Security Reductions for White-Box Key-Storage in Mobile Payments

Estuardo Alpirez Bock, Chris Brzuska, Marc Fischlin, Christian Janson, Wil Michiels

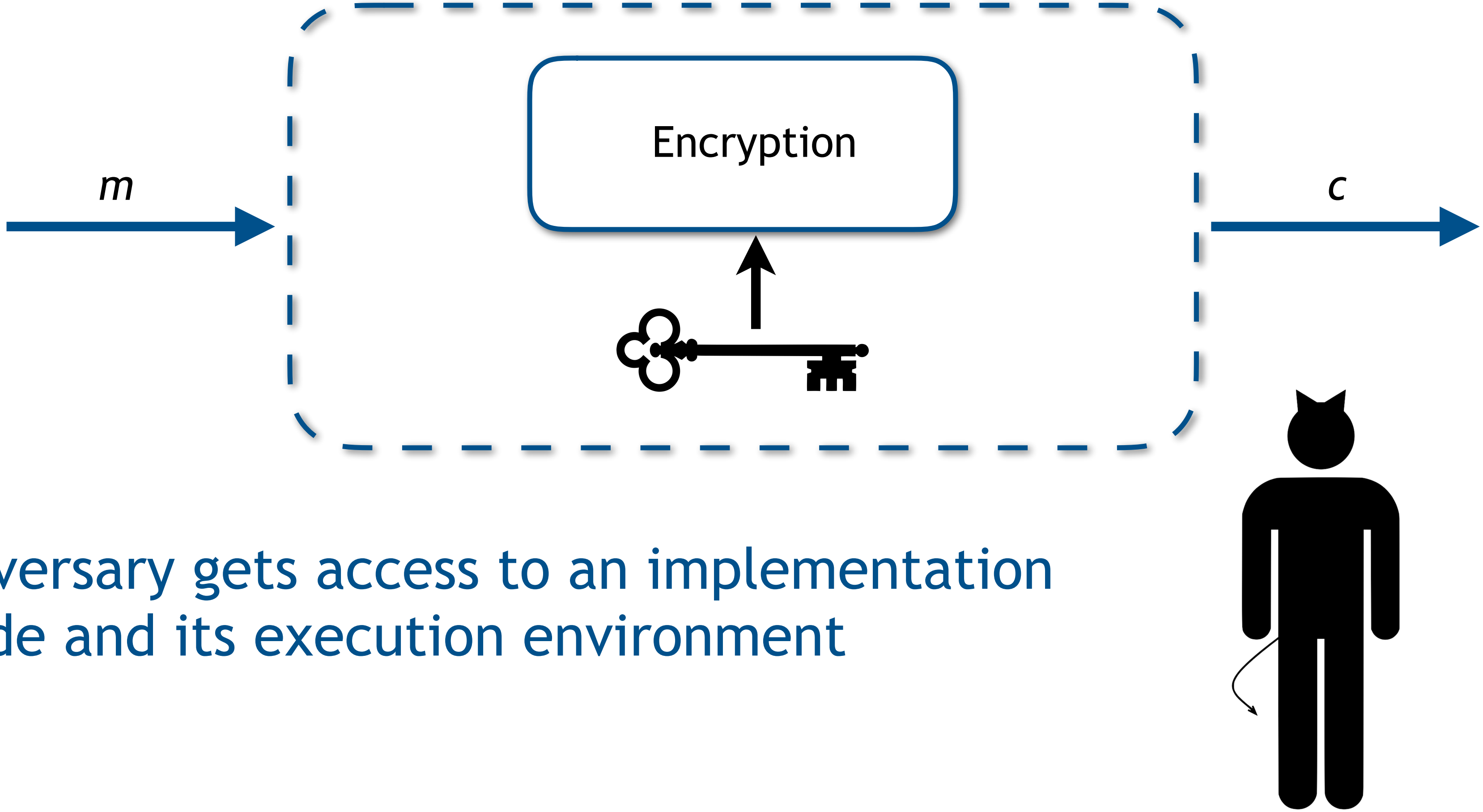
TECHNISCHE
UNIVERSITÄT
DARMSTADT

TU/e

NXP

Aalto University

White-box attack scenario



Adversary gets access to an implementation code and its execution environment

→ WB Cryptography aims to provide security even under such attack threats

Outline

- White-box crypto for mobile payments
- Device-binding
- White-box key derivation function with device-binding
- White-box mobile payment application

White-box crypto for mobile payment applications

White-box crypto for payment applications

Traditionally white-box crypto was mainly used in the context of DRM applications

In 2015, Android introduced Host Card Emulation (HCE), allowing the application processor of mobile phones to use Near Field Communication (NFC)

