RSAConference2019

San Francisco | March 4 – 8 | Moscone Center

BETTER.

SESSION ID: CRYP-W02

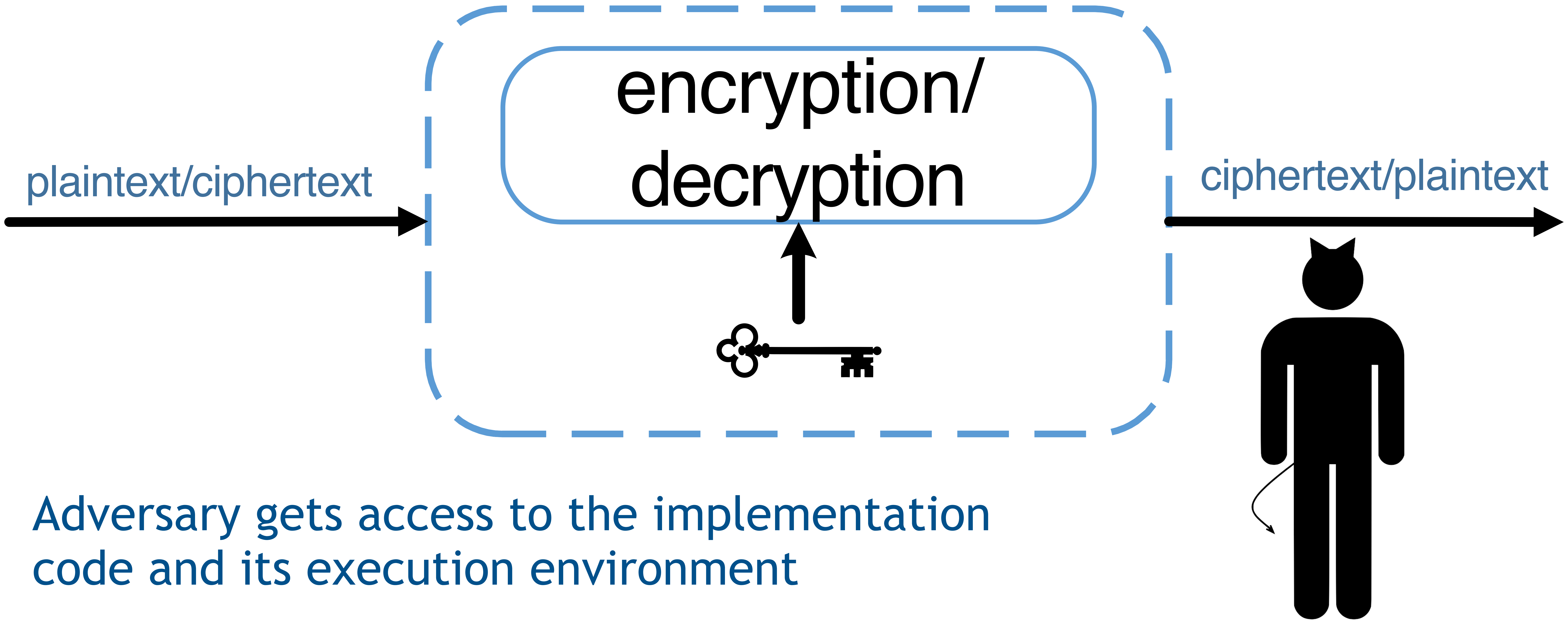
Doubly half-injective PRGs for incompressible white-box cryptography

Estuardo Alpirez Bock

Aalto University, Finland

Alessandro Amadori, Joppe W. Bos, Chris Brzuska, Wil Michiels

White-box attack scenario



Adversary gets access to the implementation code and its execution environment

→ WB Cryptography aims to maintain a program secure even when subject to this attack model

Aalto University

Outline

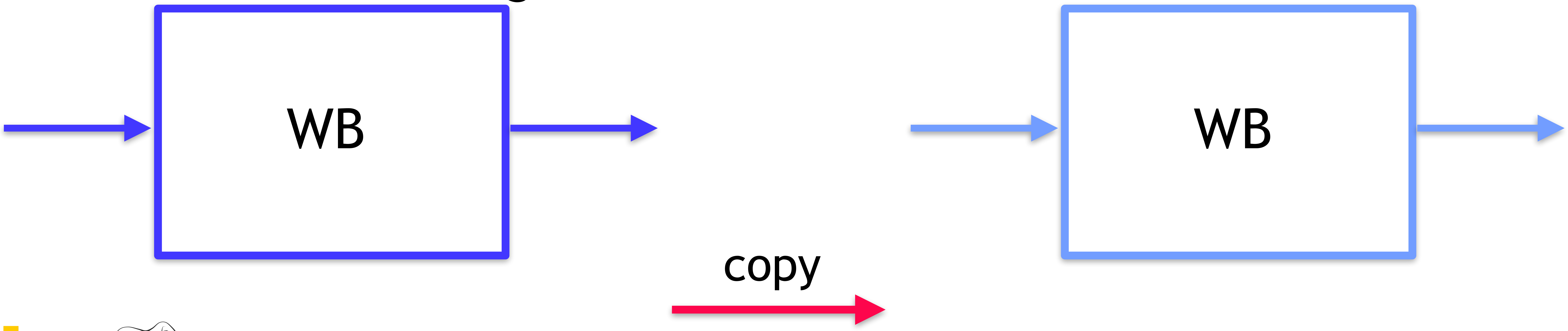
- Incompressibility for white-box cryptography
- PRGs, PRFs and the GGM tree
- An incompressible PRF
- Doubly-half injective PRGs
- Conclusions

RSAConference2019

Incompressibility for white-box cryptography

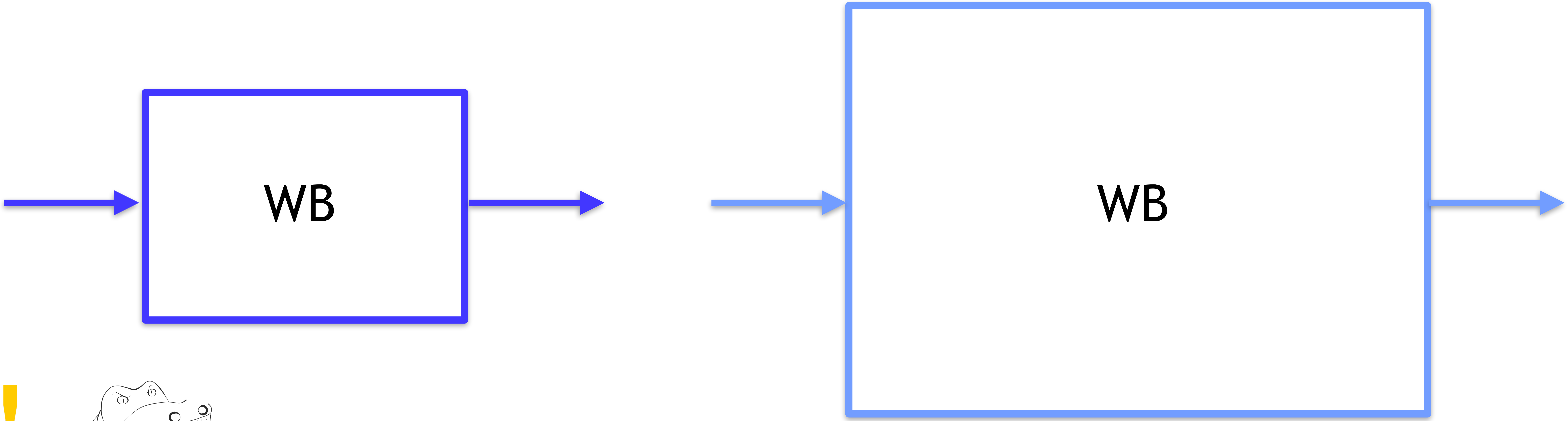
Adversarial capabilities

- The adversary gets access to the program code of an implementation
- He could extract keys, but also copy the program and its functionality
- Threat of code-lifting attacks



