Watermarking and Traitor Tracing for PRFs

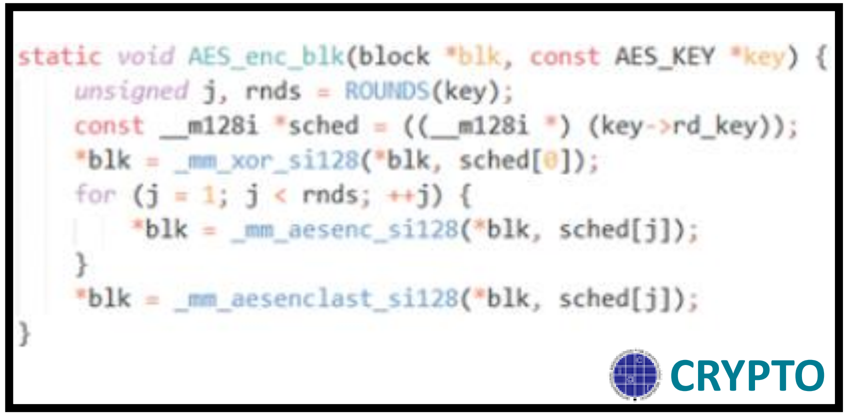
David Wu

April 2020

based on joint work with Rishab Goyal, Sam Kim, and Brent Waters

Software Watermarking

[NSS99, BGIRSVY01, HMW07, CHNVW16]



Embed a "mark" within a program

If mark is removed, then program is destroyed

Applications: proving software ownership, preventing unauthorized distribution of software

Software Watermarking

[NSS99, BGIRSVY01, HMW07, CHNVW16]

Embed a “mark” within a program

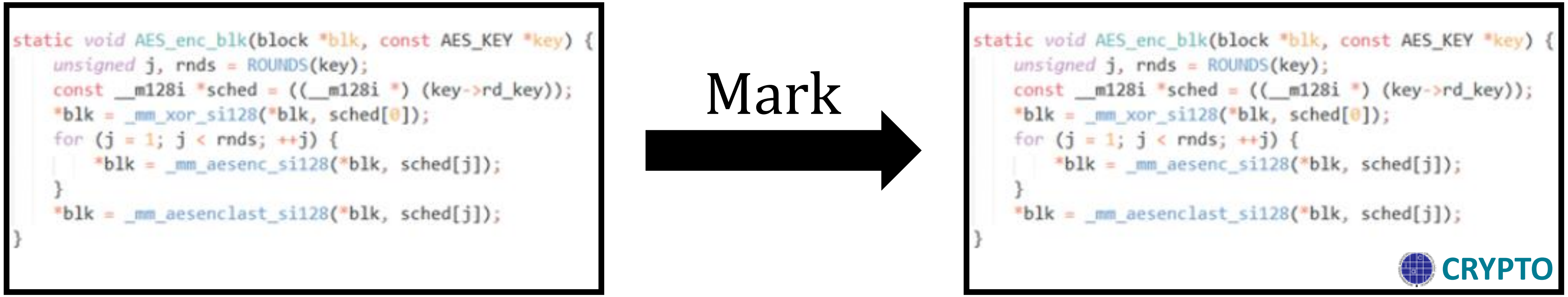
If mark is removed, then program is destroyed

Two main algorithms:

- $\mathrm{Mark}(C, m) \to C'$: Takes circuit C and mark m and outputs a marked circuit C'
- $\mathrm{Extract}(C') \to m/\bot$: Extracts the mark from a circuit C'

Software Watermarking

[NSS99, BGIRSVY01, HMW07, CHNVW16]



Functionality-preserving: On input a circuit C (and mark m), the Mark algorithm outputs a circuit C' where

$$C(x) = C'(x)$$

on almost all inputs x

Software Watermarking

[NSS99, BGIRSVY01, HMW07, CHNVW16]

Unremovability: Given a program C' with mark m, no efficient adversary can construct a circuit C^* where