Enable vendors to distribute payment applications implemented in software only

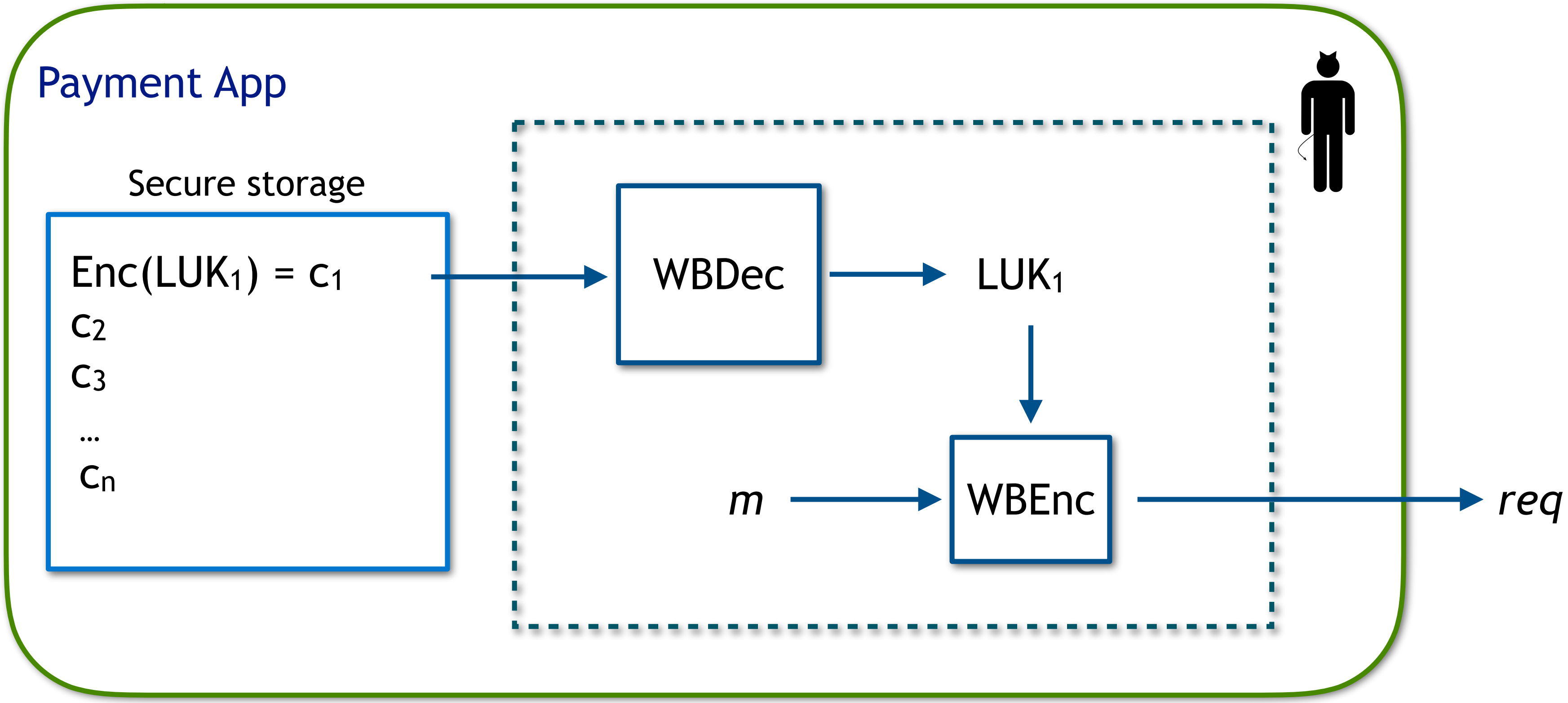
- increasing their deployability and gaining independence from the phone manufacturers

White-box crypto was proposed as a software countermeasure technique to protect mobile payment applications [1]

[1] EMVCo: *EMV Mobile payment: Software-based mobile payment security requirements (2019)*

White-box crypto for payment applications

- The application stores user specific keys in encrypted form
- The keys are decrypted and used for generating a transaction request message