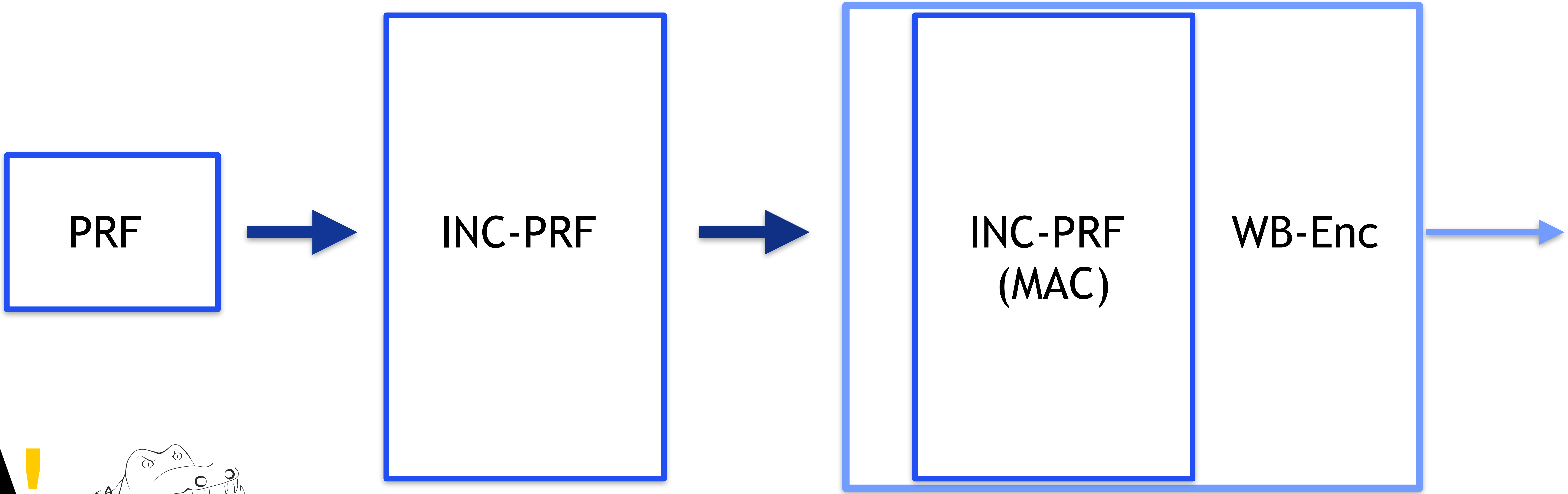
Methods for mitigating code-lifting attacks

- Incompressibility
  Delerablée, Lepoint, Paillier, Rivain: *White-box security notions for symmetric encryption schemes*
  Fouque, Karpman, Kirchner, Minaud: *Efficient and provable white-box primitives*



In this work

- We build an incompressible wb-encryption scheme
- Our construction is based on standard assumptions, such as pseudorandom generators and pseudorandom functions