- $C^*(x) = C'(x)$ on almost all inputs x
- The circuit C^* does not preserve the mark: $\text{Extract}(C^*) \neq m$

Software Watermarking

[NSS99, BGIRSVY01, HMW07, CHNVW16]

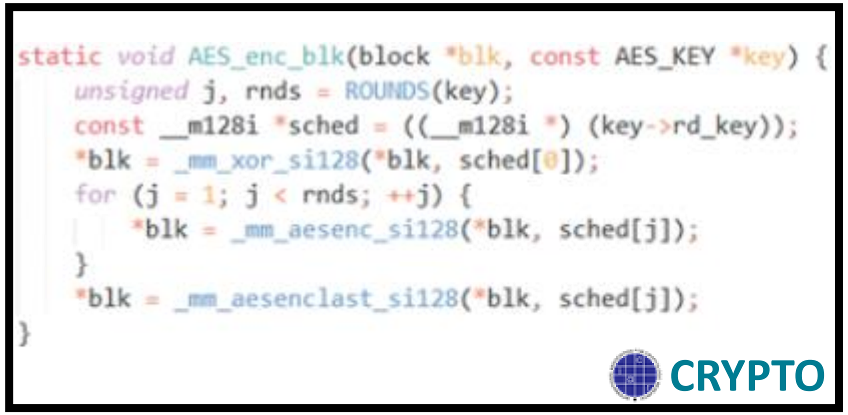
Adversary is very powerful: sees the code of the marked program C' and has complete flexibility in crafting C^*

Unremovability: Given a program C' with mark m, no efficient adversary can construct a circuit C^* where

- $C^*(x) = C'(x)$ on almost all inputs x
- The circuit C^* does not preserve the mark: $\text{Extract}(C^*) \neq m$

Software Watermarking

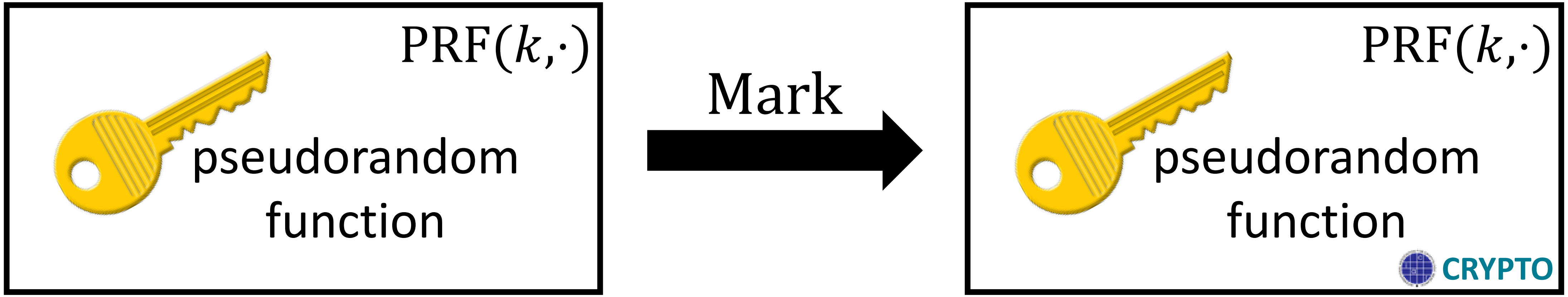
[NSS99, BGIRSVY01, HMW07, CHNVW16]



Learning the original (unmarked) function gives a way to remove the watermark

- Notion only achievable for functions that are not learnable
- Focus has been on cryptographic functions