Payment App

Secure storage

$\mathrm{Enc}(LUK_1) = c_1$

c_2

c_3

…

c_n

WBDec

LUK_1

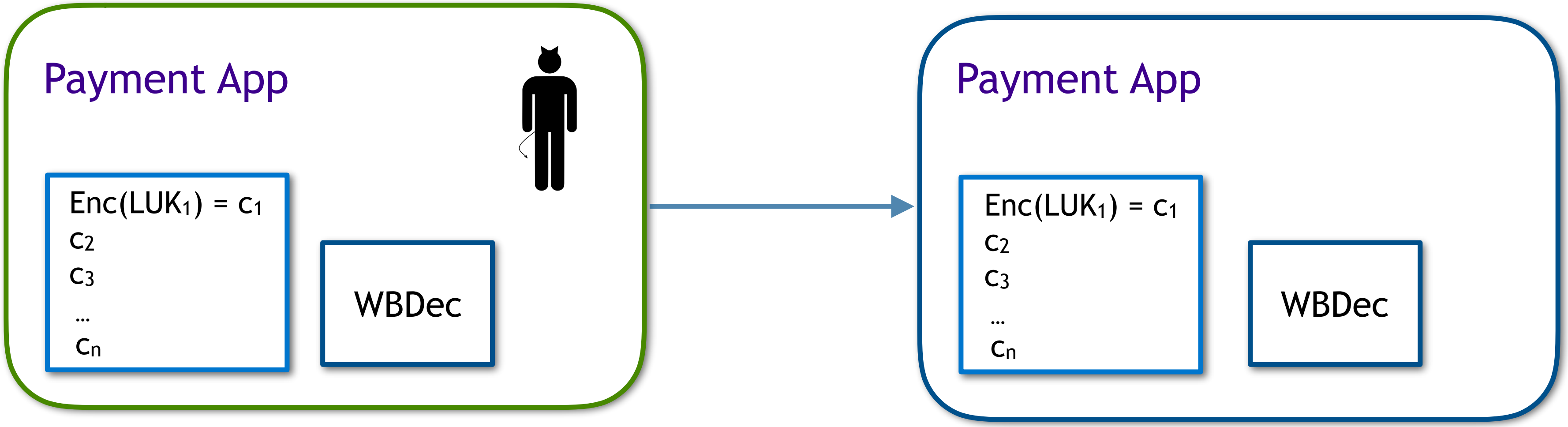
m

WBEnc

req

White-box crypto for payment applications

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



Payment App

$Enc(LUK_1) = c_1$
c_2
c_3
…
c_n

WBDec

Payment App

$Enc(LUK_1) = c_1$
c_2
c_3
…
c_n

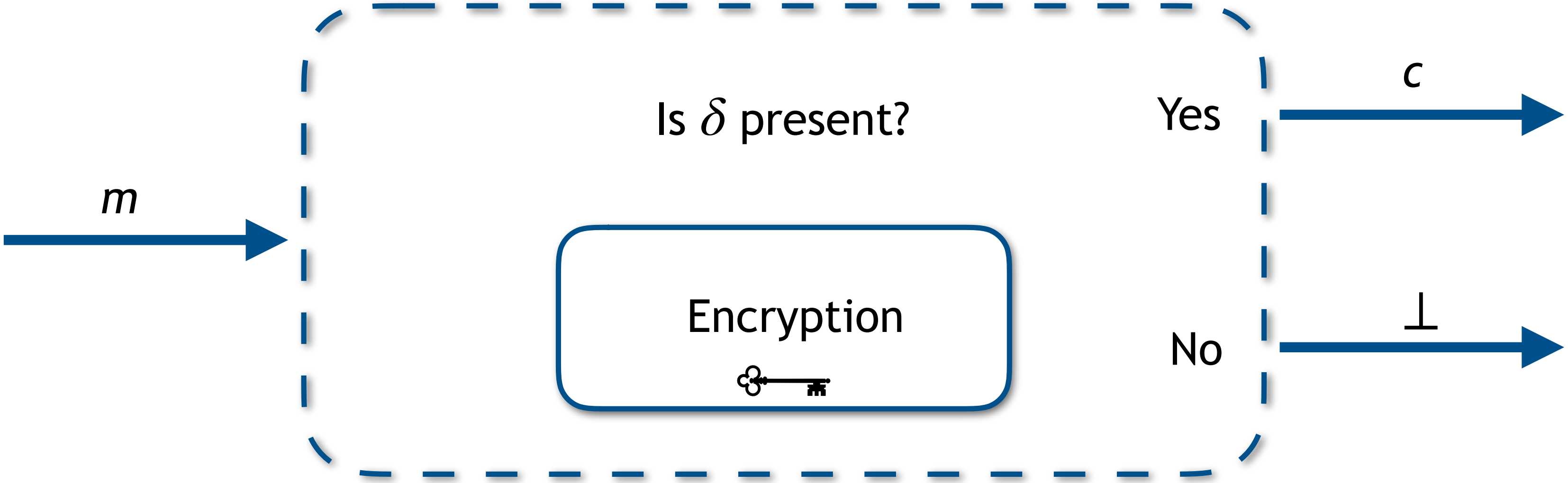
WBDec

Our white-box program should provide protection against such *code-lifting* attacks

Device binding for mitigating code-lifting attacks

Device-binding

- **Generate a white-box program such that it can only be executed on one specific device. The execution is dependable on a unique hardware identifier δ.**



Device-binding for white-box programs

For protecting white-box programs from code-lifting attacks, we propose to focus on the property of device (or hardware) binding

See [2] for more motivation on the use of hardware-binding

We introduce security notions for a white-box KDF with hardware-binding and for a white-box mobile payment application

Present corresponding constructions based on puncturable PRFs and indistinguishability obfuscation

Our constructions help understand how such white-box programs can be implemented in practice (substituting the PPRFs and iO by more efficient primitives)

