copy the app and run it at a phone and terminal of its choice



We need protection against *code-lifting* attacks

Security notions

- Security notions for white-box crypto have been introduced in the literature
  - Motivated by the DRM use case [5,6] and
  - Mobile payment applications [7,8]

- [5] presents basic properties such as security against key extraction and one-wayness, but also presents notions in light of code-lifting attacks

[5] Delerablée, Lepoint, Paillier, Rivain - White-box security notions for symmetric encryption schemes, SAC 2013
[6] Fouque, Karpman, Kirchner, Minaud - Efficient and provable white-box primitives, ASIACRYPT 2016
[7] Alpirez Bock, Amadori, Brzuska, Michiels - On the security goals of white-box cryptography, CHES 2020
[8] Alpirez Bock, Brzuska, Fischlin, Janson, Michiels: Security reductions for white-box key storage in mobile payments, Asiacrypt 2020

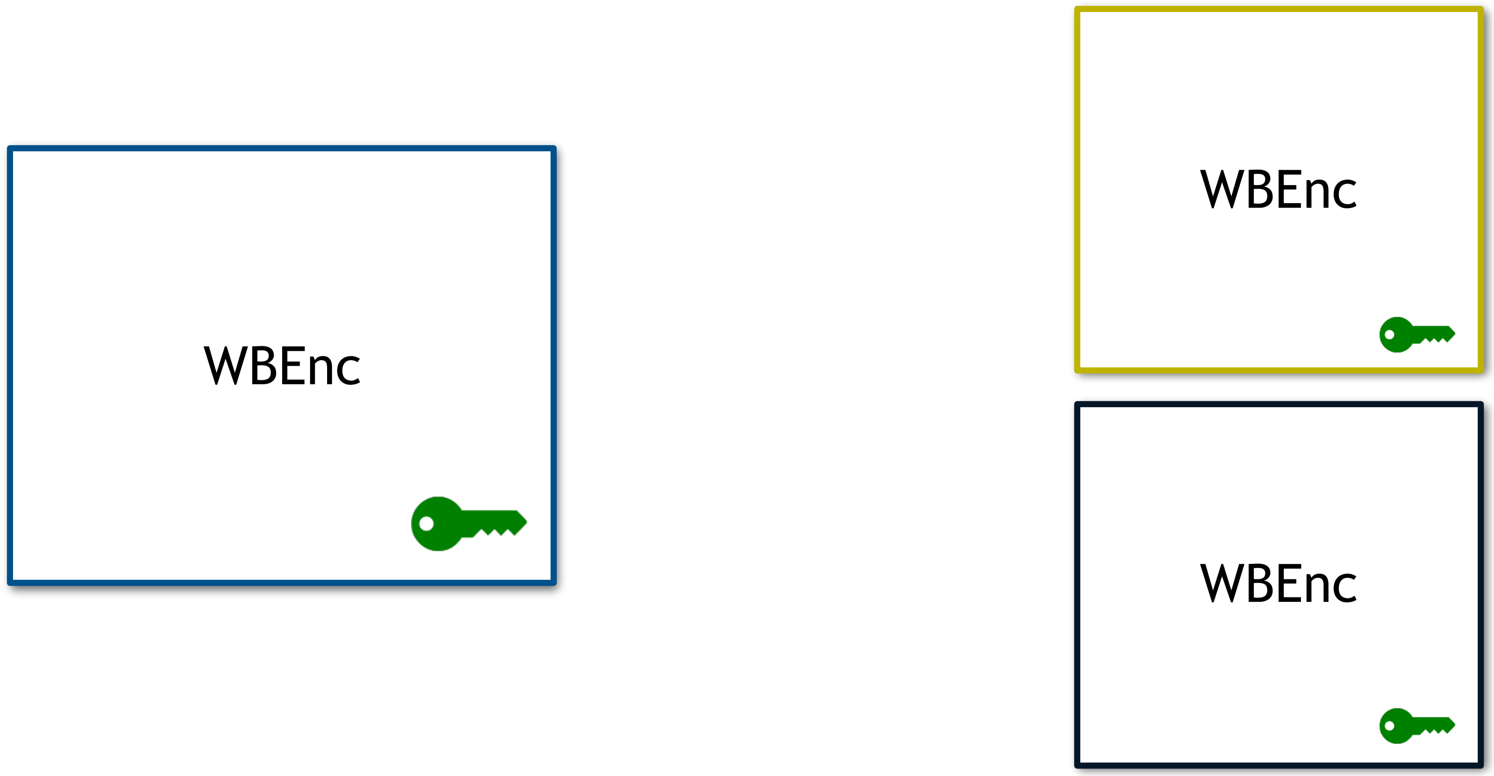
Popular notions and mitigation techniques

- The properties of *traceability* and *incompressibility* were considered in the early works

- Security notions and constructions have been proposed e.g. in:
  - Delerablée, Lepoint, Paillier, Rivain - White-box security notions for symmetric encryption schemes, SAC 2013
  - Fouque, Karpman, Kirchner, Minaud - Efficient and provable white-box primitives, ASIACRYPT 2016
  - Bogdanov, Isobe, Tischhauser - Towards practical white box cryptography: optimizing efficiency and space hardness, ASIACRYPT 2016
  - Alpirez Bock, Amadori, Bos, Brzuska, Michiels - Doubly half-injective PRGs for incompressible white-box cryptography, CT-RSA 2019
  - Alex Biryukov - White-box and asymmetrically hard crypto design, WhibOx 2019 Workshop

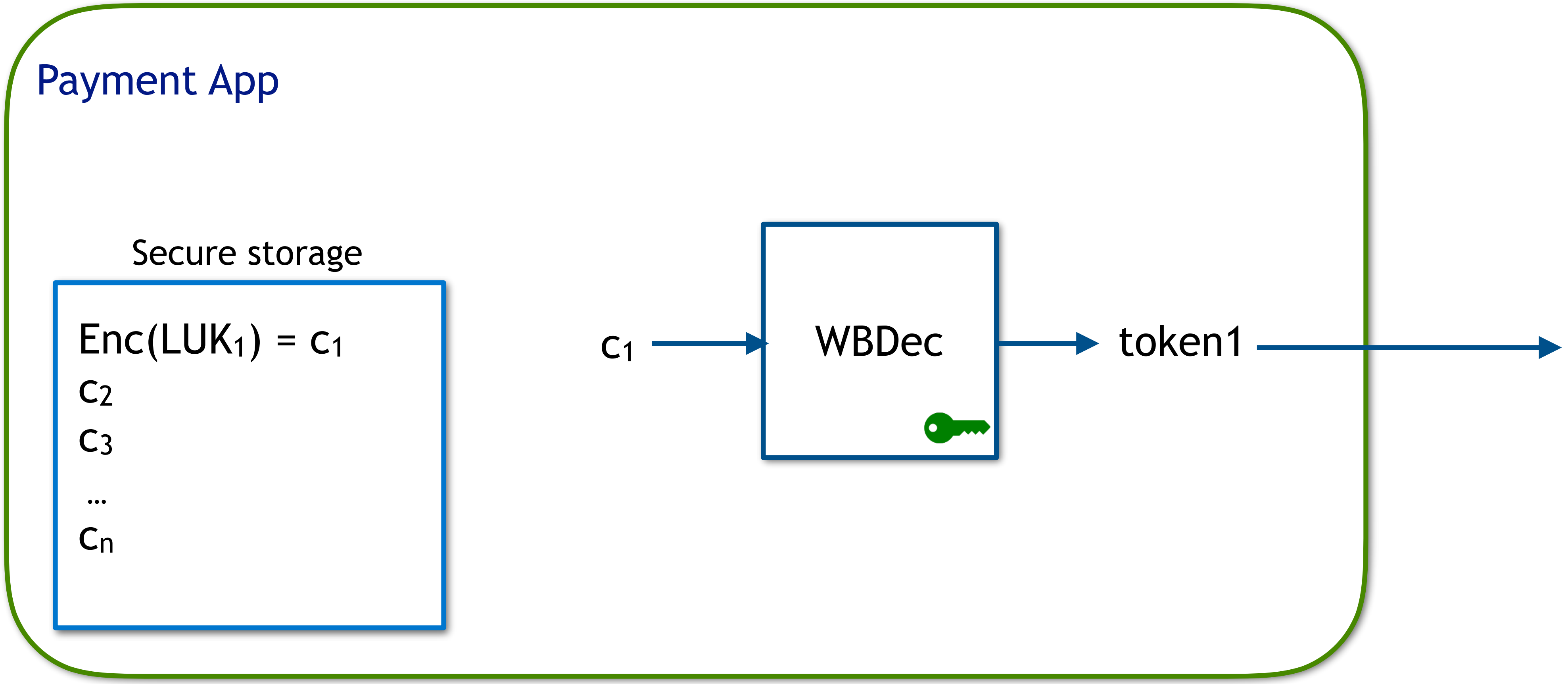
Traceability

- A white-box program is watermarked with a *tracing key*. Each program has its own tracing key.