Watermarking Cryptographic Programs

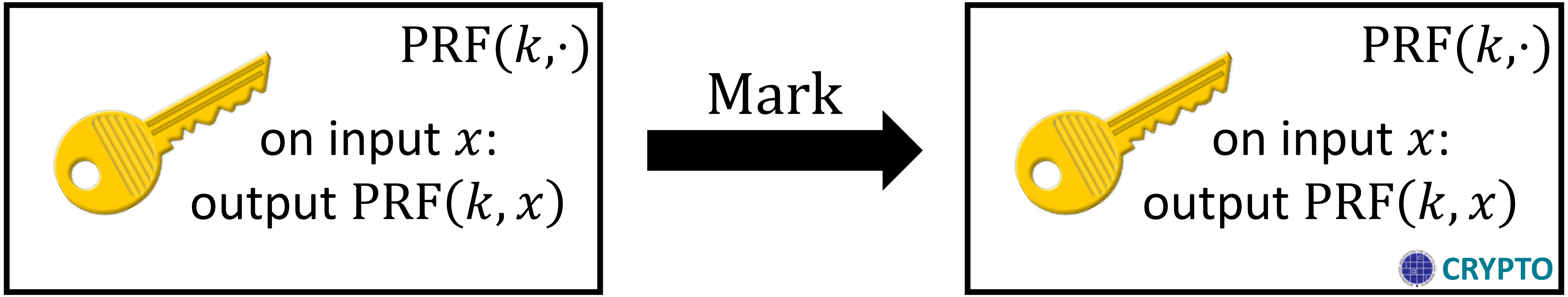


Previous works: watermarking PRFs [CHNVW16, BLW17, KW17, QWZ18, KW19]

Suffices for watermarking other symmetric primitives:

(e.g., MAC signing key, symmetric decryption key)

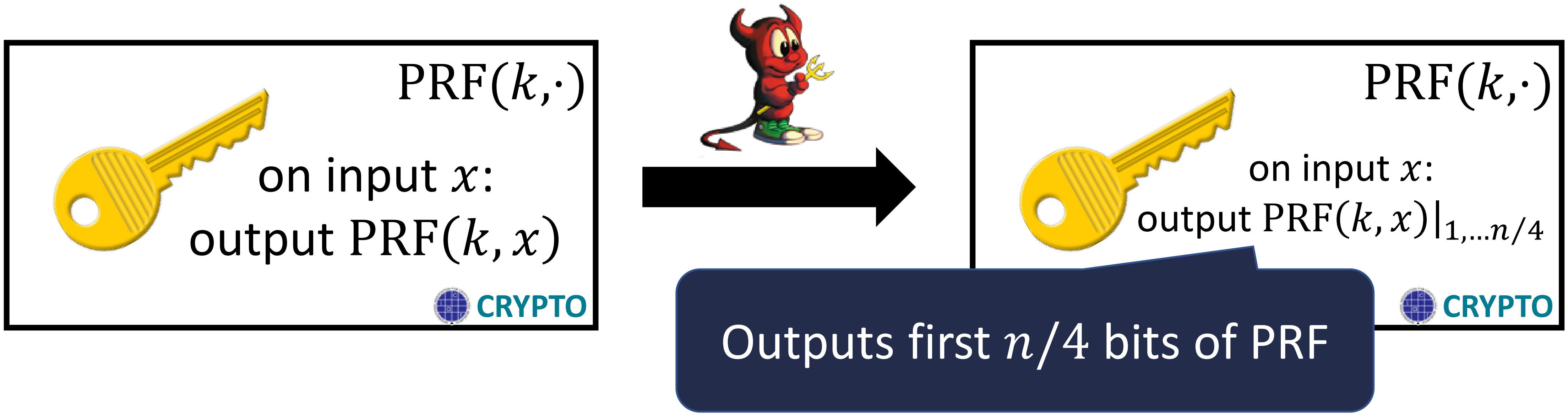
A Closer Look at Watermarking Security



Unremovability: Given a program C' with mark m, no efficient adversary can construct a circuit C^* where

- $C^*(x) = C'(x)$ on almost all inputs x
- The circuit C^* does not preserve the mark: $\text{Extract}(C^*) \neq m$