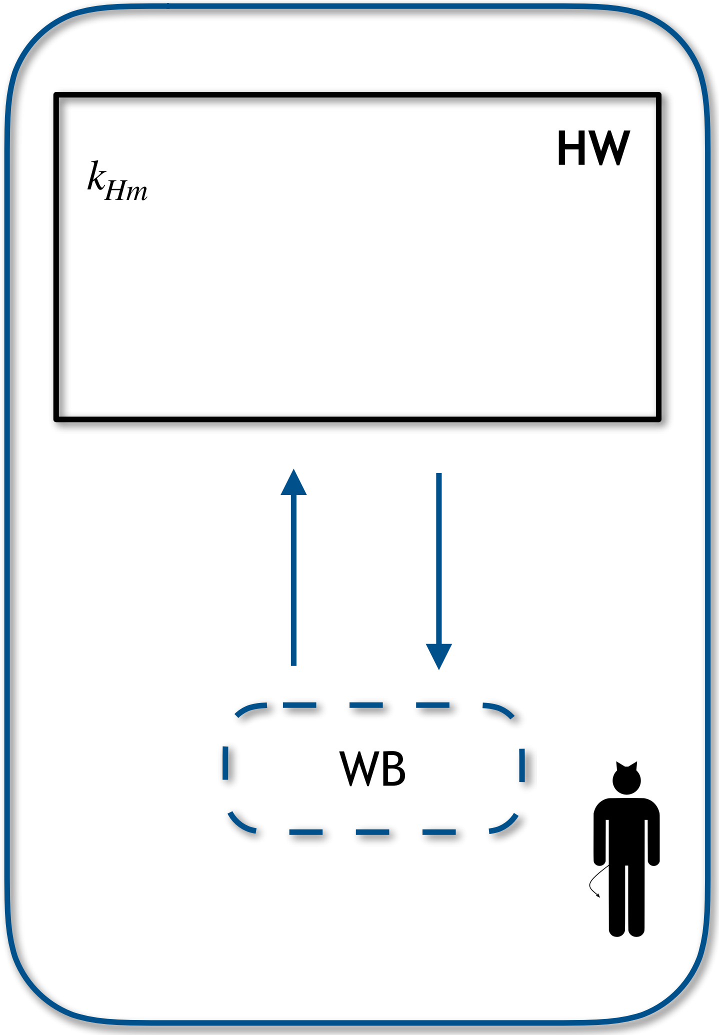
[2] E. Alpirez Bock, A. Amadori, C. Brzuska and W. Michiels: *On the security goals of white-box cryptography*, CHES 2020

Secure hardware in the device

Assume our device has some hardware component, which is not accessible to the white-box adversary

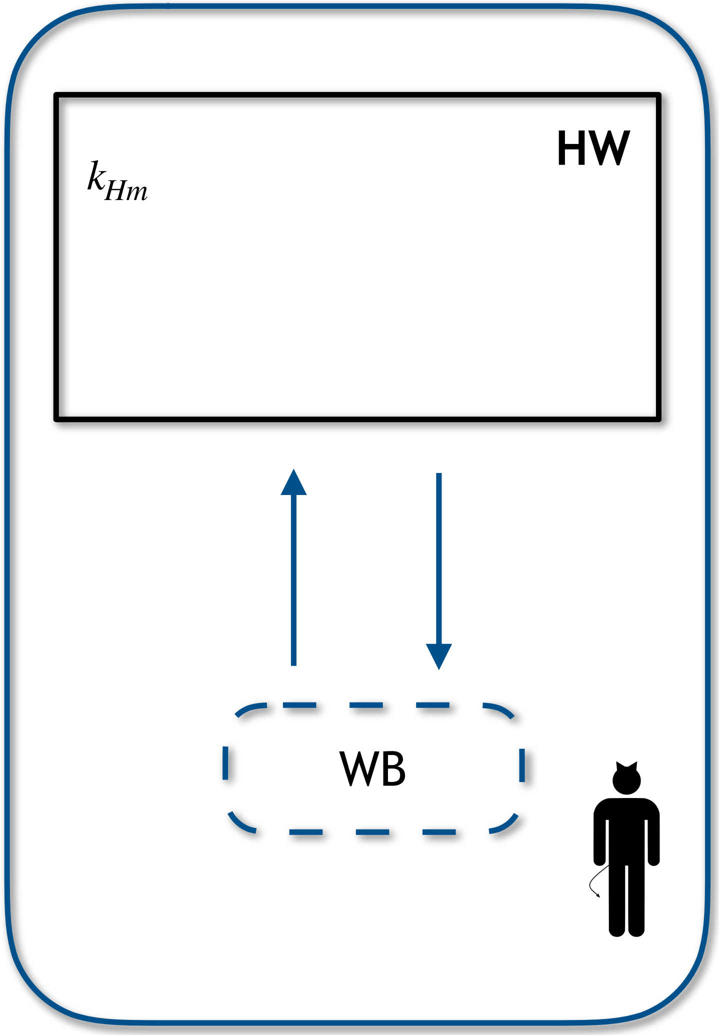
The secure hardware stores some secret, main key k_{Hm}

Based on this key, the hardware generates responses to the white-box program



Why white-box with hw-binding?