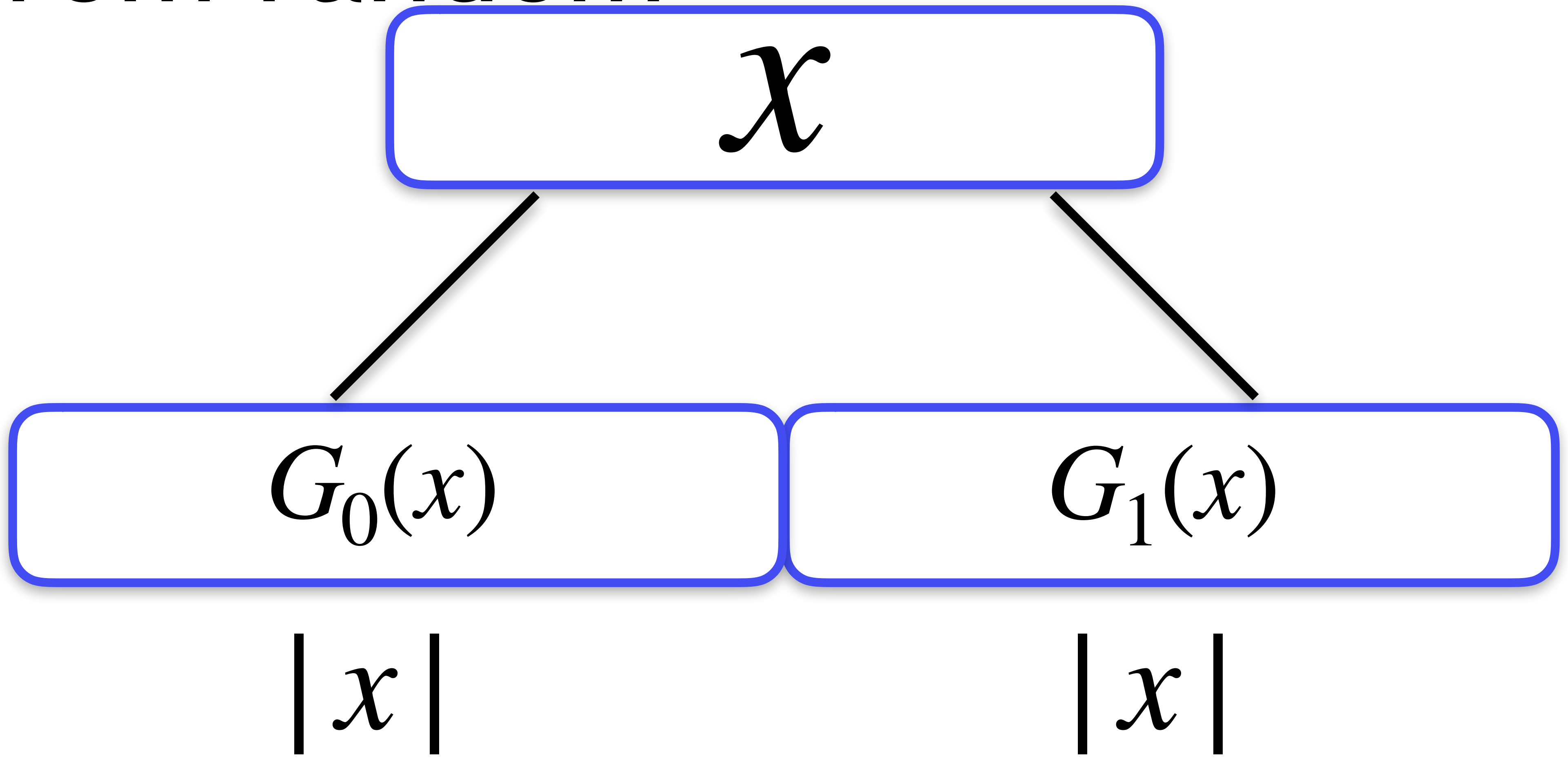


RSA Conference 2019

PRGs, PRFs and the GGM tree

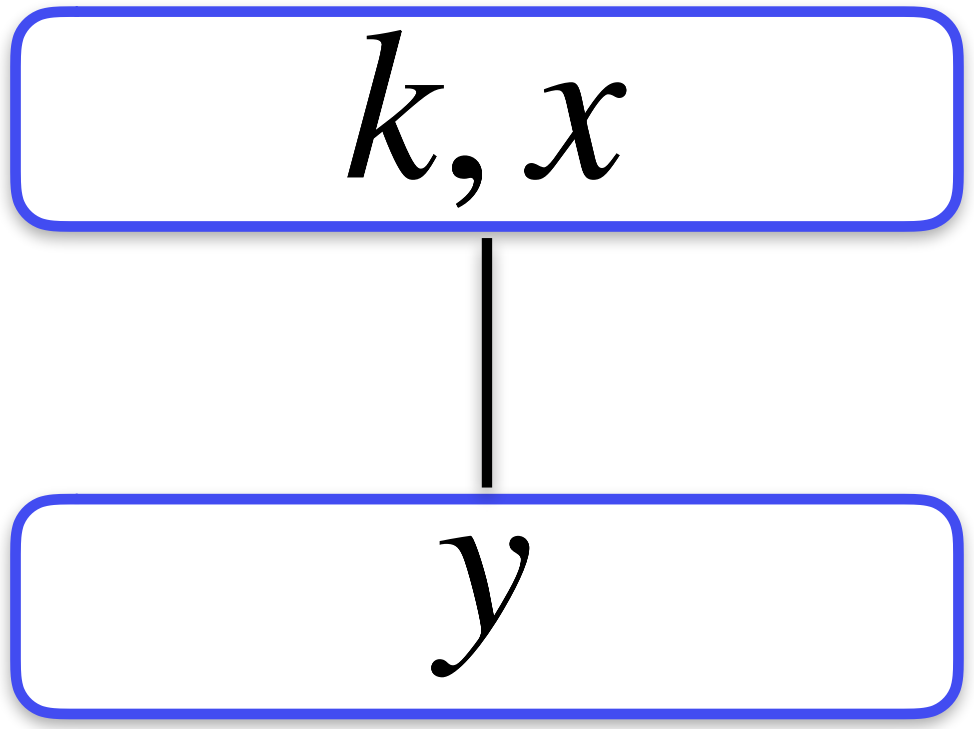
Pseudorandom generators

- Deterministic, polynomial time computable function satisfying:
  - **Length-expansion**: for all $x \in \{0,1\}^* \; |PRG(x)| = 2|x|$
  - **Pseudorandomness**: the output from the PRG should be indistinguishable from random



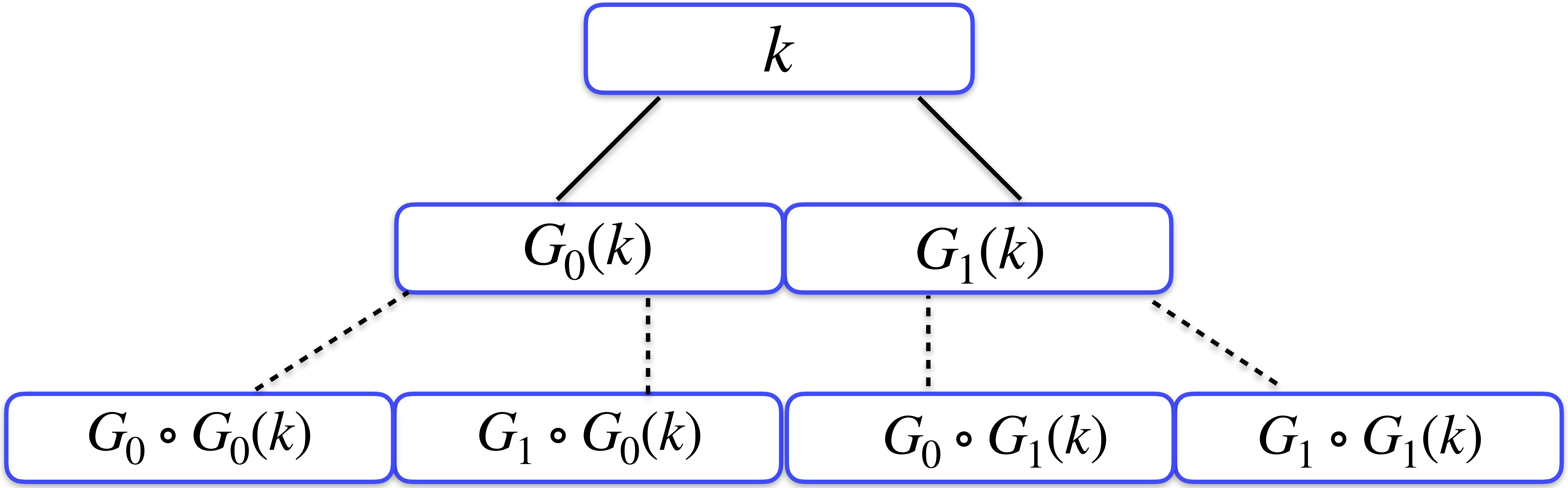
Pseudorandom functions

- Deterministic, polynomial time computable function satisfying:
  - **Length-condition**: for all $n \in \mathbb{N}, k, x \in \{0,1\}^n, \ |PRF(k,x)| = |y|$
  - **Pseudorandomness**: the output from the PRF should be indistinguishable from random



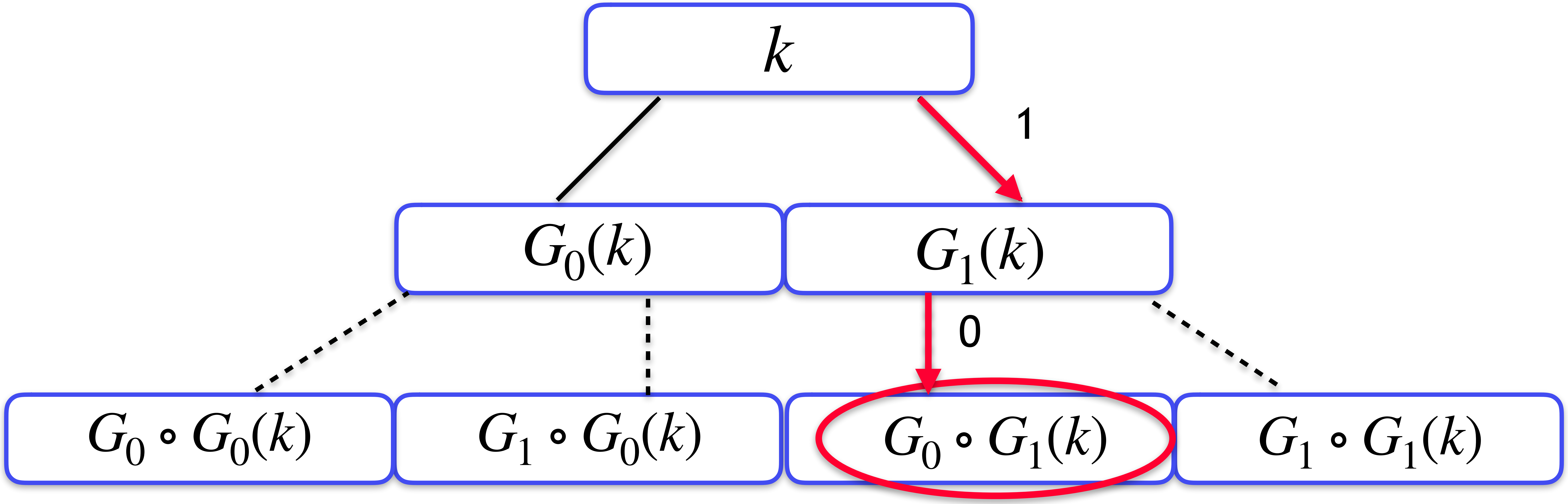
GGM tree: building a PRF from a PRG

- Introduced by Goldreich, Goldwasser and Micali
- Input x of the PRF(k,x) represents the binary address of the binary tree