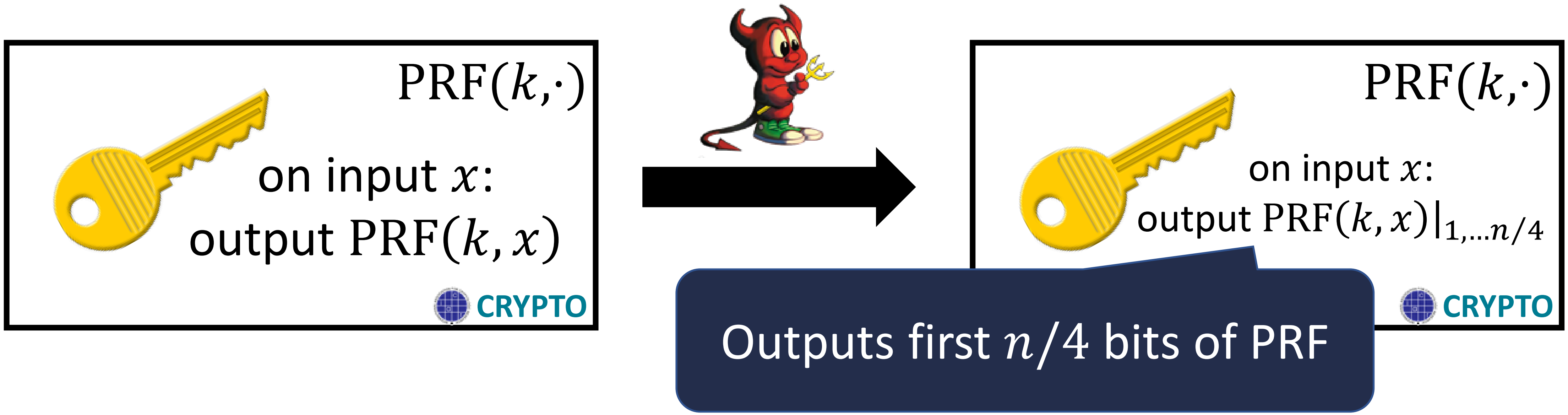
A Closer Look at Watermarking Security



Unremovability: Given a program C' with mark m, no efficient adversary can construct a circuit C^* where

- $C^*(x) = C'(x)$ on almost all inputs x
- The circuit C^* does not preserve the mark: $\mathrm{Extract}(C^*) \neq m$

A Closer Look at Watermarking Security

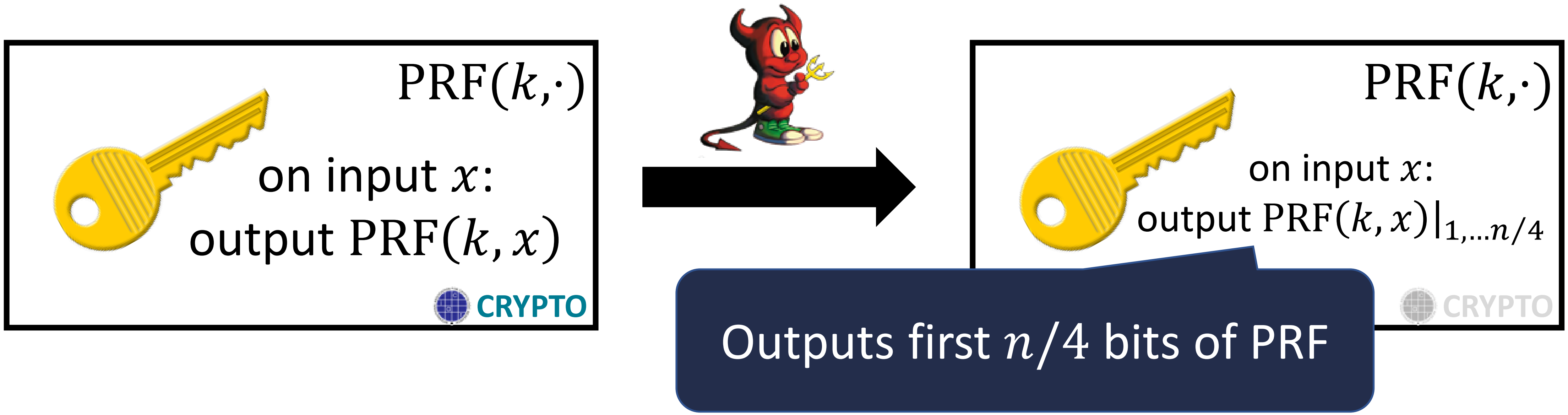


Unremovability: Given a program C' with mark m, no efficient adversary can construct a circuit C^* where