Question: why use white-box crypto, if my device has a secure hardware with some key material anyway?



1) independence from the phone manufacturer

2) not all HWs are implemented equally

3) avoid context switches to improve performance

4) avoid exposure of intermediate values during the calculations

Idea: make our white-box program dependable on a simple operation performed by the hardware

Defining Hardware-binding

Defining a white-box primitive in combination of a *hardware module*

White-box key derivation function (WKDF)

- Consider a key derivation function (KDF)

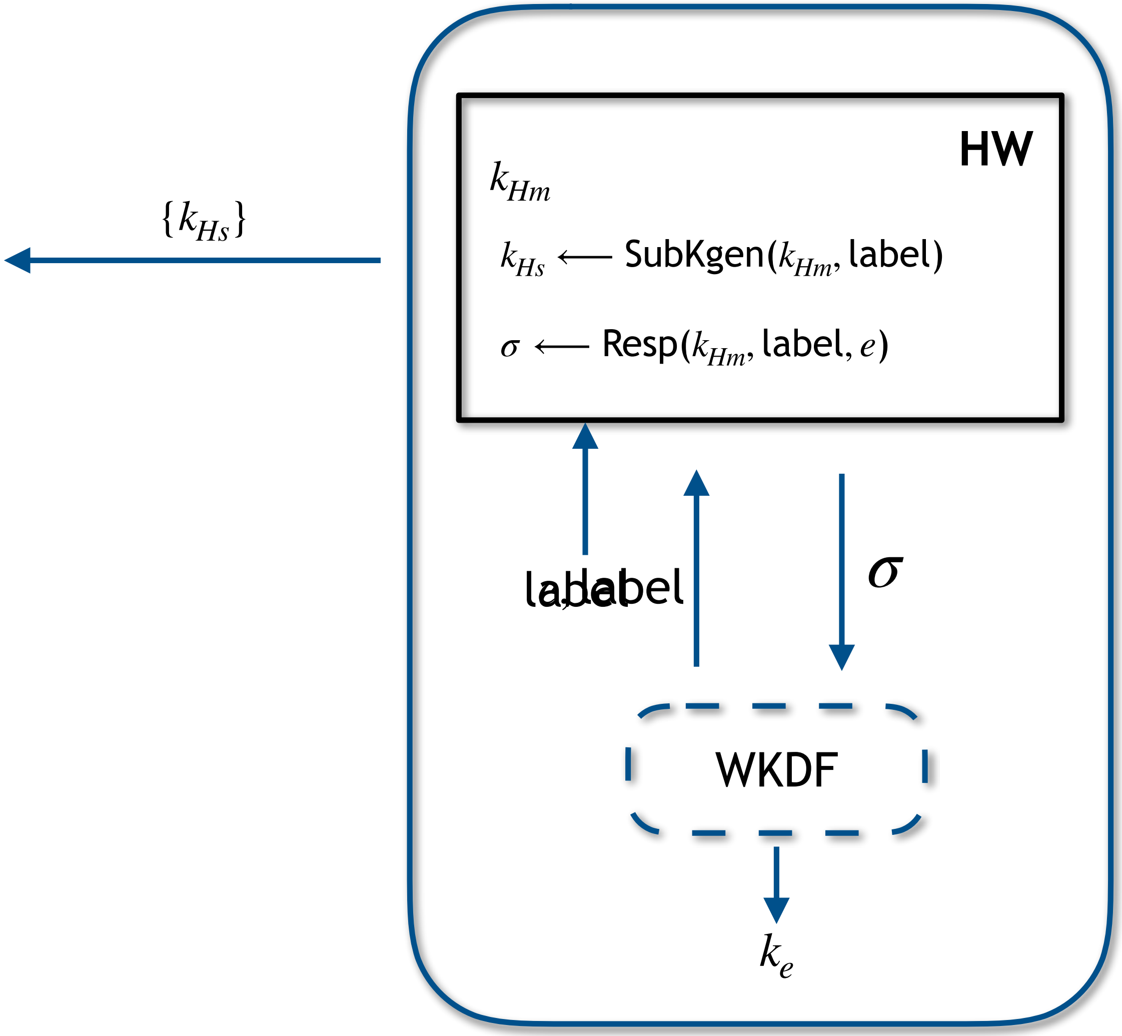
$$k_e \longleftarrow \mathsf{KDF}(k, e)$$