GGM tree

- E.g. *x*= 10
- PRF(k,x)= GGM(k,m)= $G_0 \circ G_1(k)$



RSAConference2019

An incompressible white-box pseudorandom function

(Incompressible) PRF implementation

- Build a PRF which uses a large, incompressible key
  - **Key expansion**

$$k \in \{0,1\}^*$$

$$K = Comp_{PRF}(k), \text{with } |K| >> |k|$$

  - **Functionality preservation:**

$$\forall k, x \in \{0,1\}^*, f(k,x) = F(K,x)$$

Construction (1) - PRF

$$\underline{f(k, x)}$$

$$y \leftarrow \mathtt{GGM}(k, x)$$

$$\textbf{return} \quad y$$

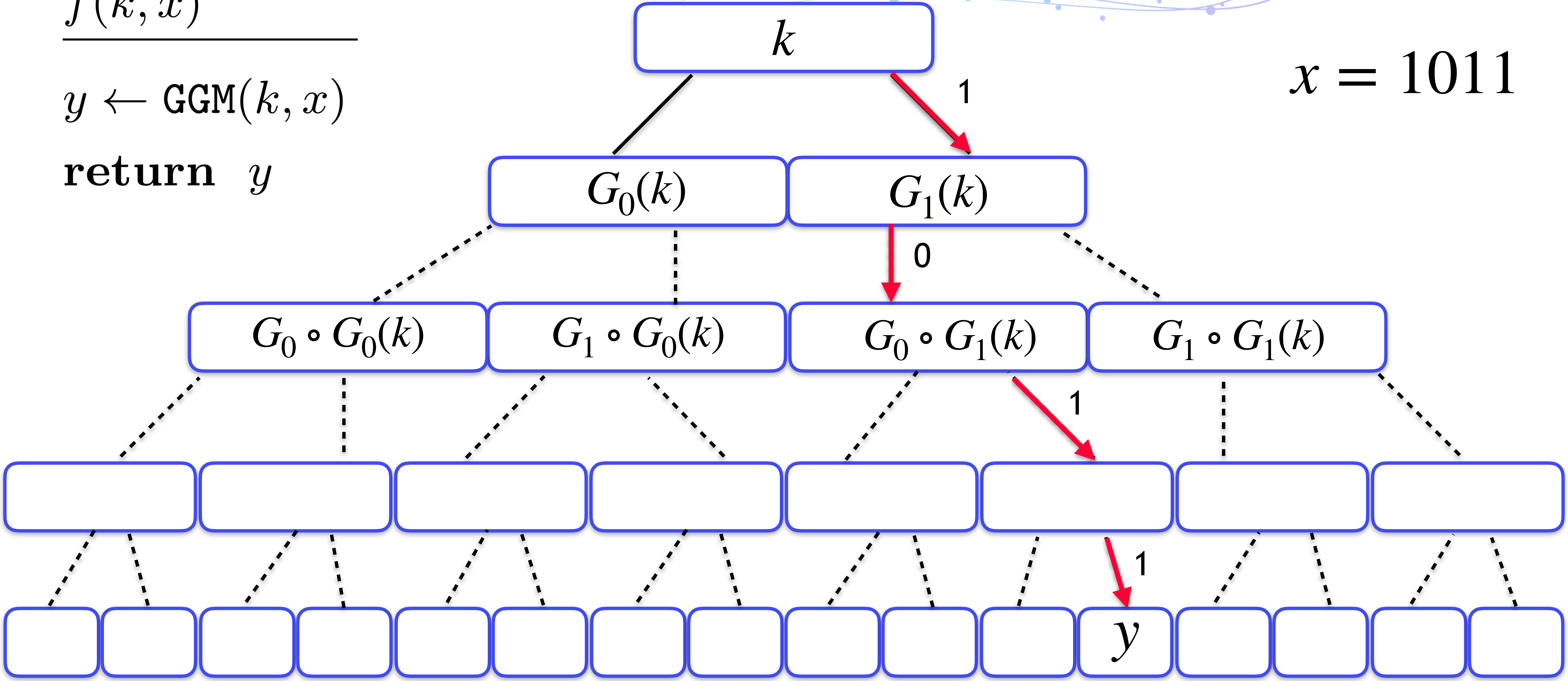
- Standard PRF based on the GGM tree

$$\underline{f(k, x)}$$

$y \leftarrow \mathtt{GGM}(k, x)$

return y

$x = 1011$



Construction (2) - Compiler

$\mathtt{Comp}_{\mathtt{PRF}}(k)$

$\textbf{for } j \textbf{ from } 0 \textbf{ to } 2^{\ell} - 1$

$\quad k_j := \mathtt{GGM}(k, < j >)$

$K \leftarrow k_0 || \ldots || k_{2^{\ell}-1}$

$\textbf{return} \quad K$

- Iterate the GGM on key *k* and all possible values of length ℓ