The tracing key helps identify the origin of the copied program

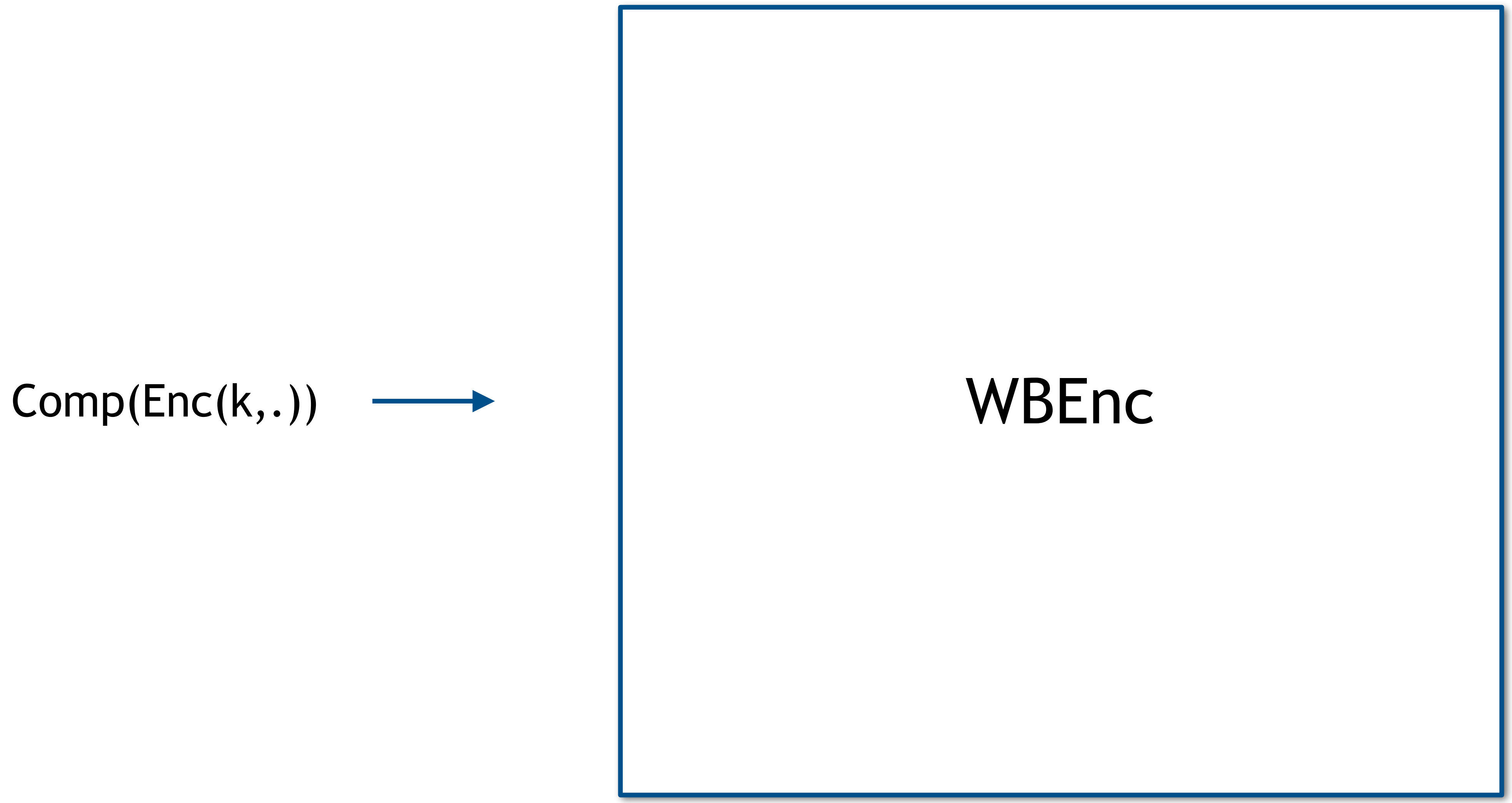
Traceability



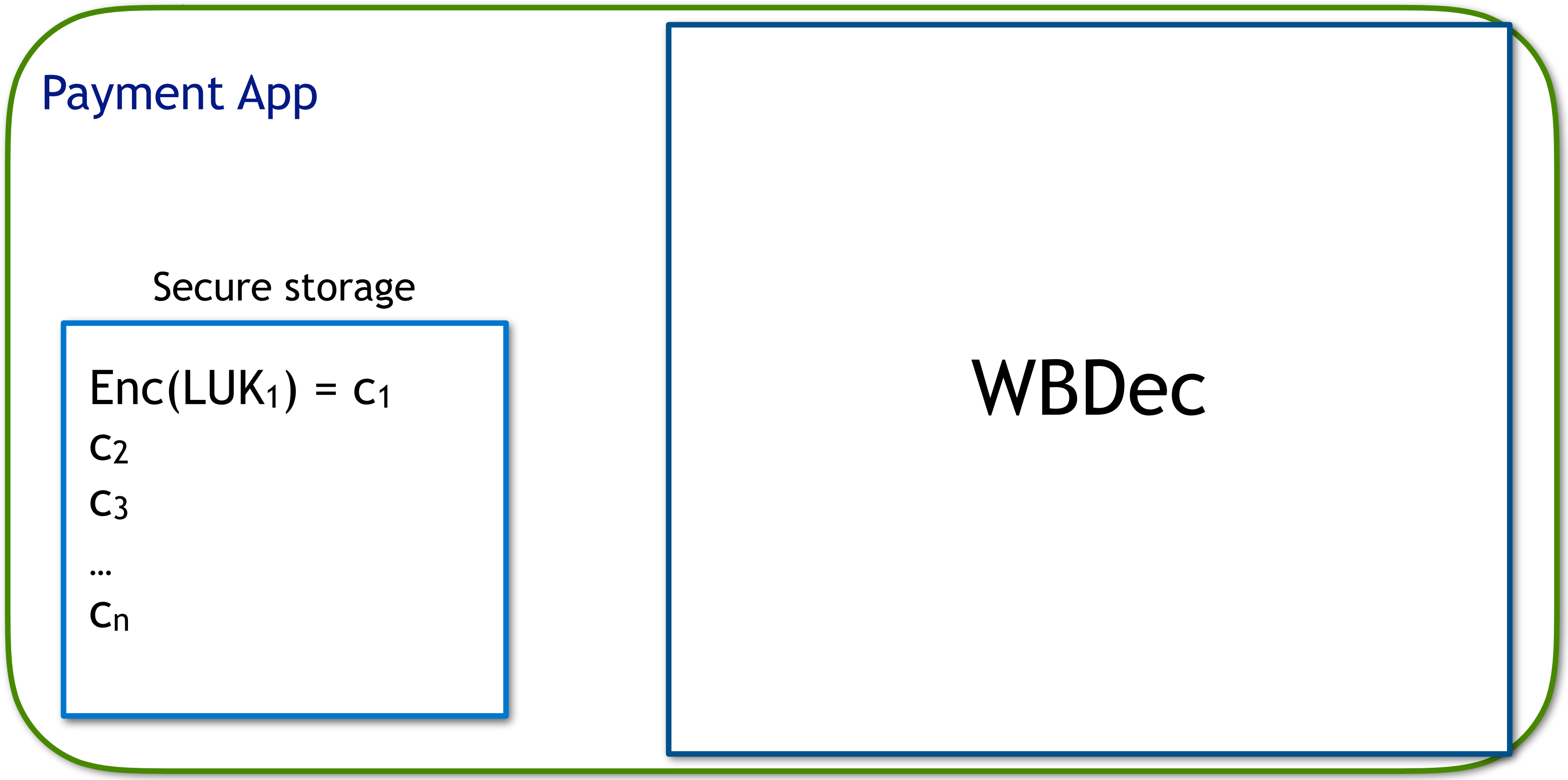
The owner of a payment application will not make copies of it and share it
This would enable people to access the user’s keys, i.e. my money.

Incompressibility

- Make a program very large in size. If the program is compressed or fragments are removed, the program loses its functionality.



Incompressibility



Large programs take too much space from a mobile application - contrast to IoT

Large programs are also difficult to distribute *legally*

Incompressibility and traceability

Theoretical constructions can be achieved under nice assumptions (standard, public-key type, etc.)

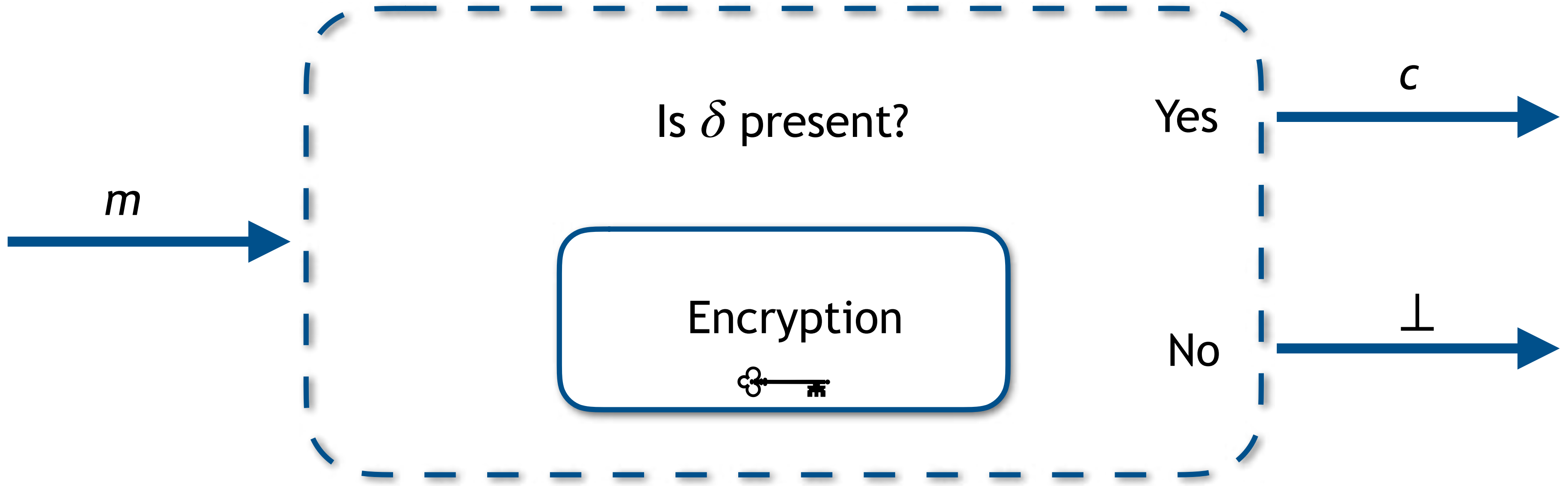
Traceability is useful in the DRM setting, but not in the mobile payment setting

Incompressibility is not really used in practice

- In practice we need more efficient techniques for mitigating code-lifting
- Specially for applications running on mobile phones and IoT
- Incompressibility has some interesting links to the bounded retrieval model

Alternative: hardware-binding

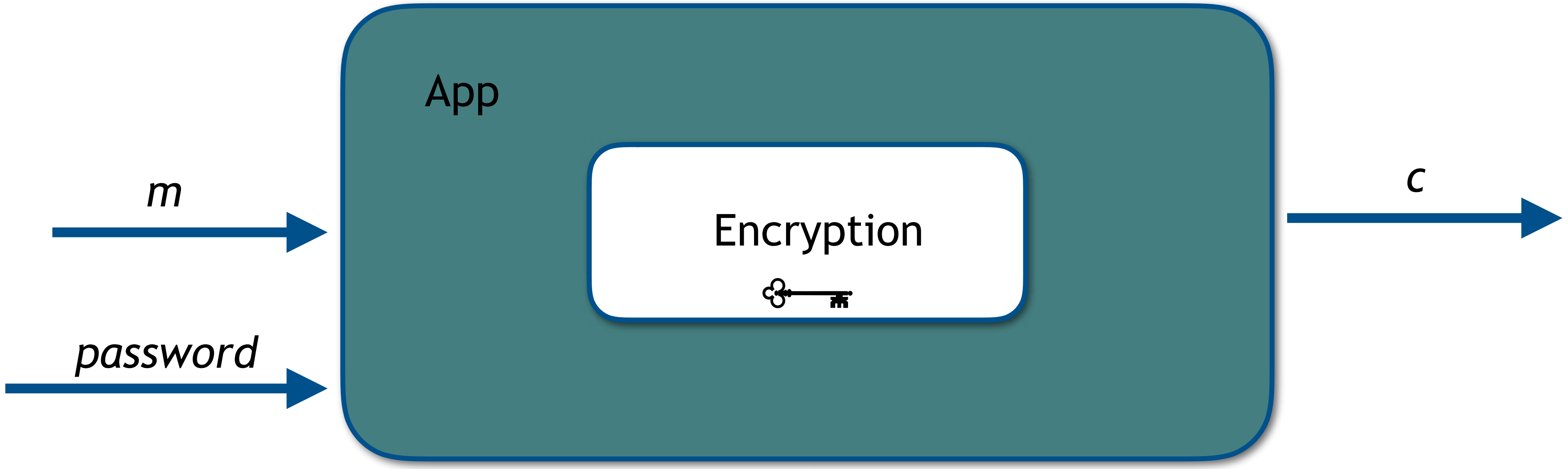
- An encryption program should only be executable on one specific device. The execution is dependable on a unique hardware identifier δ.