- Build a functional equivalent (hardware-bound) WKDF

$$\mathsf{KDF}(k, e) = \mathsf{WKDF}(e, \,.\,)$$

- We define the syntax of the hardware module and the WKDF

White-box KDF

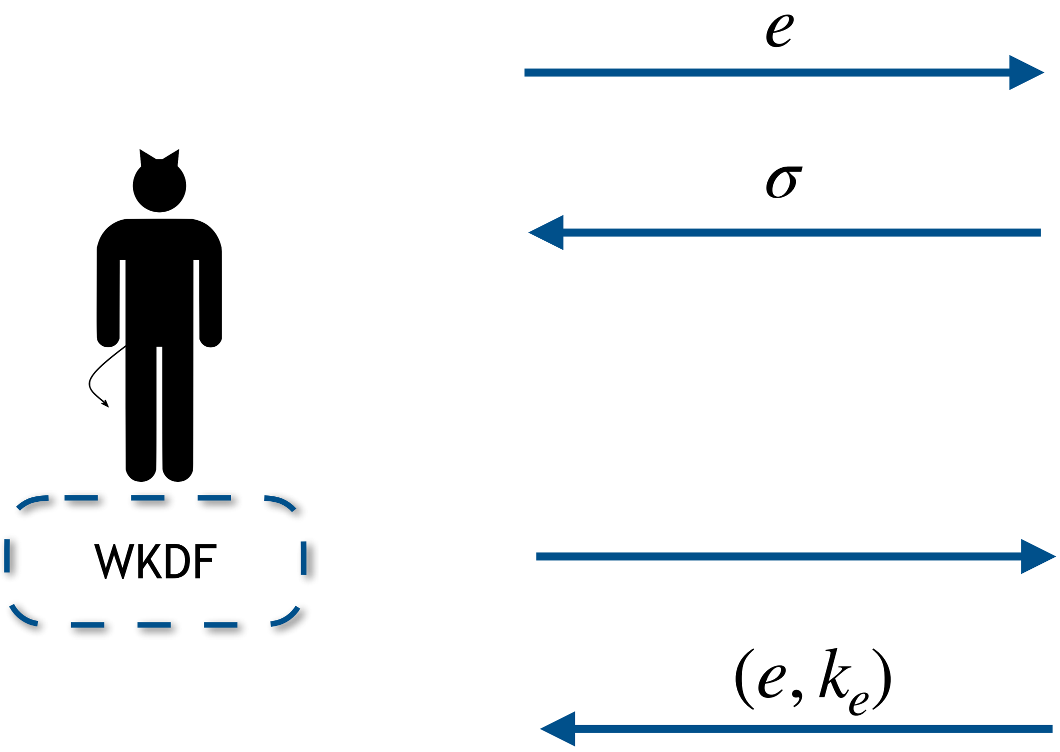


$$\text{WKDF} \longleftarrow \$\text{Comp}(k, k_{Hs})$$

$$\text{WKDF}(e, \sigma) = \text{KDF}(k, e)$$

```
$\text{WKDF}(e, \sigma)$
-----------------------------
$b \longleftarrow \text{Check}(k_{Hs}, e, \sigma)$
if $b = 0$
    return $\perp$
else $k_e \longleftarrow \text{KDF}(k, e)$
return $k_e$
```

Security of WKDF



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

```
KDF()
$e \longleftarrow \$\{0,1\}^n$
$Q := Q \cup \{e\}$
if $b = 1$
    $k_e \longleftarrow \text{KDF}(k, e)$
else $k_e \longleftarrow \$\{0,1\}^n$
```

Construction

```
$\mathsf{SubKgen}(k_{Hm}, \mathsf{label})$
$k_{Hs} \longleftarrow \mathsf{PRF}(k_{Hm}, \mathsf{label})$
return $k_{Hs}$
```

```
$\mathsf{Resp}(k_{Hm}, \mathsf{label}, e)$
$\sigma \longleftarrow \mathsf{PPRF}(\mathsf{PRF}(k_{Hm}, \mathsf{label}), e)$
return $\sigma$
```

```
$\mathsf{Comp}(k, k_{Hs})$
$\mathsf{WKDF} \longleftarrow \$ iO(C[k, k_{Hs}])$
return WKDF
```

```
$C[k_{Hs}, k](e, \sigma)$
if $\mathsf{PRG}(\sigma) = \mathsf{PRG}(\mathsf{PPRF}(k_{Hs}, e))$        // $\mathsf{Check}(k_{Hs}, e, \sigma)$
    $k_e \longleftarrow \mathsf{PPRF}(k, e)$        // $k_e \longleftarrow \mathsf{KDF}(k, e)$
    return $k_e$
else return $0^n$
```

Security

We prove security via the punctured programs approach from Sahai and Waters [3]

```
$C[k_{Hs}, k](e, \sigma)$
if PRG($\sigma$) = PRG(PPRF($k_{Hs}, e$))
    $k_e \longleftarrow$ PPRF($k, e$)
    return $k_e$
else return $0^n$
```

$$\equiv$$

```
$C_2[k^z_{Hs}, k, z, \tau](e, \sigma)$
if $e = z$ and PRG($\sigma$) = PRG($\tau$)
or if PRG($\sigma$) = PRG(PPRF($k^z_{Hs}, e$))
    $k_e \longleftarrow$ PPRF($k, e$)
    return $k_e$
else return $0^n$
```

with $\tau = \text{PPRF}(k_{Hs}, z)$

[3] A. Sahai and B. Waters: *How to use indistinguishability obfuscation: deniable encryption and more*, STOC 2014