$\texttt{Comp}_{\texttt{PRF}}(k)$

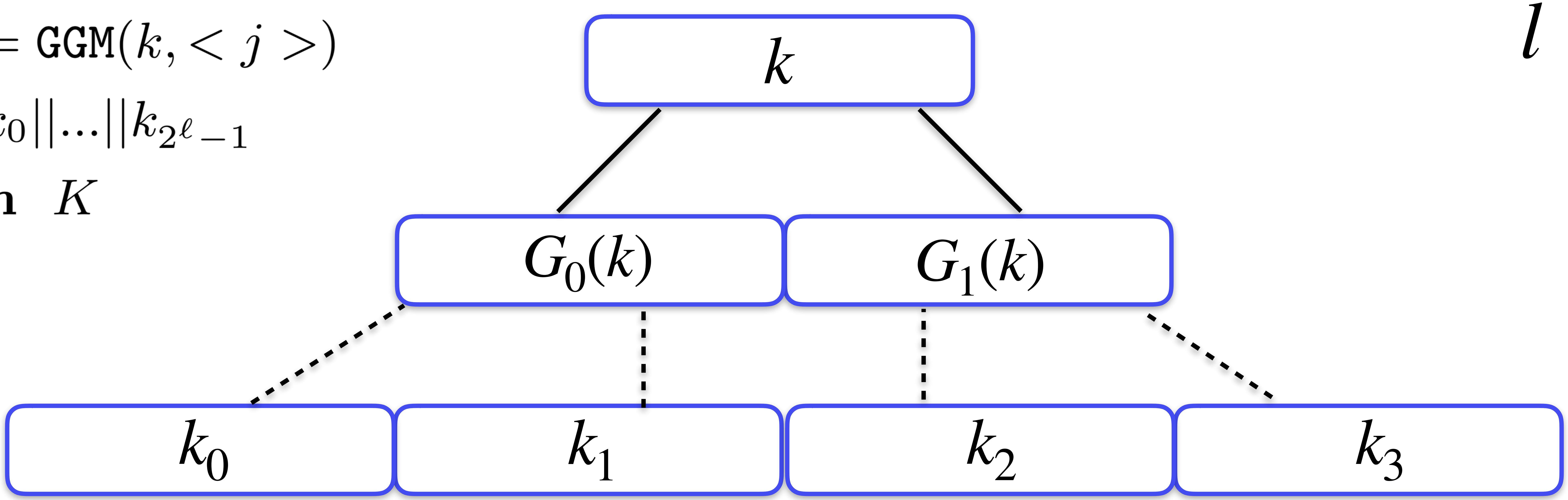
for j **from** 0 **to** $2^{\ell} - 1$

$\quad k_j := \texttt{GGM}(k, < j >)$

$K \leftarrow k_0 || ... || k_{2^{\ell}-1}$

return K

$l = 2$



$$K = k_0 || k_1 || k_2 || k_3$$

Construction (3) - Incompressible PRF

$$\underline{F(K,x)}$$

$$(x[1...\ell], x[\ell+1...|x|]) \leftarrow x$$

$$j \leftarrow x[1...\ell]$$

$$y \leftarrow \mathtt{GGM}(k_j, x[\ell+1...|x|])$$

$$\textbf{return} \quad y.$$

- F takes as input the long key K. Input x is split in two.

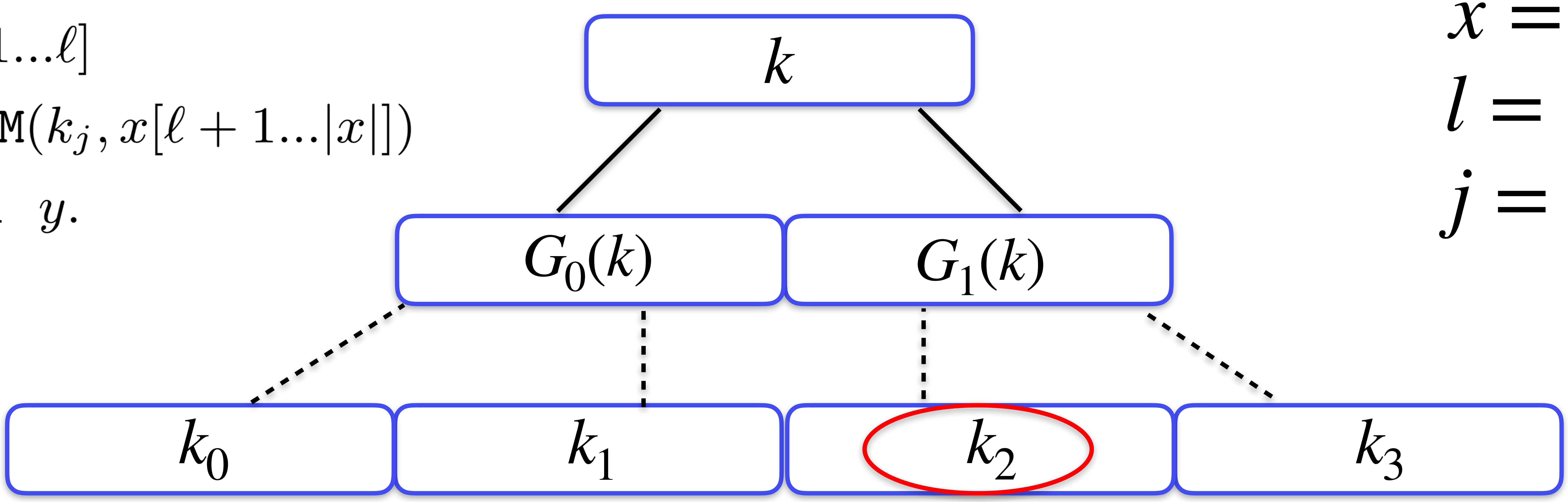
$F(K, x)$

$(x[1 \ldots \ell], x[\ell+1 \ldots |x|]) \leftarrow x$

$j \leftarrow x[1 \ldots \ell]$

$y \leftarrow \mathtt{GGM}(k_j, x[\ell+1 \ldots |x|])$

$\textbf{return} \quad y.$



$x = 1011$

$l = 2$

$j = 10$

GGM$(k_2, 11)$

Aalto University

$F(K, x)$

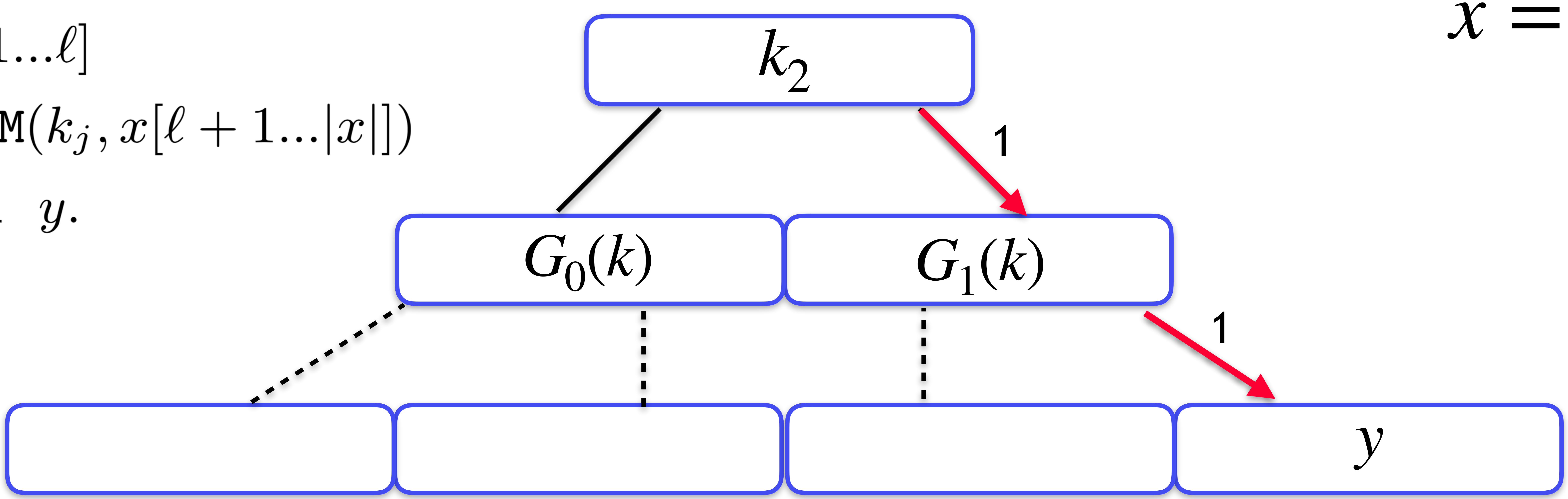
$(x[1...\ell], x[\ell+1...|x|]) \leftarrow x$