Alternative: application-binding

- **An encryption program should only be executable within one specific application**

Useful in the case that the application performs authentication operations



Defining Hardware-binding

Security notions for white-box crypto with hardware-binding have been presented for white-box KDFs, white-box payment applications and white-box encryption programs [7,8]

We obtain feasibility results from indistinguishability obfuscation and puncturable PRFs

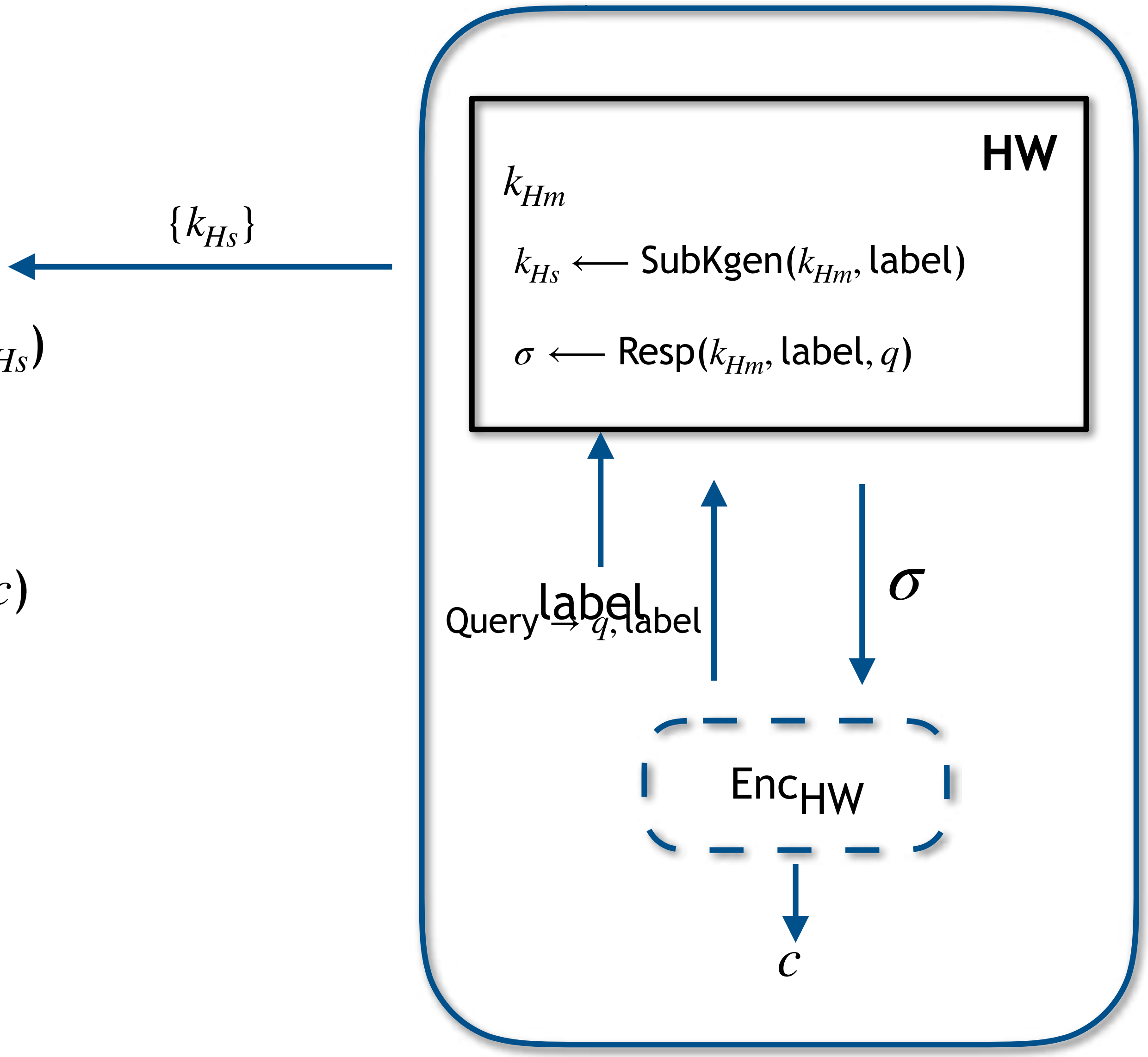
All definitions capture the property that an adversary is unable to use the white-box program without the corresponding hardware

Defining application binding presents several challenges

[7] Alpirez Bock, Amadori, Brzuska, Michiels - On the security goals of white-box cryptography, CHES 2020

[8] Alpirez Bock, Brzuska, Fischlin, Janson, Michiels: Security reductions for white-box key storage in mobile payments, Asiacrypt 2020

Hardware module



$\{k_{Hs}\}$
HW
k_{Hm}
$k_{Hs} \longleftarrow$ SubKgen(k_{Hm}, label)
$\sigma \longleftarrow$ Resp(k_{Hm}, label, q)
label
Query $\rightarrow q$, label
σ
Enc$_{\text{HW}}$
c

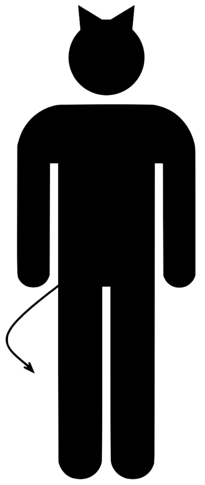
Query, Enc$_{\text{HW}}$ $\longleftarrow$ \$Comp(k, k_{Hs})

$q \leftarrow$ Query(m, nc)

Enc$_{\text{HW}}$(m, nc, σ) = Enc(k, m, nc)

```
Enc_HW($m, nc, \sigma$)
$b \longleftarrow$ Check($k_{Hs}, q, \sigma$)
if $b = 0$
        return $\perp$
else $c \longleftarrow$ Enc($k, m, nc$)
return $c$
```

Security of White-box encryption



$\mathrm{Enc}_{\mathrm{HW}}$

Query

$\xrightarrow{q}$

$\xleftarrow{\sigma}$

```
HW(q)
  assert $q \notin Q$
  $Q := Q \cup \{q\}$
  $\sigma \longleftarrow \mathrm{Resp}(k_{Hm}, \mathrm{label}, q)$
```

$\xrightarrow{m}$

$\xleftarrow{(c, nc)}$

```
Enc(m)
  $m_0 \leftarrow m, m_1 \leftarrow 0^{|m|}$
  $nc \longleftarrow \$\{0,1\}^n$
  assert $q_i \notin Q$
    with $q_i \leftarrow \mathrm{Query}(m_i, nc)$
  $Q := Q \cup \{q_i\}$
  $c \longleftarrow \mathrm{enc}(k, m_b, nc)$
  $C := C \cup \{(c, nc)\}$
```

$\xrightarrow{(c, nc)}$

$\xleftarrow{m}$

```
Dec(c)
  assert $(c, nc) \notin C$
  $m \leftarrow \mathrm{dec}(k, c, nc)$
  assert $q \notin Q$
      with $q \leftarrow \mathrm{Query}(m, nc)$

  if $b = 1$    $m \leftarrow \perp$
  else return $m$
```

Challenges defining application-binding

- What exactly is an application?

- Alternative: focus on specific applications, e.g. applications performing authentication operations:
  - A user authenticates himself via passwords or fingerprints. However, such values can be intercepted by a white-box adversary
    - Alternative: weaken the attack model. However, this leads to the following issues:
      - Presents an inconsistent attack scenario
      - In order to define security, we need to consider long enough secret authentication values. In that case, we could even consider a keyless white-box implementation

White-box implementations

White-box implementations

White-boxes used in practice usually follow the table-based approach, implementing some countermeasures against known attacks

Known attacks are based on:

Reverse engineering and recovering encoding

e.g. BGE attack on Chow et al.'s framework [9]

Side-channel-inspired approaches to exploit key dependencies from computational traces [10]

No openly known designs are resistant against key-extraction attacks

[9] Billet, Gilbert, Ech-Catbi: Cryptanalysis of a white-box AES implementation, Asiacrypt 2003

[10]Alpirez Bock, Bos, Brzuska, Hubain, Michiels, Mune, Sanfelix Gonzalez, Teuwen, Treff: White-box cryptography: don't forget about gray-box attacks, J. Of Cryptology 2019

But it’s not all bad news

CHES CTF Challenge

CHES 2017 Capture the Flag Challenge

The WhibOx Contest

An ECRYPT White-Box Cryptography Competition

WB competitions

Designers were invited to submit white-box implementations of AES-128

Implementation language must be C, without includes, libraries, etc

Size and runtime restrictions:

Source code	≤	50MB
Binary	≤	20MB
Runtime	≤	1s

Attackers were invited to break the implementations.