- $C^*(x) = C'(x)$ on almost all inputs x
- The circuit C^* does not preserve the mark: Extract$(C^*) \neq m$

Adversary's circuit does not preserve functionality

A Closer Look at Watermarking Security



Unremovability: Given a program C' with mark m, no efficient adversary can construct a circuit C^* where

- $C^*(x) = C'(x)$ on almost all inputs x
- The circuit C^* does not preserve the mark: $\mathrm{Extract}(C^*) \neq m$

Adversary's circuit does not preserve functionality

No guarantees on whether the mark is preserved or not!

A Closer Look at Watermarking Security

$\text{PRF}(k, \cdot)$

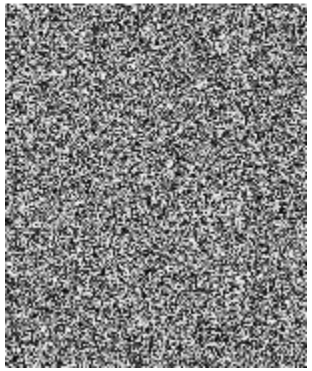
on input x:
output $\text{PRF}(k, x)|_{1,\ldots,n/4}$

Suppose circuit that only outputs leading $n/4$ bits does not contain the watermark

Is this a problem?

For building blocks like PRFs, we do not necessarily need to recover exact output to "break" functionality

Suppose watermarkable PRF used to protect against unauthorized distribution of decryption keys



Encrypted image
(PRF in counter mode)

Partial decryption
(using program on left)

Adversary's program is "good enough" in most settings, but may not preserve watermark

A Closer Look at Watermarking Security

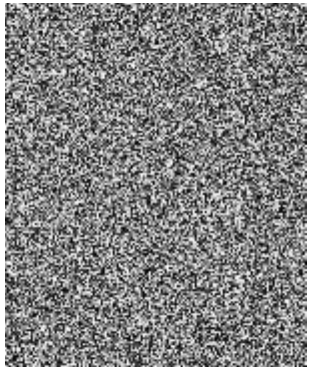
$\mathrm{PRF}(k, \cdot)$

on input x:
output $\mathrm{PRF}(k, x)|_{1,\ldots n/4}$

Watermarking cryptographic programs:

- Exact functionality preserving does not seem like the right security notion
- If adversary's program can break the primitive, then watermark should be preserved

Suppose watermarkable PRF used to protect against unauthorized distribution of decryption keys



Encrypted image
(PRF in counter mode)

Partial decryption
(using program on left)

Adversary's program is "good enough" in most settings, but may not preserve watermark

A Closer Look at Watermarking Security

$\mathrm{PRF}(k,\cdot)$

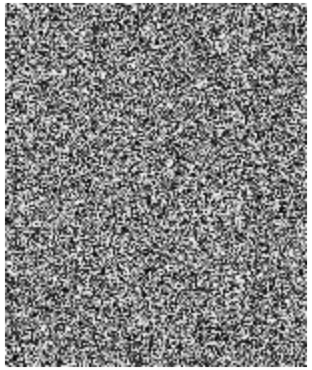
on input x:
output $\mathrm{PRF}(k,x)|_{1,\dots,n/4}$

Watermarking cryptographic programs:

ot

Existing watermarking constructions are <u>unable</u> to recover the watermark from this type of program

Suppose watermarkable PRF used to protect against unauthorized distribution of decryption keys