$j \leftarrow x[1...\ell]$

$y \leftarrow \mathtt{GGM}(k_j, x[\ell+1...|x|])$

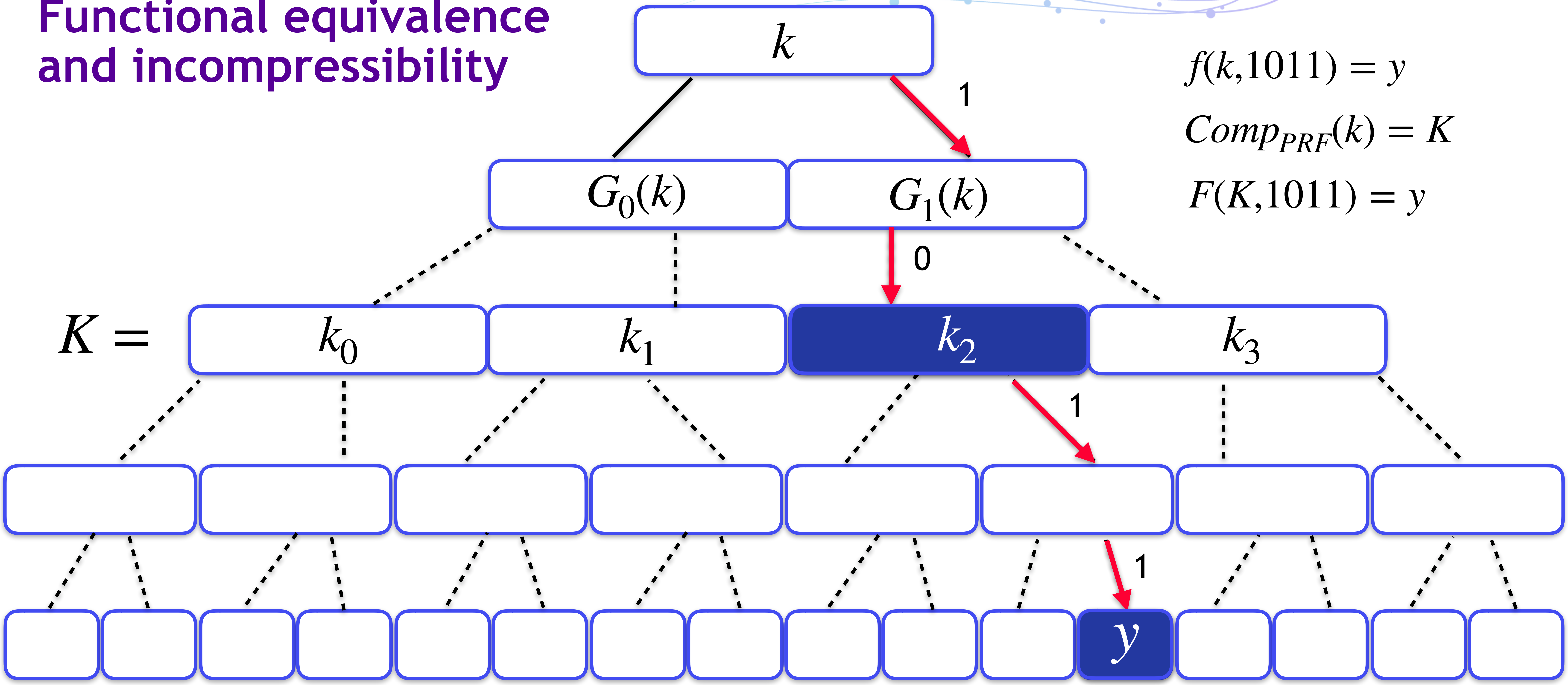
return y.

$x = 10\underline{11}$



$$y \leftarrow \mathrm{GGM}(k_2, 11)$$

Functional equivalence and incompressibility



$f(k, 1011) = y$

$Comp_{PRF}(k) = K$

$F(K, 1011) = y$

Possible collisions

$$k = k_a \,||\, \cancel{k_b} \qquad\qquad k' = k_a \,||\, \cancel{k_c}$$

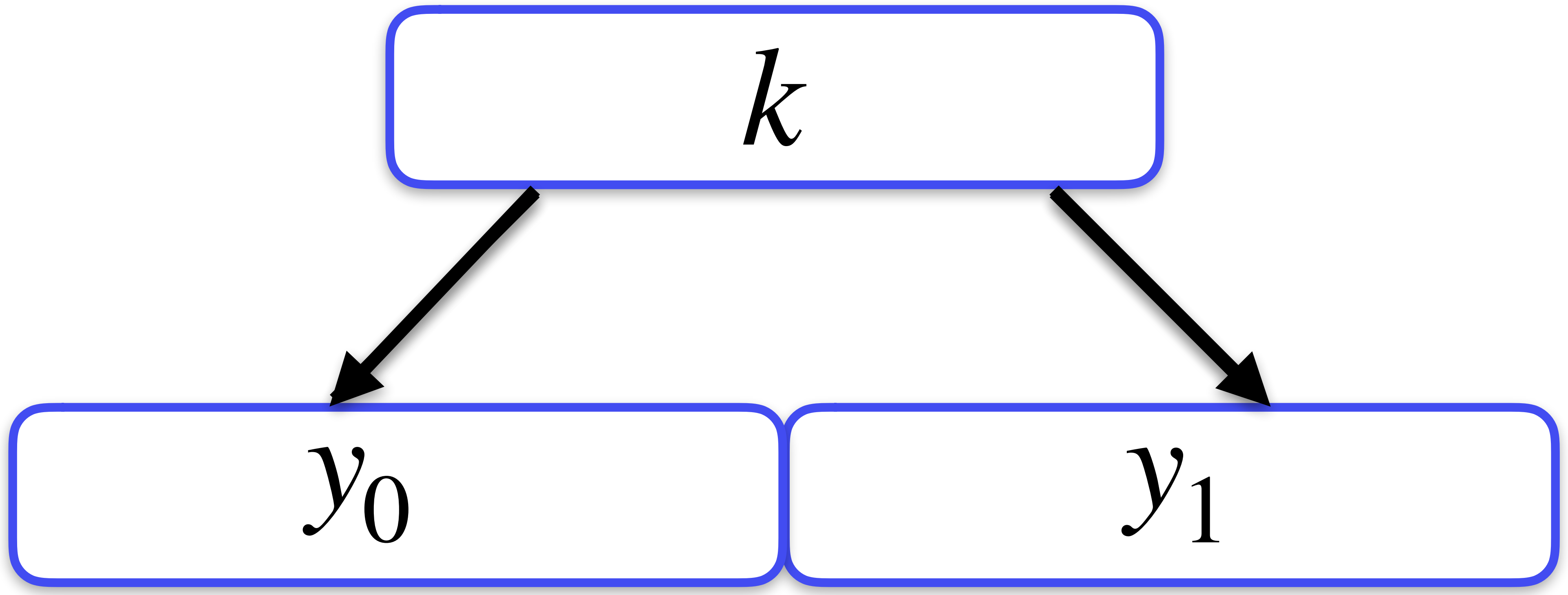
$$\boxed{y_0}\,\boxed{y_1} \quad = \quad \boxed{y_0}\,\boxed{y_1}$$