The longer an implementation remained unbroken, the more points it got

Competition results

Submissions:

94 design candidates were submitted
13 remained unbroken for at least 24 hours and earned > 0 points

All broken

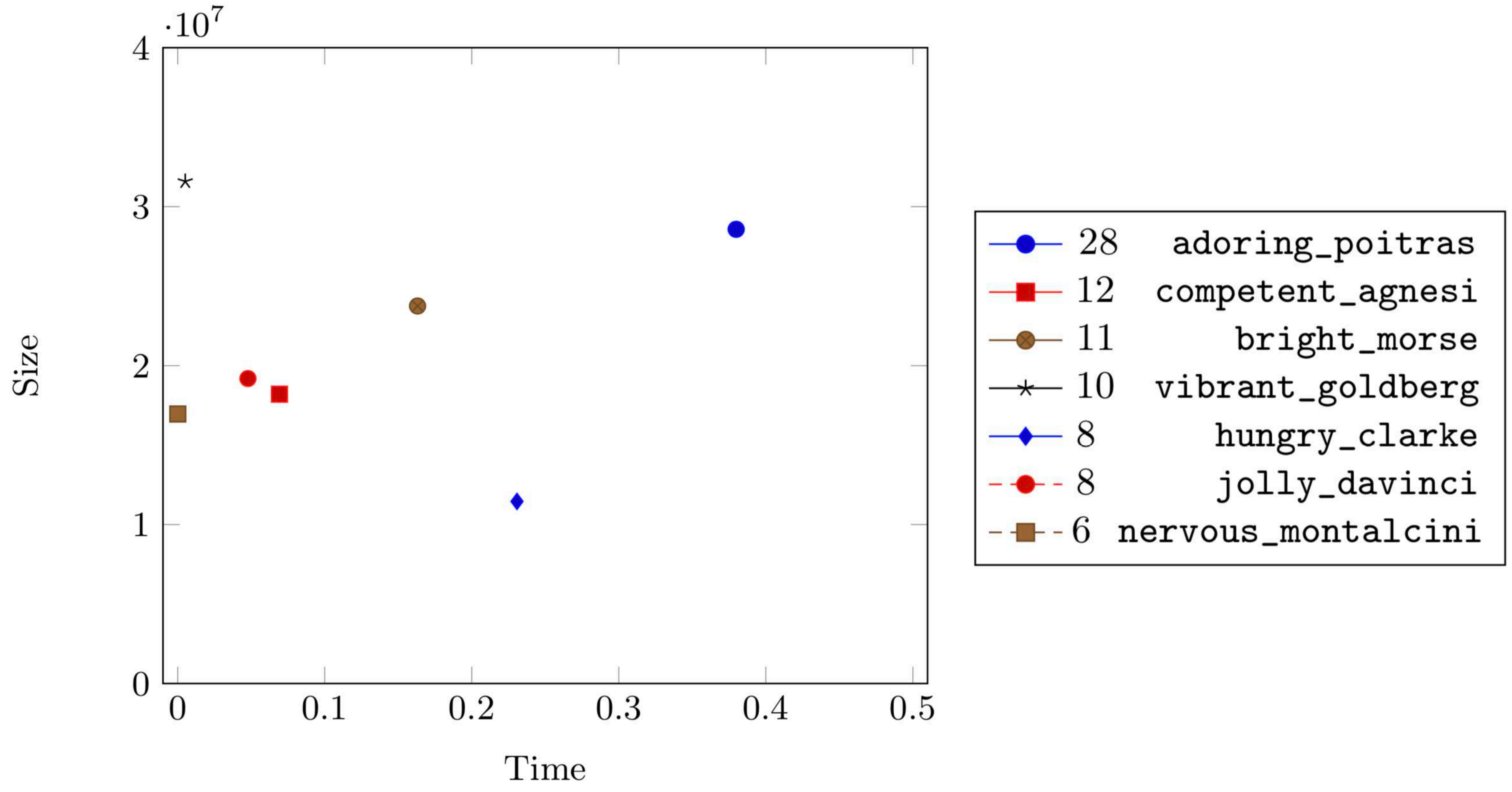
Winning challenge: adoring_poitras by Alex Biryukov and Aleksei Udovenko from the University of Luxembourg

Remained unbroken for 28 days

Broken by the CryptoExperts team [11]

[11] Goubin, Paillier, Rivain, Wang: How to reveal the secrets of an obscure white-box implementation, J. of Cryptographic Engineering

Top 8 challenges



Size in bytes / Time in seconds

2019 edition

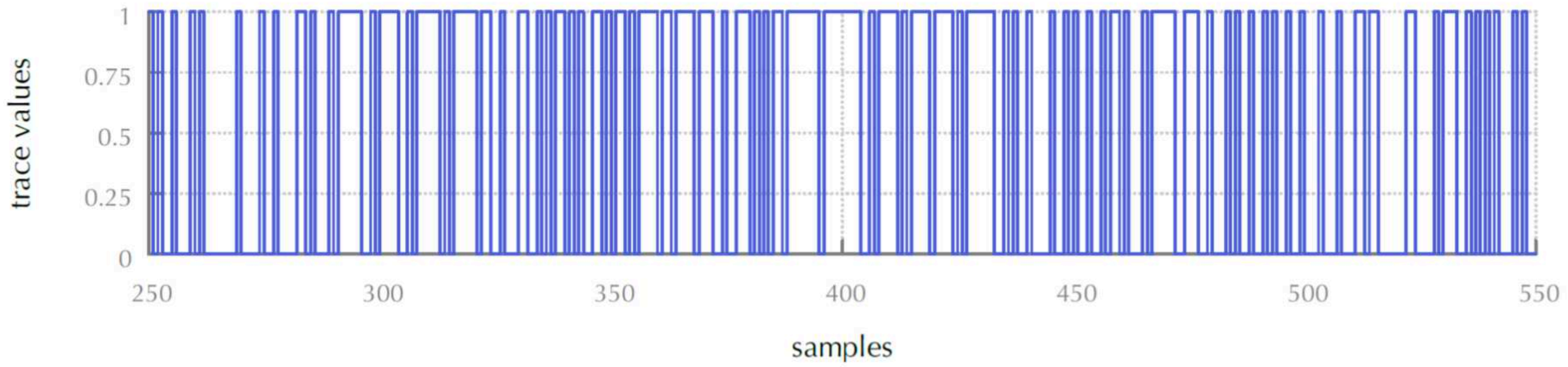
—> winning challenge remained unbroken for 51 days

—> 2 other challenges remained unbroken for 50 and 30 days

Automated attacks

Differential Computation Analysis

- Automated and efficient attack on white-box implementations presented by Bos et al. [1] and Sanfelix et al. [2]
- Records the memory addresses accessed during the encryption process and obtains *software execution traces*

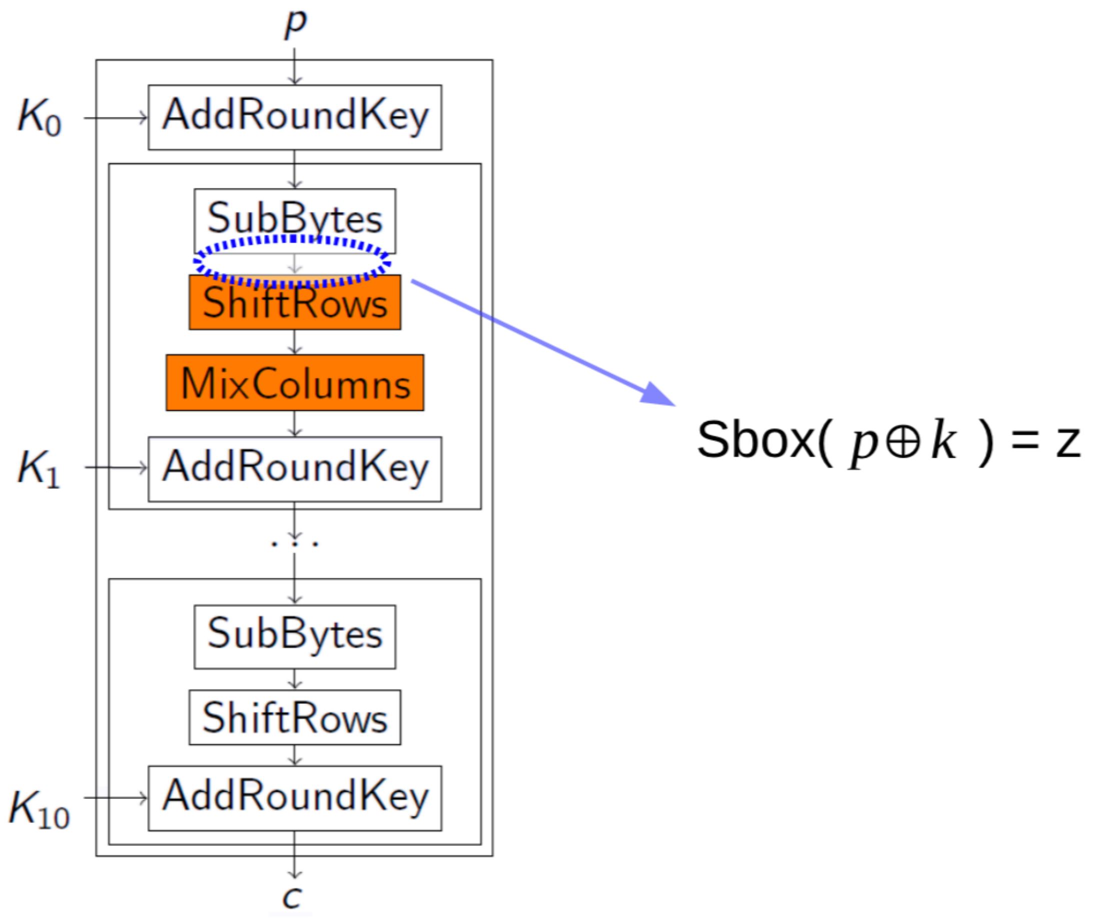


- Software traces can be analysed with traditional DPA tools

[1] J. W. Bos, C. Hubain, W. Michiels, and P. Teuwen: *Differential Computation Analysis: Hiding your White-Box Designs is Not Enough*. **CHES 2016**.
[2] E. Sanfelix, C. Mune, J. de Haas: *Unboxing the White-Box: Practical Attacks Against Obfuscated Ciphers*. **Black Hat Europe 2015.**

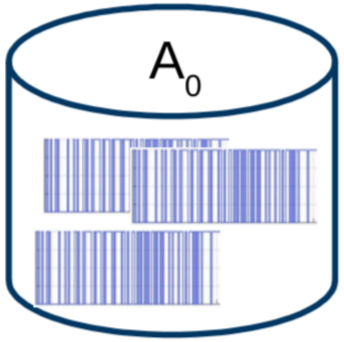
Differential Computation Analysis

1. Encrypt *n* plaintexts and record one software trace by each encryption
2. Define a *selection function sel = z[b]* ∈ {0,1} where *z* is an intermediate value calculated based on the known plaintext p_i and a key guess k^h



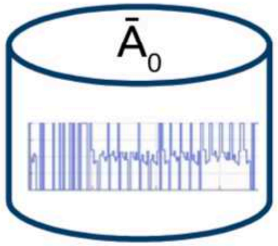
Differential Computation Analysis

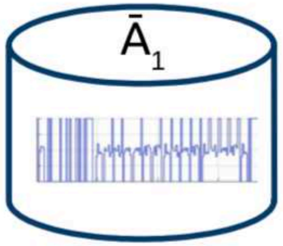
1. Encrypt *n* plaintexts and record one software trace by each encryption
2. Define a *selection function* *sel = z[b]* ∈ {0,1} where *z* is an intermediate value calculated based on the known plaintext p_i and a key guess k^h
3. For each plaintext p_i, calculate $sel(p_i,k^h)=b$ and sort each software trace s_i in the set $\boldsymbol{A_b}$, with b ∈ {0,1}