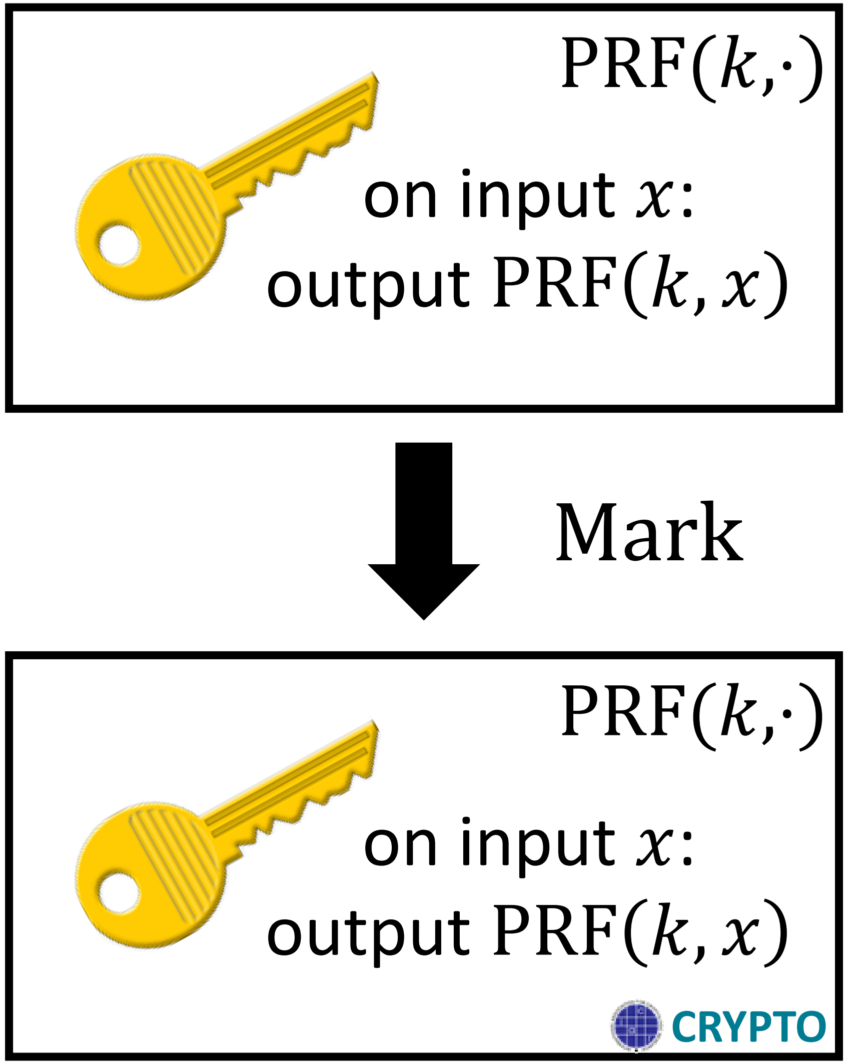


Encrypted image
(PRF in counter mode)

Partial decryption
(using program on left)

Adversary's program is "good enough" in most settings, but may <u>not</u> preserve watermark