For our incompressibility property to hold, we need injectivity

RSAConference2019

Doubly-half injective PRGs

PRG with double injectivity



We want injectivity from

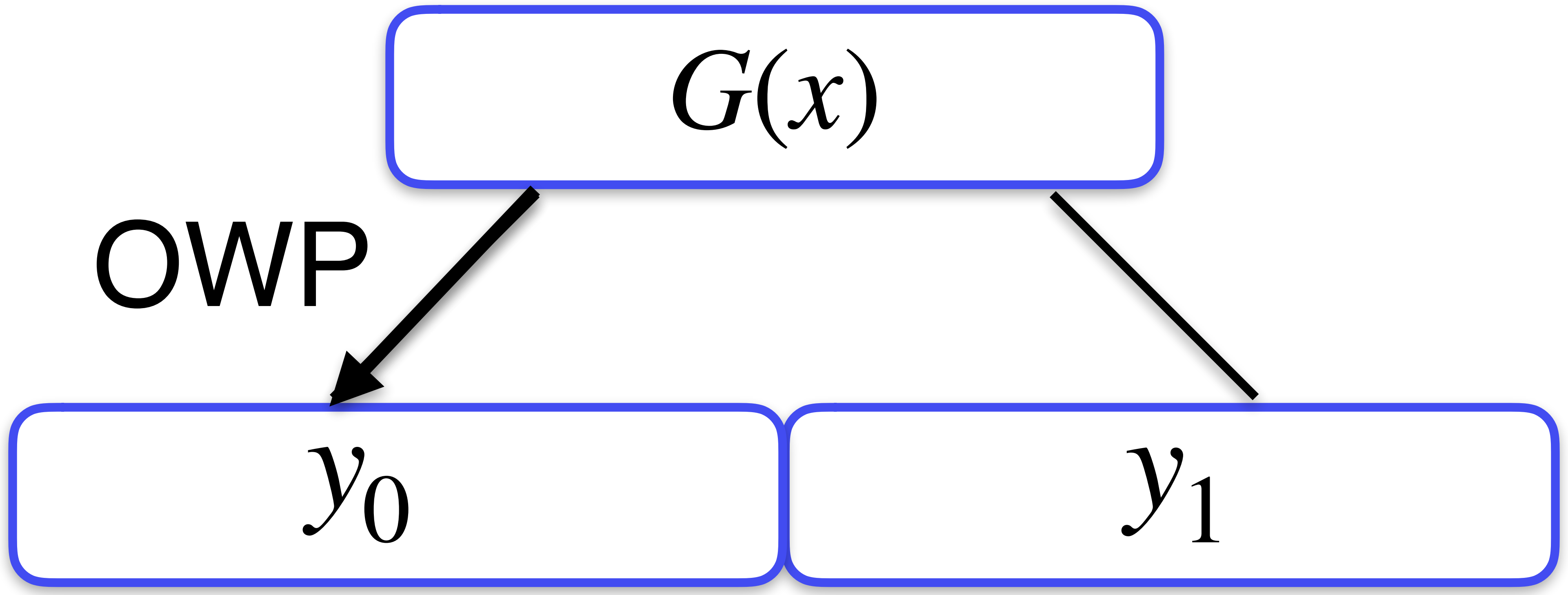
L to the set Y, with $k \in L$ and $y_0, y_1 \in Y$.

Aalto University

Left-half-injective PRG

- Construction by Garg, Pandey, Srinivasan and Zhandry: use a one-way permutation to construct a left-half injective PRG

  Breaking the sub-exponential barrier in obfustopia



$$G(x) := \mathsf{OWP}^{|x|}(x)\,||\,B(x)\,||\,B(\mathsf{OWP}(x))\,||\,\ldots\,||\,B(\mathsf{OWP}^{|x|-1}(x)),$$

with $B =$ hardcore bit

Aalto University

Doubly-half injective PRG

- Assuming a left-half injective, length doubling PRG

$$G = G_0 || G_1$$

OWP-*Injective*

$$g(x_0||x_1) := G_0(x_0) || G_1(x_0) \oplus G_0(x_1) || G_0(x_1) || G_1(x_1) \oplus G_0(x_0)$$

Injective *Injective*

Doubly-half injective PRG

$$g(x_0||x_1) := G_0(x_0)||G_1(x_0) \oplus G_0(x_x)||G_0(x_1)||G_1(x_1) \oplus G_0(x_0)$$

- Left half is injective,

Let $w_0||w_1$, s.t. $g_0(w_0||w_1) = g_0(x_0||x_1)$

G_0 is a permutation $\rightarrow x_0 = w_0$

$\cancel{G_1(w_0)} \oplus G_0(w_1) = \cancel{G_1(x_0)} \oplus G_0(x_1)$

G_0 is a permutation $\rightarrow x_1 = w_1$

The injectivity of the right half follows analogously

RSAConference2019

Conclusions

Overview of our construction

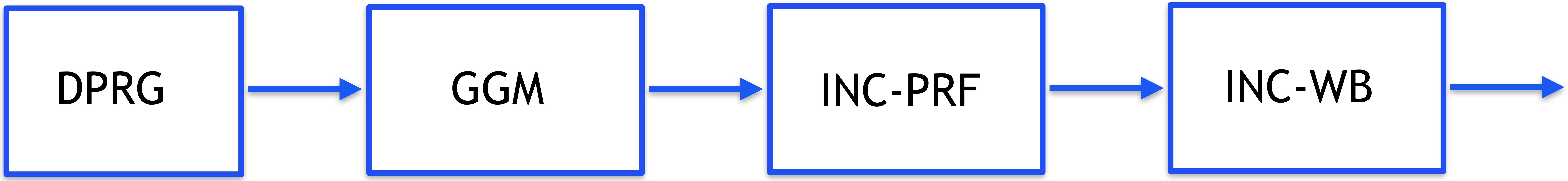
DPRG → GGM → INC-PRF → INC-WB →

- Provide an incompressible (big key) white-box encryption scheme
- Results based on standard crypto-assumptions
- Construct a new type of PRG

Backup slides

Conclusions

- Provide an incompressible (big key) white-box encryption scheme
- Results based on standard crypto-assumptions
- Construct a new type of PRG



Aalto University

Alternative desirable properties

- Making a program traceable (*traceability*)
- Binding the WB to a precise hardware device (*hardware binding*)
- Making the functionality of the WB dependent of a set of inputs (*input binding/application binding*)

Why is F incompressible?