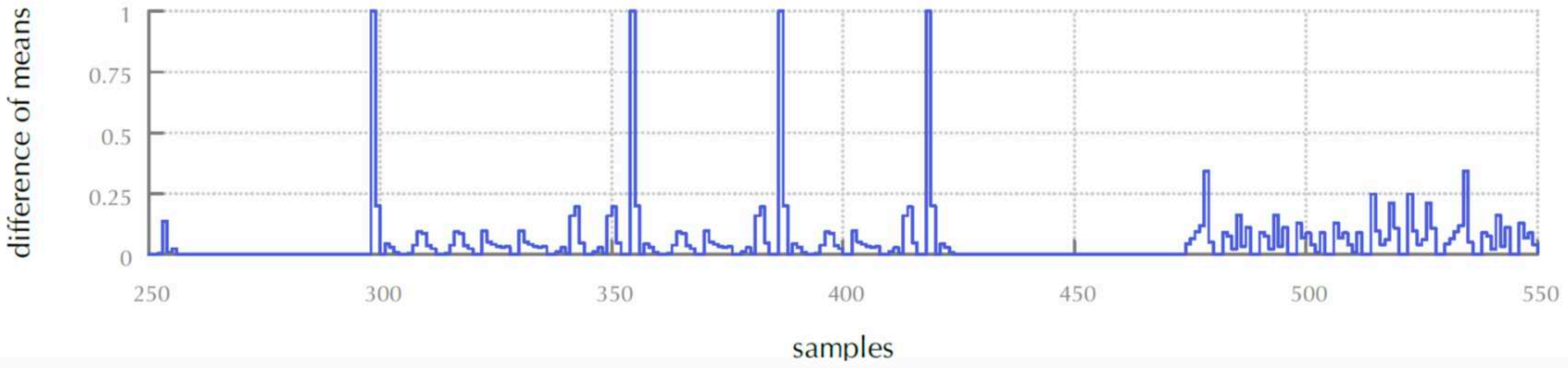
Differential Computation Analysis

4. Calculate the mean value $\bar{A}_b$ of each set.

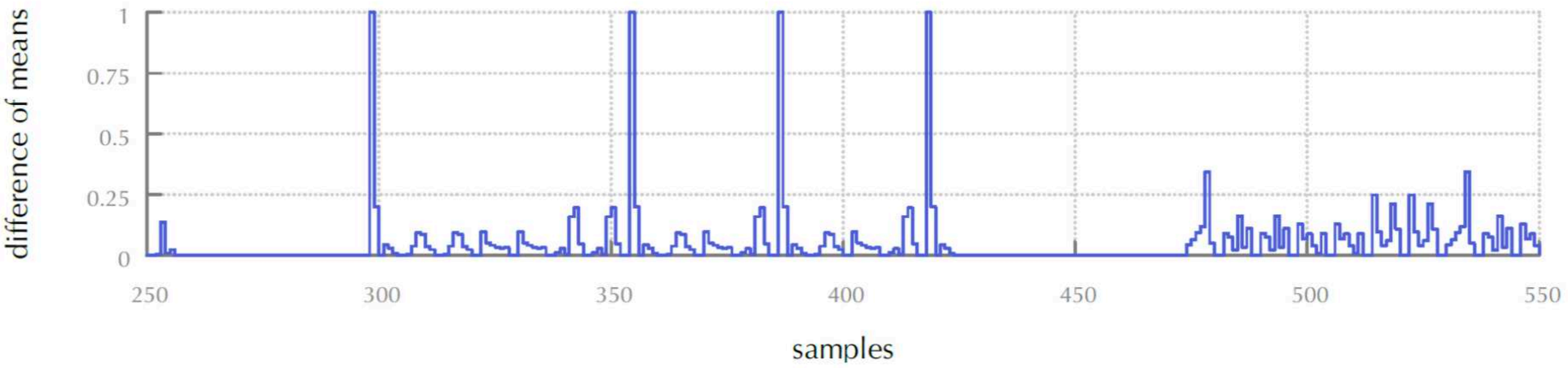




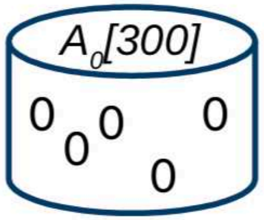
5. Calculate the difference between the average of each set $\Delta = |\bar{A}_0 - \bar{A}_1|$

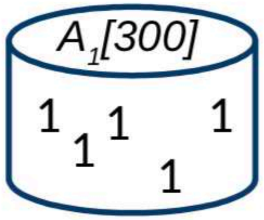


Analysing the results

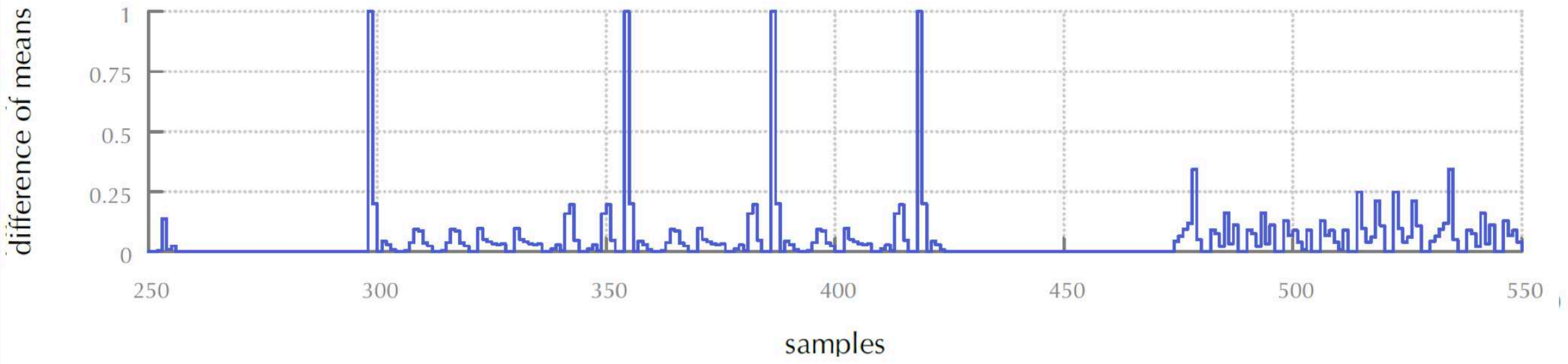


The peaks help us recognize that our key guess was correct: we calculated all values z[b] correctly and the traces have been sorted correctly in the sets.

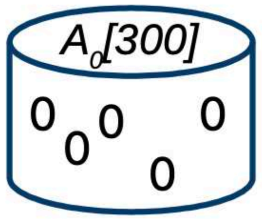


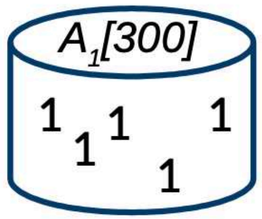


Analysing the results



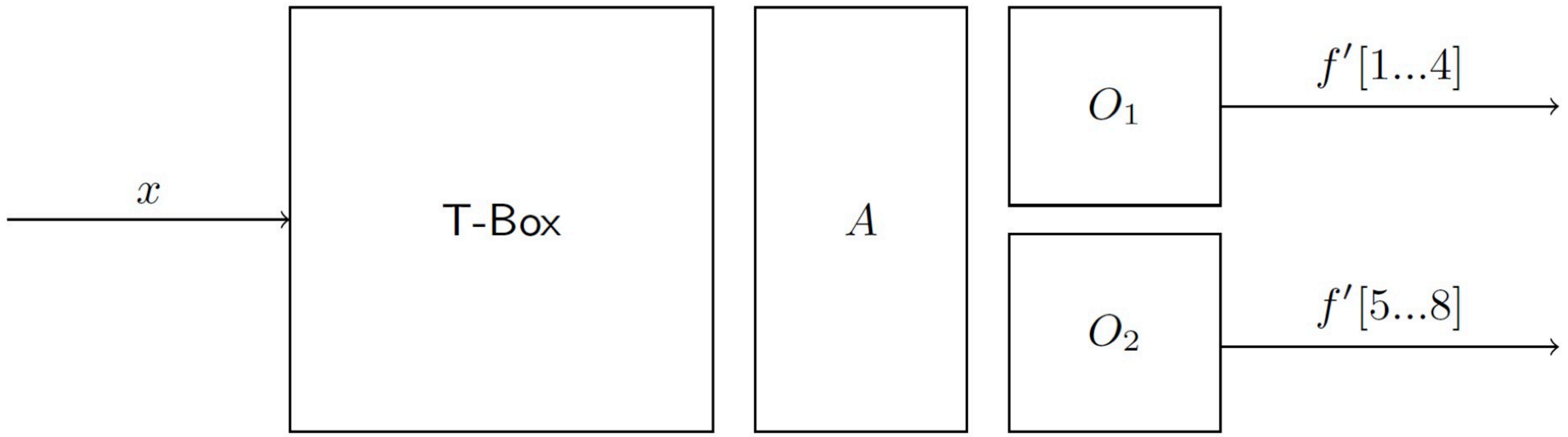
We also learn that the intermediate values z were not encoded by the white-box implementation