Security

```
$C_2[k^z_{Hs}, k, z, \tau](e, \sigma)$
if $e = z$ and PRG($\sigma$) = PRG($\tau$)
or if PRG($\sigma$) = PRG(PPRF($k^z_{Hs}, e$))
    $k_e \longleftarrow$ PPRF($k, e$)
    return $k_e$
else return $0^n$
```

with $\tau = \text{PPRF}(k_{Hs}, z)$

$$\equiv$$

```
$C_3[k^z_{Hs}, k, z, y](e, \sigma)$
if $e = z$ and PRG($\sigma$) = $y$
or if PRG($\sigma$) = PRG(PPRF($k^z_{Hs}, e$))
    $k_e \longleftarrow$ PPRF($k, e$)
    return $k_e$
else return $0^n$
```

with $\tau \longleftarrow \$\{0,1\}^n$ and $y = \text{PRG}(\tau)$

Security

```
$C_3[k^z_{Hs}, k, z, y](e, \sigma)$
if $e = z$ and $\mathrm{PRG}(\sigma) = y$
or if $\mathrm{PRG}(\sigma) = \mathrm{PRG}(\mathrm{PPRF}(k^z_{Hs}, e))$
    $k_e \longleftarrow \mathrm{PPRF}(k, e)$
    return $k_e$
else return $0^n$
```

with $\tau \longleftarrow \$\{0,1\}^n$ and $y = \mathrm{PRG}(\tau)$

$$\equiv$$

```
$C_4[k^z_{Hs}, k^z, z, y, k^*_e](e, \sigma)$
if $e = z$ and $\mathrm{PRG}(\sigma) = y$
    return $k^*_e$
if $\mathrm{PRG}(\sigma) = \mathrm{PRG}(\mathrm{PPRF}(k^z_{Hs}, e))$
    $k_e \longleftarrow \mathrm{PPRF}(k^z, e)$
    return $k_e$
else return $0^n$
```

with $y \longleftarrow \$\{0,1\}^{2n}$ and $k^*_e \leftarrow$ PPRF(k,z)

Security

```
$C_4[k^z_{Hs}, k^z, z, y, k^*_e](e, \sigma)$
if $e = z$ and PRG($\sigma$) = $y$
    return $k^*_e$
if PRG($\sigma$) = PRG(PPRF($k^z_{Hs}, e$))
    $k_e \longleftarrow$ PPRF($k^z, e$)
    return $k_e$
else return $0^n$
```

with $y \longleftarrow \$\{0,1\}^{2n}$

$$\equiv$$

```
$C_5[k^z_{Hs}, k^z, z, y](e, \sigma)$
if $e = z$ and PRG($\sigma$) = $y$
    return $0^n$
if PRG($\sigma$) = PRG(PPRF($k^z_{Hs}, e$))
    $k_e \longleftarrow$ PPRF($k^z, e$)
    return $k_e$
else return $0^n$
```