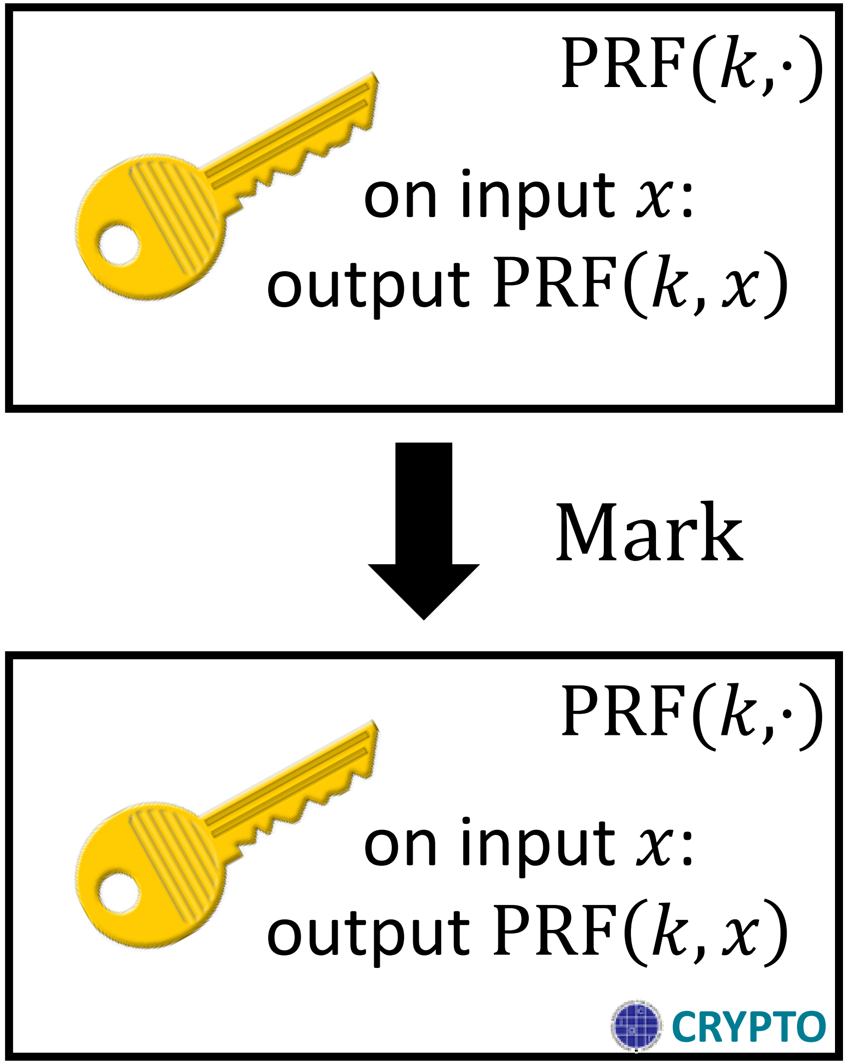
Traceable PRFs



PRF security:
PRF$(k,\cdot)$ indistinguishable from random function

Marking security (informal):
if program C can *distinguish* PRF$(k,\cdot)$ from random, then mark should be preserved

Traceable PRFs



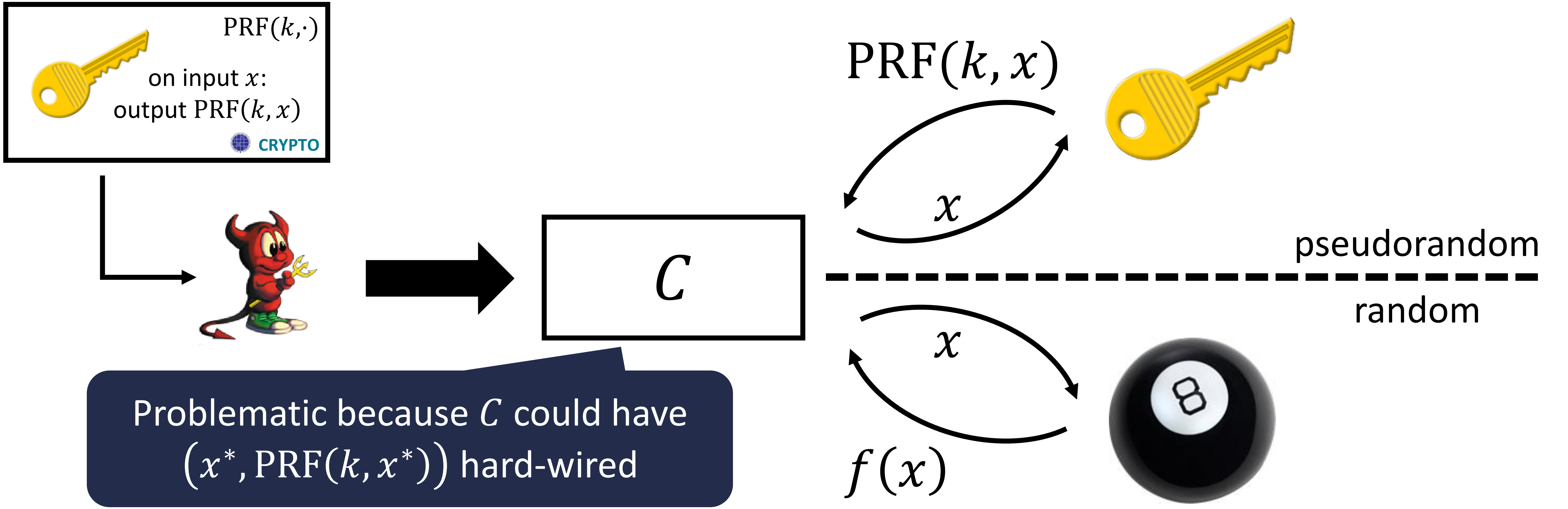
Traitor tracing: if program can distinguish ciphertexts, then mark is preserved