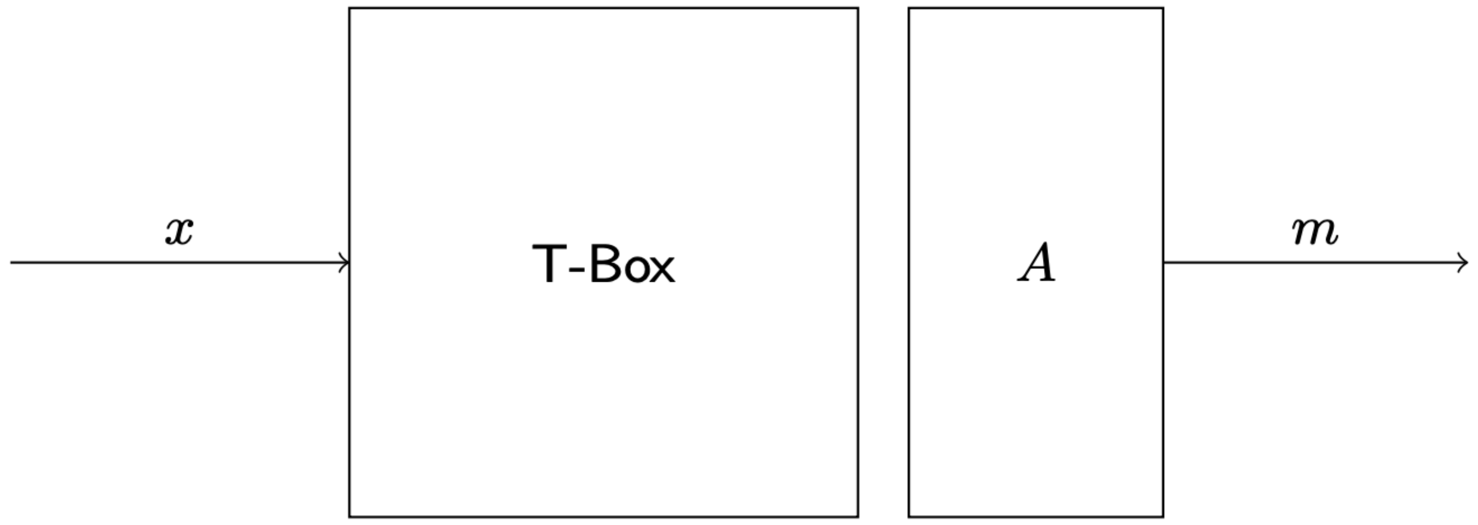


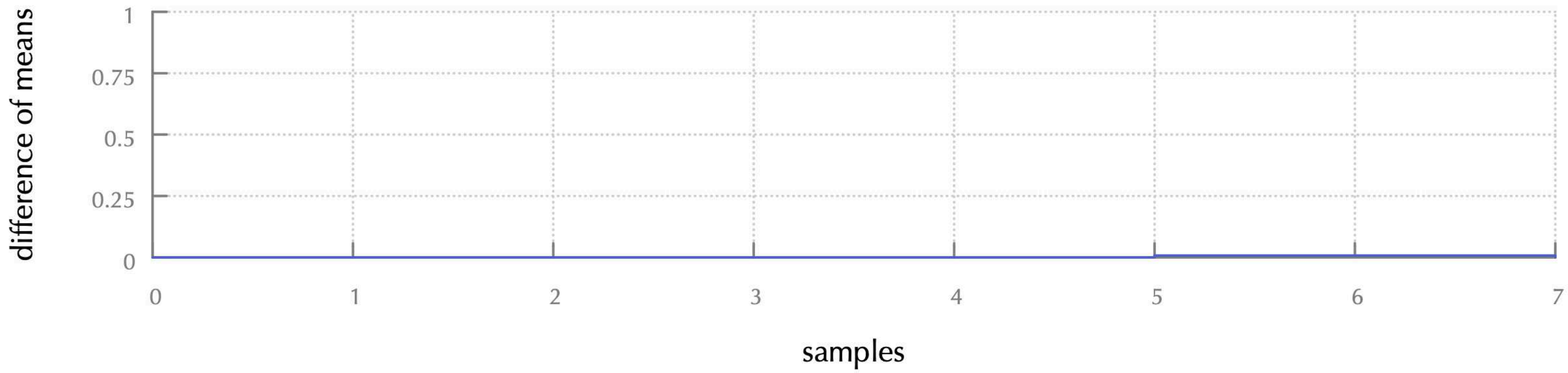
Encoded states

- Combination of linear and non-linear encodings to protect key-dependent look-up tables in a white-box design.



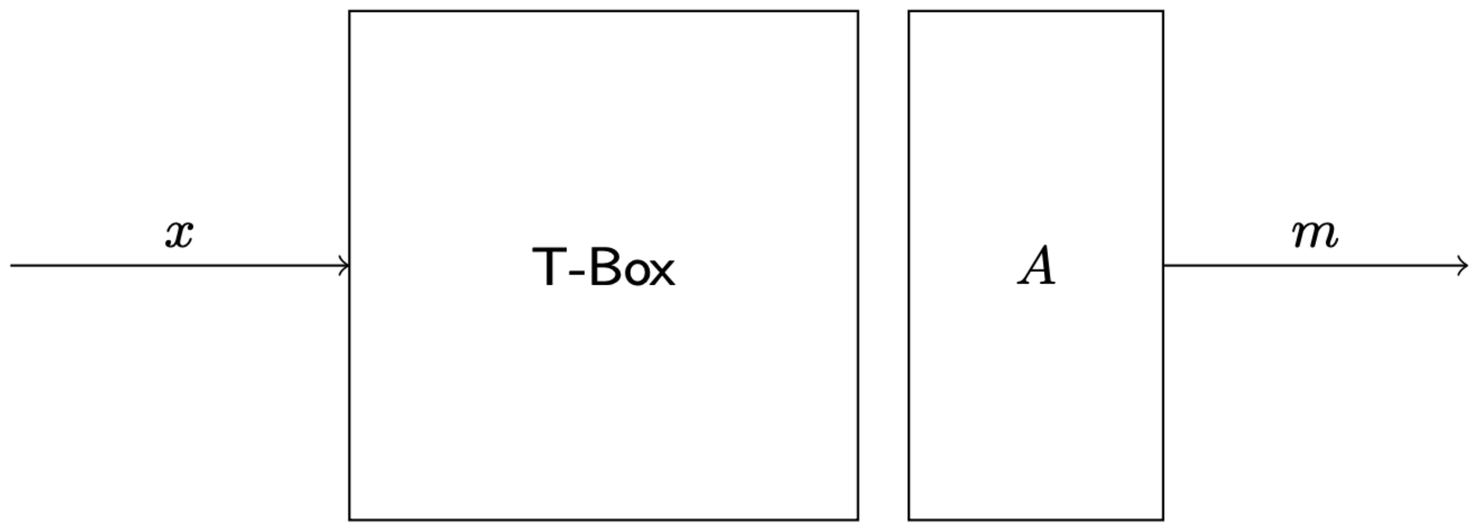
Only linearly encoded

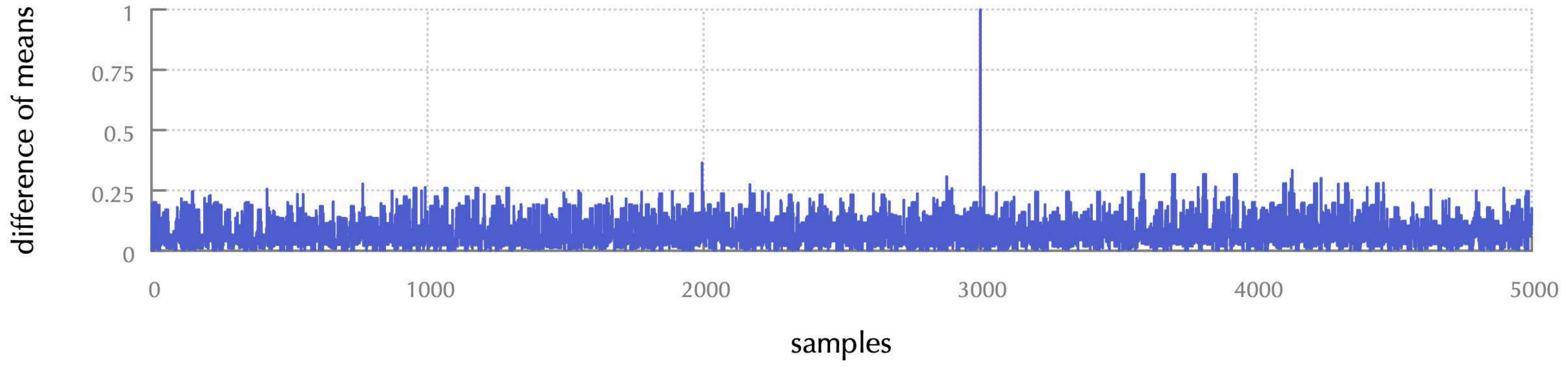




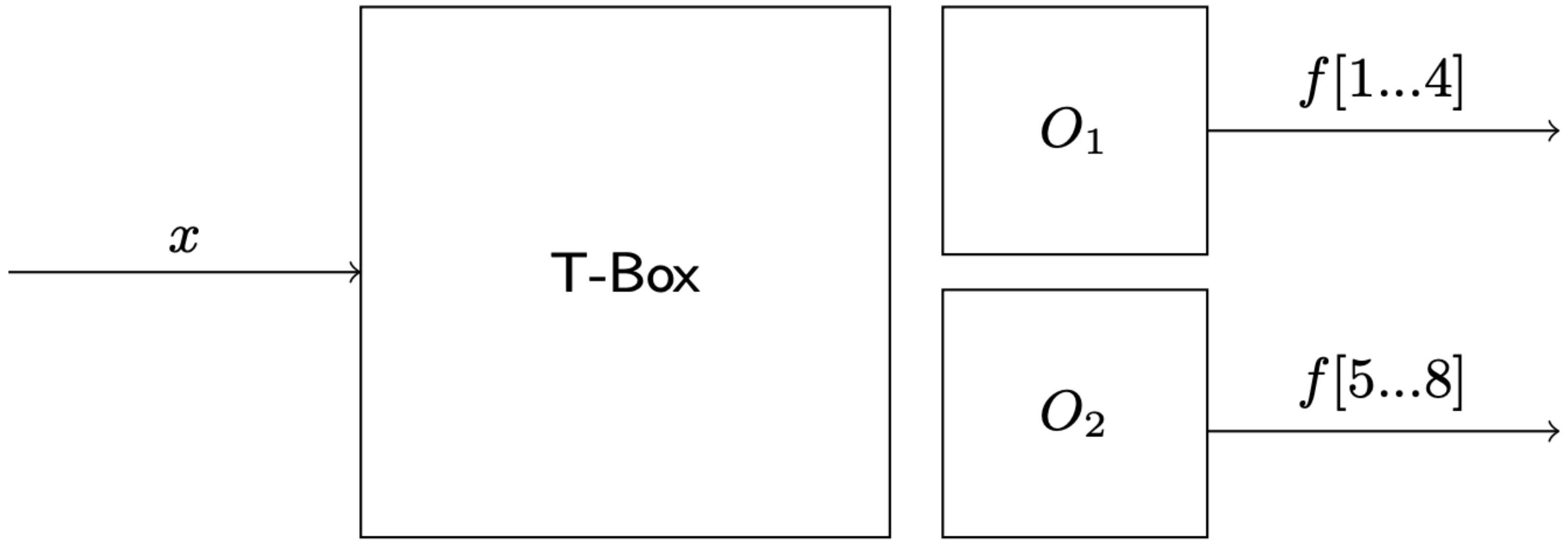
Only linearly encoded

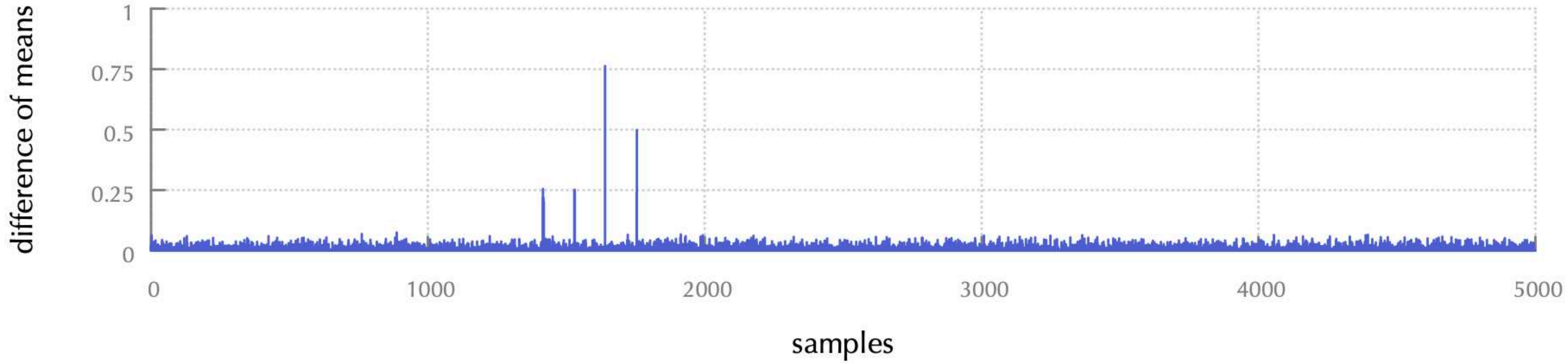
$$\begin{tikzcd} {} \arrow[r, "x"] & \boxed{\text{T-Box}}\,\boxed{A} \arrow[r, "m"] & {} \end{tikzcd}$$