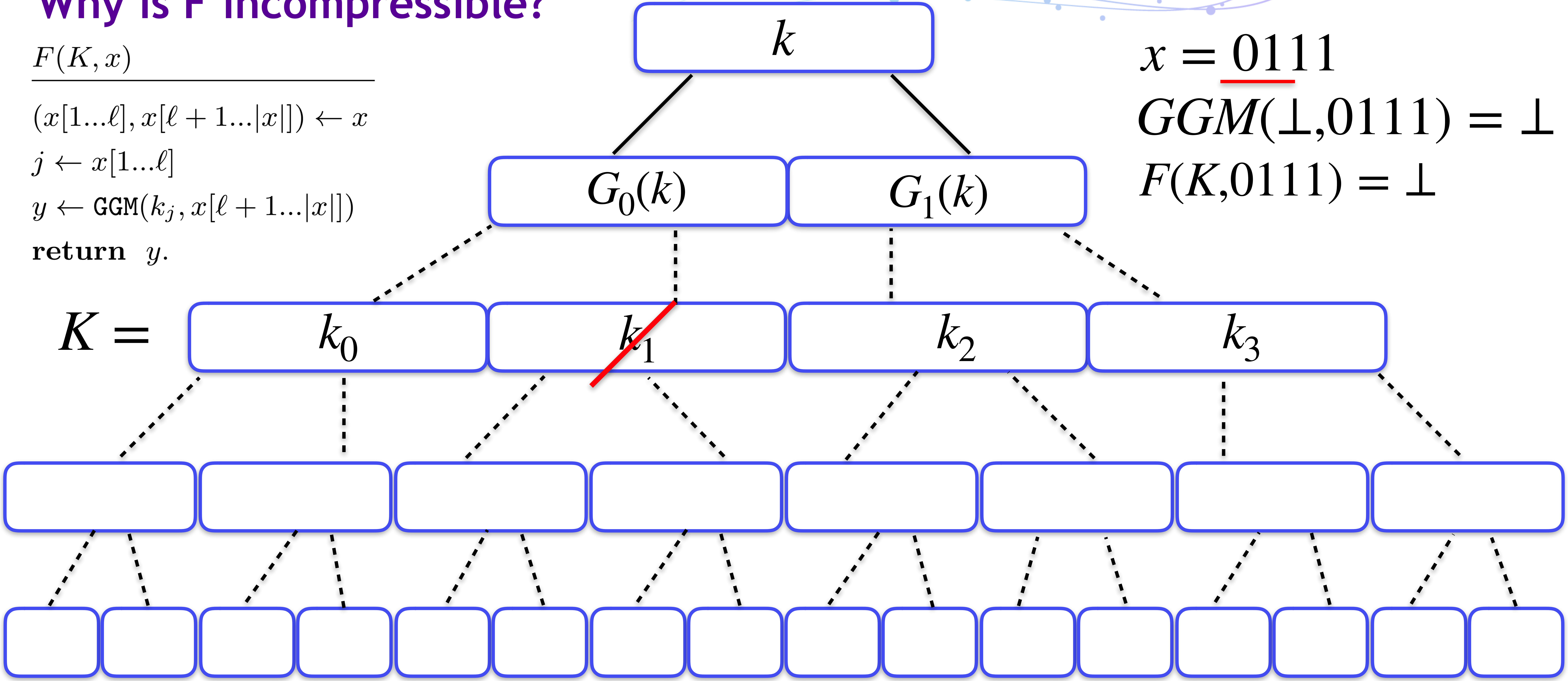
$F(K, x)$

$(x[1...\ell], x[\ell+1...|x|]) \leftarrow x$

$j \leftarrow x[1...\ell]$

$y \leftarrow \texttt{GGM}(k_j, x[\ell+1...|x|])$

return y.



$x = \underline{01}11$

$GGM(\perp, 0111) = \perp$

$F(K, 0111) = \perp$

Why is F incompressible?

$f(k,x)$

$y \leftarrow \texttt{GGM}(k,x)$

$\textbf{return} \quad y$

$\texttt{Comp}_{\texttt{PRF}}(k)$

$\textbf{for } j \textbf{ from } 0 \textbf{ to } 2^\ell - 1$

$\quad k_j := \texttt{GGM}(k, <j>)$

$K \leftarrow k_0 || ... || k_{2^\ell - 1}$

$\textbf{return} \quad K$

$F(K,x)$

$(x[1...\ell], x[\ell+1...|x|]) \leftarrow x$

$j \leftarrow x[1...\ell]$

$y \leftarrow \texttt{GGM}(k_j, x[\ell+1...|x|])$

$\textbf{return} \quad y.$

We need the complete key K to achieve $f(k,x) = F(K,x)$ for all $x \in \{0,1\}^*$

However, this might only hold depending on the definition of the PRG used in the GGM tree.

Aalto University

Theorem 1

IF PRF admits a computationally (σ, λ) – incompressible implementation F, the wb-encryption scheme in Constructino 1 is a $(\sigma, \lambda - n - o(1))$– incompressible wb-encryption scheme.

- Proof sketch via reduction: we reduce the incompressibility of F to the incompressibility of the encryption scheme.
  - Cannot produce a valid MAC without the complete key K

Doubly-half injective PRG

- We define a PRG which is left-half and right-half injective.
- Three properties required:
  - **Length-doubling:** For all $x \in \{0,1\}^* \, |g(x)| = 2|x|$. $g_0(x)$ is the left haf of g and $g_1(x)$is the right half.
  - **Doubly-half injective:** g_0 and g_1 are injective.
  - **Pseudorandomness:** $g(U_n)$ is computationally indistinguishable fromU_{2n}.

Aalto University

Construction 1 via AE-scheme and F

$\mathtt{Kgen}(1^n)$

$$
\begin{array}{l}
k' \leftarrow_{\$} \{0,1\}^n \\
k'' \leftarrow_{\$} \{0,1\}^n \\
k \leftarrow k' || k'' \\
\textbf{return } k
\end{array}
$$

$\mathtt{Enc}(k, m)$

$$
\begin{array}{l}
k' \leftarrow k[0 : n-1] \\
k'' \leftarrow k[n : 2n-1] \\
t \leftarrow f(k', m) \\
\tau \leftarrow (m, t) \\
c \leftarrow_{\$} \mathtt{AEnc}(k'', \tau) \\
\textbf{return } c
\end{array}
$$

$\mathtt{Dec}(k, c)$

$$
\begin{array}{l}
k' \leftarrow k[0 : n-1] \\
k'' \leftarrow k[n : 2n-1] \\
\tau \leftarrow \mathtt{ADec}(k'', c) \\
(m, t) \leftarrow \tau \\
\textbf{if } t = f(k', m) \textbf{ return } m. \\
\textbf{else return } \bot
\end{array}
$$

$\mathtt{Comp}(k)$

$$
\begin{array}{l}
k' \leftarrow k[0 : n-1] \\
k'' \leftarrow k[n : 2n-1] \\
K := \mathtt{Comp}_{\mathtt{PRF}}(k') \\
\mathtt{Enc}_{\mathtt{WB}} := C[K, k''](.) \\
\textbf{return } \mathtt{Enc}_{\mathtt{WB}}
\end{array}
$$

$C[K, k''](m)$

$$
\begin{array}{l}
t \leftarrow F(K, m) \\
\tau \leftarrow (m, t) \\
c \leftarrow_{\$} \mathtt{AEnc}(k'', \tau) \\
\textbf{return } c
\end{array}
$$

Use cases of white-box cryptography

- Original concern: Digital Rights Management
  - White-box crypto introduced as a method to mitigate piracy
    - Chow, Eisen Johnson and van Oorschot - A white-box cryptography and an AES implementation
- Recently proposed as a method for protecting cryptographic keys within mobile payment applications implemented in software

Global construction of the scheme

- Key expansion property
- Pseudorandomness property follows from the property of the GGM

$$
\begin{array}{l}
\underline{f(k,x)} \\
y \leftarrow \mathtt{GGM}(k,x) \\
\textbf{return} \quad y
\end{array}
\qquad
\begin{array}{l}
\underline{\mathtt{Comp}_{\mathtt{PRF}}(k)} \\
\textbf{for } j \textbf{ from } 0 \textbf{ to } 2^{\ell}-1 \\
\quad k_j := \mathtt{GGM}(k, <j>) \\
K \leftarrow k_0 || \ldots || k_{2^{\ell}-1} \\
\textbf{return} \quad K
\end{array}
\qquad
\begin{array}{l}
\underline{F(K,x)} \\
(x[1\ldots\ell], x[\ell+1\ldots|x|]) \leftarrow x \\
j \leftarrow x[1\ldots\ell] \\
y \leftarrow \mathtt{GGM}(k_j, x[\ell+1\ldots|x|]) \\
\textbf{return} \quad y.
\end{array}
$$

Aalto University

Methods for mitigating code-lifting attacks

- Two popular methods have been studied in the literature:
  - Traceability
    Delerablée, Lepoint, Paillier, Rivain: *White-box security notions for symmetric encryption schemes*