with $y \longleftarrow \$\{0,1\}^{2n}$

White-box mobile payment application

Using our WKDF

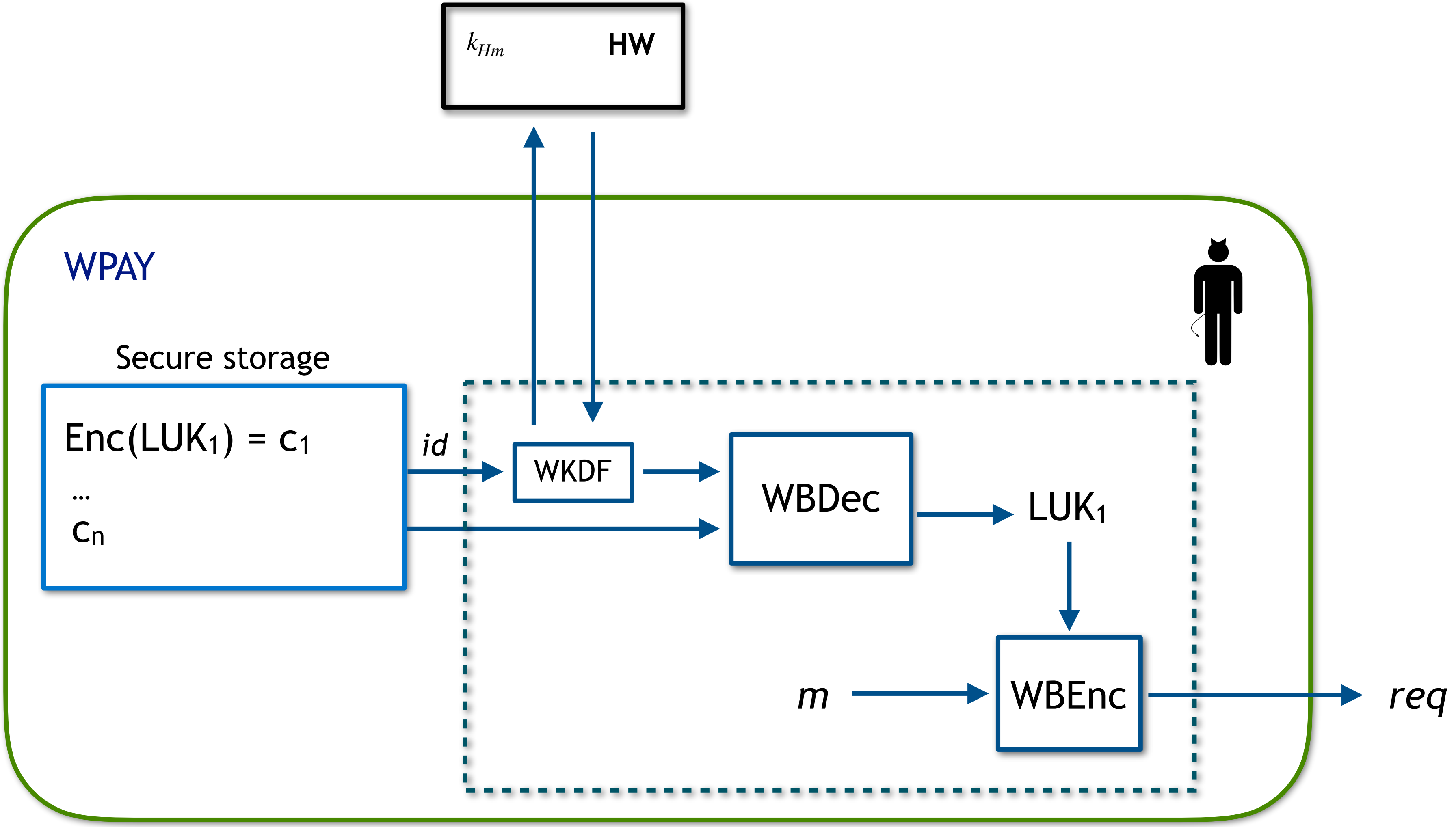
Now we can use our WKDF as a building block for further constructions in the white-box attack scenario

Idea: derive keys from the WKDF for performing encryptions/decryptions

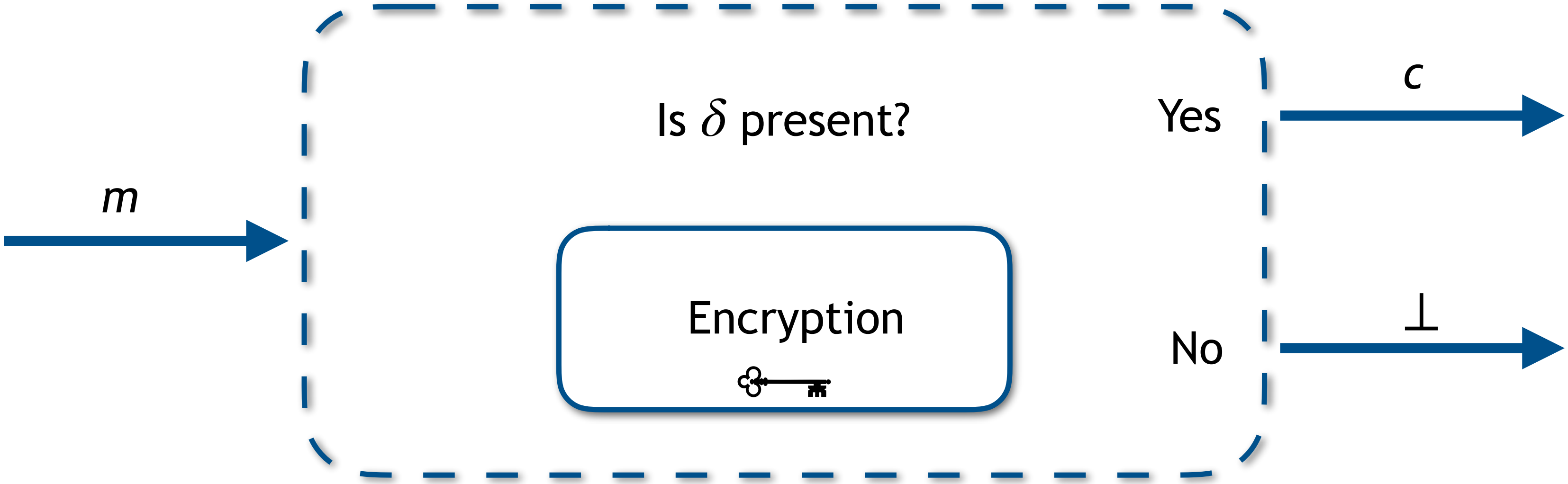
The security is derived from the WKDF, which is hardware-bound and white-box secure

We construct a mobile payment application, which is dependent on the WKDF

White-box secure Payment Applications



Thank you for your attention!



m

Is δ present?

Encryption

Yes

c

No

$\perp$