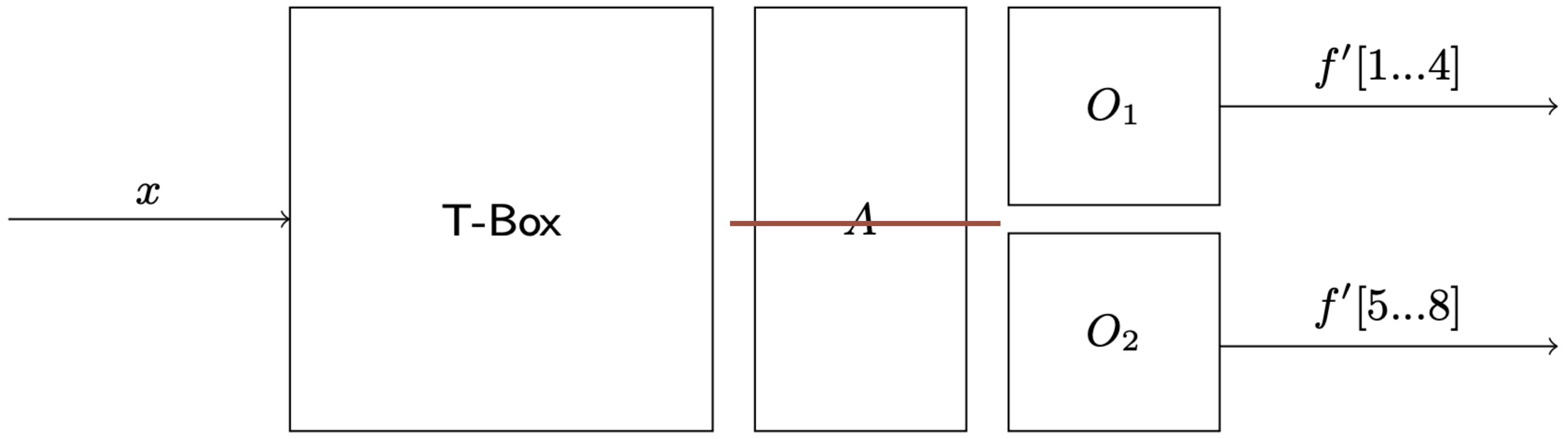


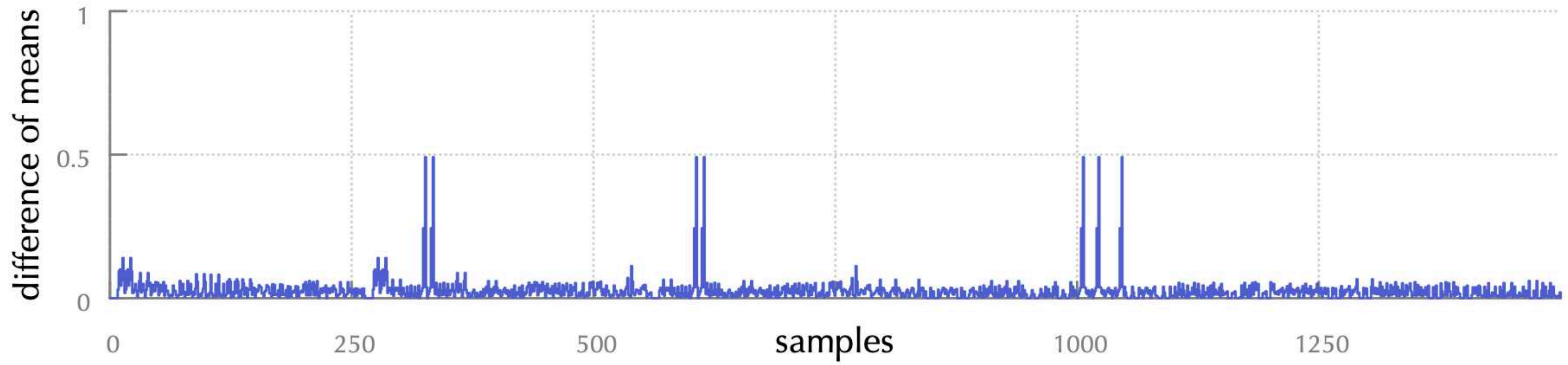
Only non-linearly encoded





Linear and non-linearly encoded





Attack extensions

Extensions of automated attacks have been presented, e.g. in

[12] Rivain and Wang: Analysis and improvement of differential computation attacks against internally -encoded white-box implementations, CHES 2019

[13] Goubin, Rivain and Wang: Defeating state-of-the-art white-box countermeasures with advanced gray-box attacks, CHES 2020

[10] Alpirez Bock, Bos, Brzuska, Hubain, Michiels, Mune, Sanfelix Gonzalez, Teuwen and Treff: White-box cryptography: don’t forget about grey-box attacks, J. of Cryptology 2019

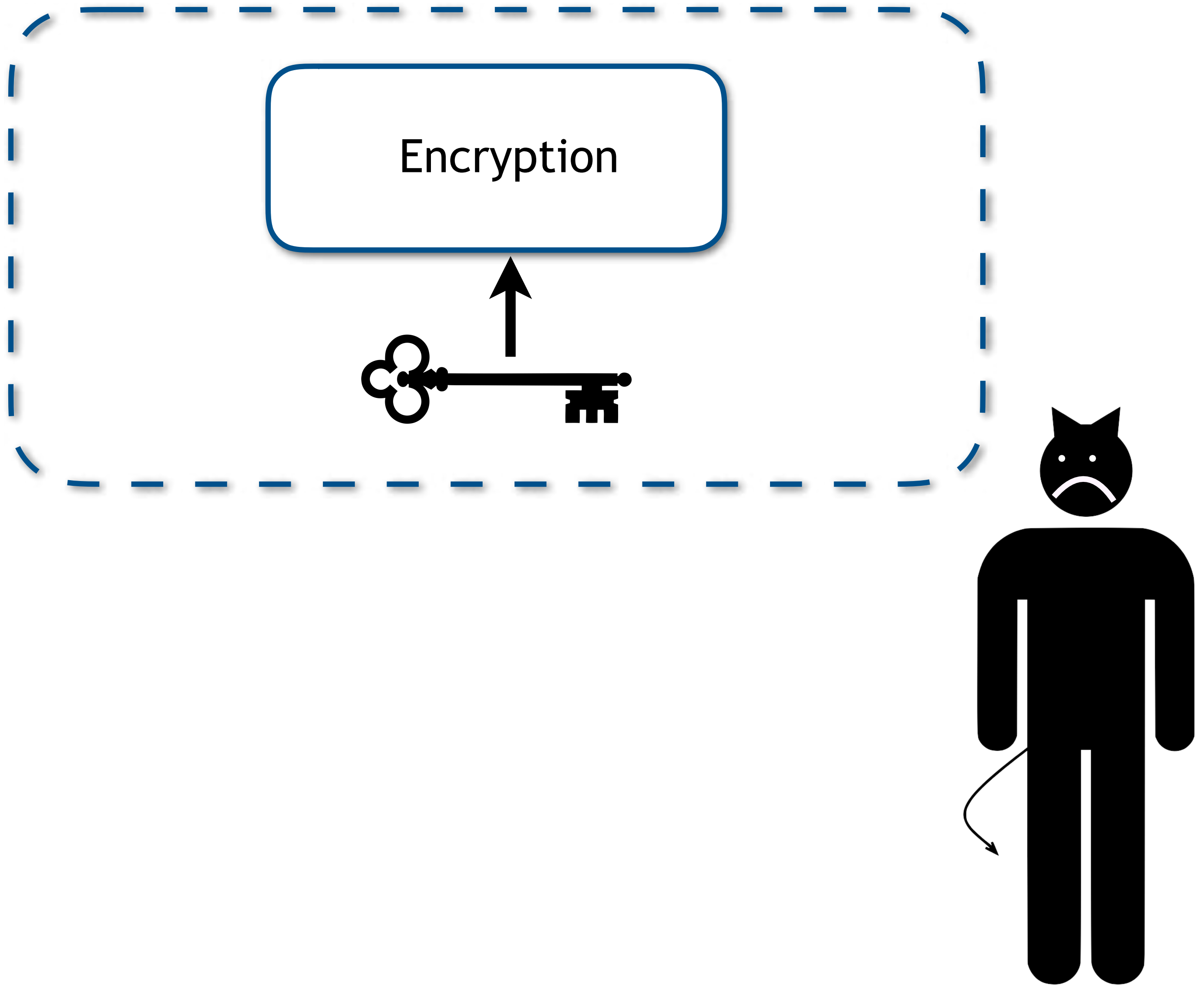
New ideas for countermeasures have also followed, e.g.

[14] Sekar, Eisenbarth,Liskiewicz: A white-box masking scheme resisting computational and algebraic attacks, CHES 2021

White-box implementations should to the very least achieve security against such attacks to provide an acceptable level of security

Going forward

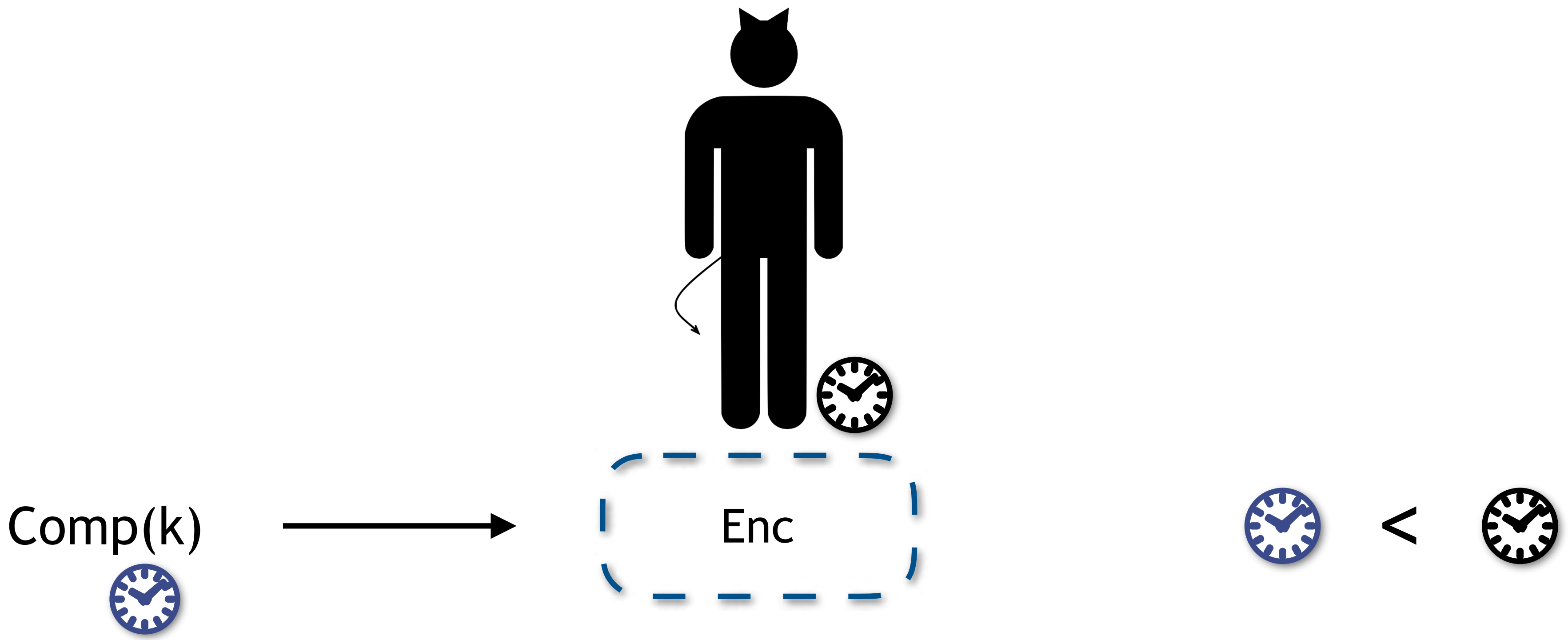
What’s left to do (besides constructing a secure WB)



What’s left to do

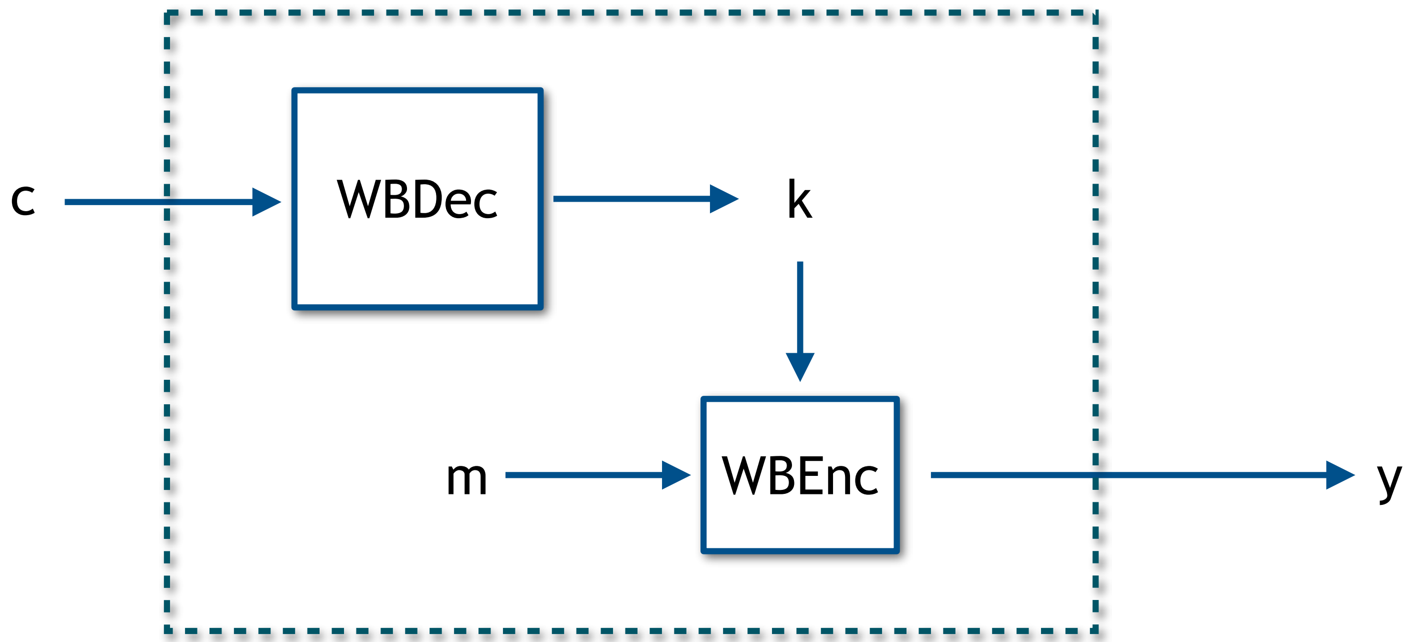
Adapt the definitional studies to suit better the real world

Current definitions consider adversaries with polynomial time, but in reality, a white-box is not used for that long



What’s left to do

Study methods of implementing dynamic white-boxes with key rotations



What’s left to do

Standardise security assessment models for white-box implementations

Propose security assessment methodologies, such as those conducted in [15] to qualify the security of a white-box design

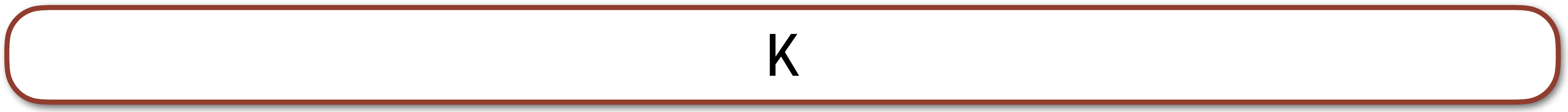
Design	Size	Speed	DCA resistant	DFA resistant	Higher-order DCA resistant	...
WB1	17MB	0.08s	Y	Y	N	
WB2	10MB	0.5s	Y	Y	Up to 1st order	
WB3	8MB	0.01s	N	Y	N	

[15] Alpirez Bock, Treff: Security assessment of white-box design submissions of the CHES 2017 CTF challenge, COSADE 2020

What’s left to do

Study links between incompressible white-box ciphers and the bounded retrieval model

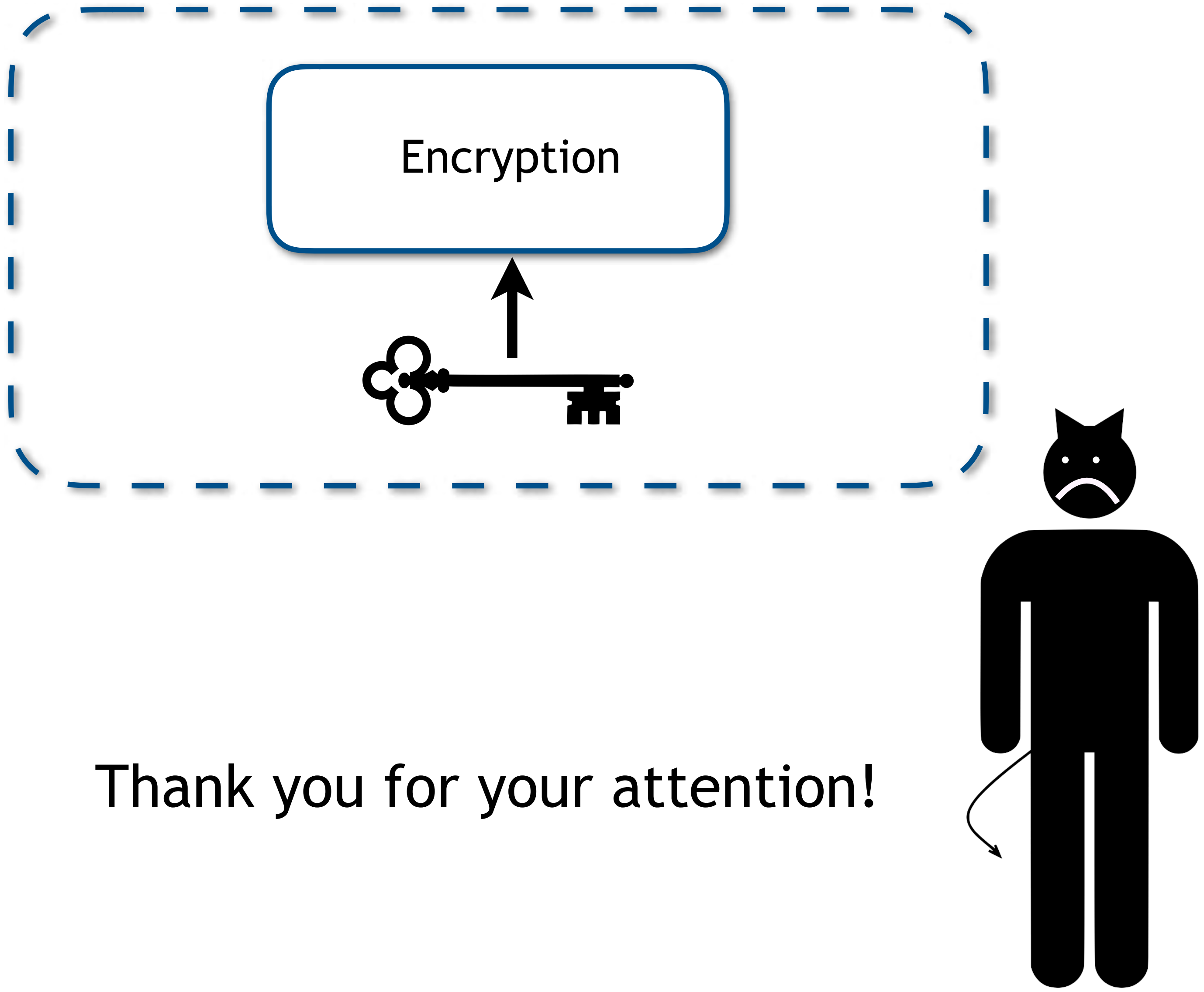
- Incompressible constructions might be useful for constructing Big Key Symmetric Encryption [16]



Enc(K,m) = c Dec(k,c) = m

$$\text{Comp}(k) \longrightarrow K$$

[16] Bellare, Kane, Rogaway: Big key symmetric encryption, CRYPTO 2016



Thank you for your attention!