Traceable PRF: analog for PRFs

Marking security (informal):
if program C can *distinguish* $\mathrm{PRF}(k,\cdot)$ from random, then mark should be preserved

Traceable PRFs

Marking security (informal):
if program C can *distinguish* $\mathrm{PRF}(k,\cdot)$
from random, then mark should be preserved

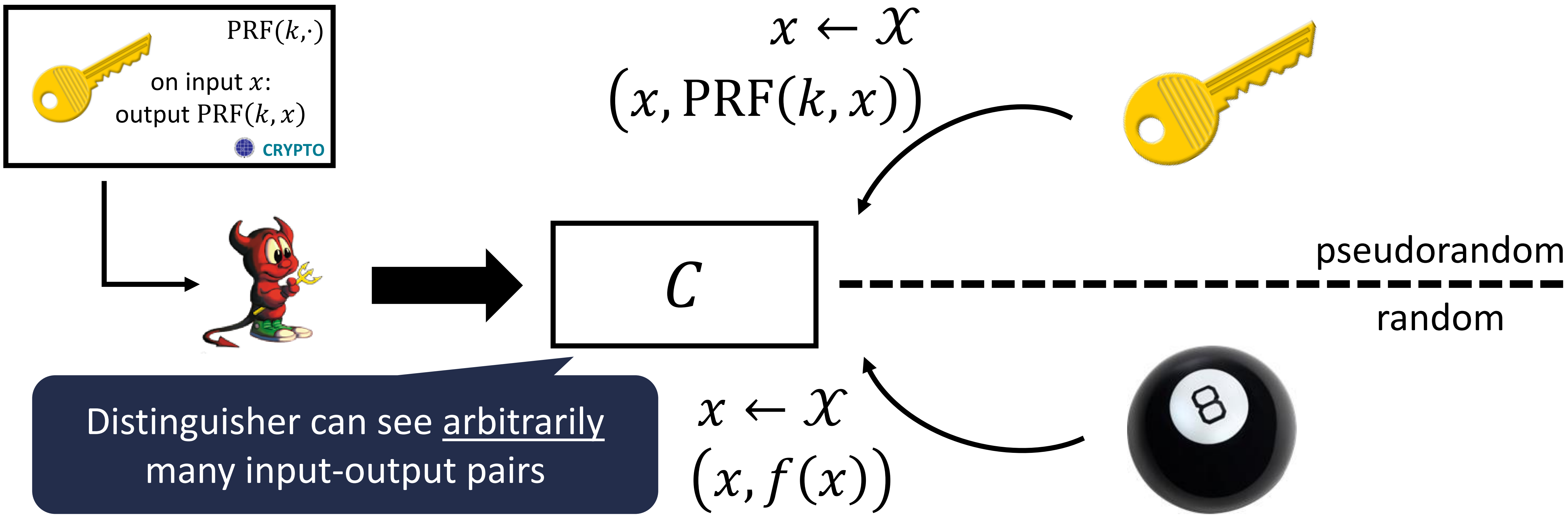


Traceable PRFs

Marking security (informal):

if program C can *distinguish* $\mathrm{PRF}(k,\cdot)$ from random

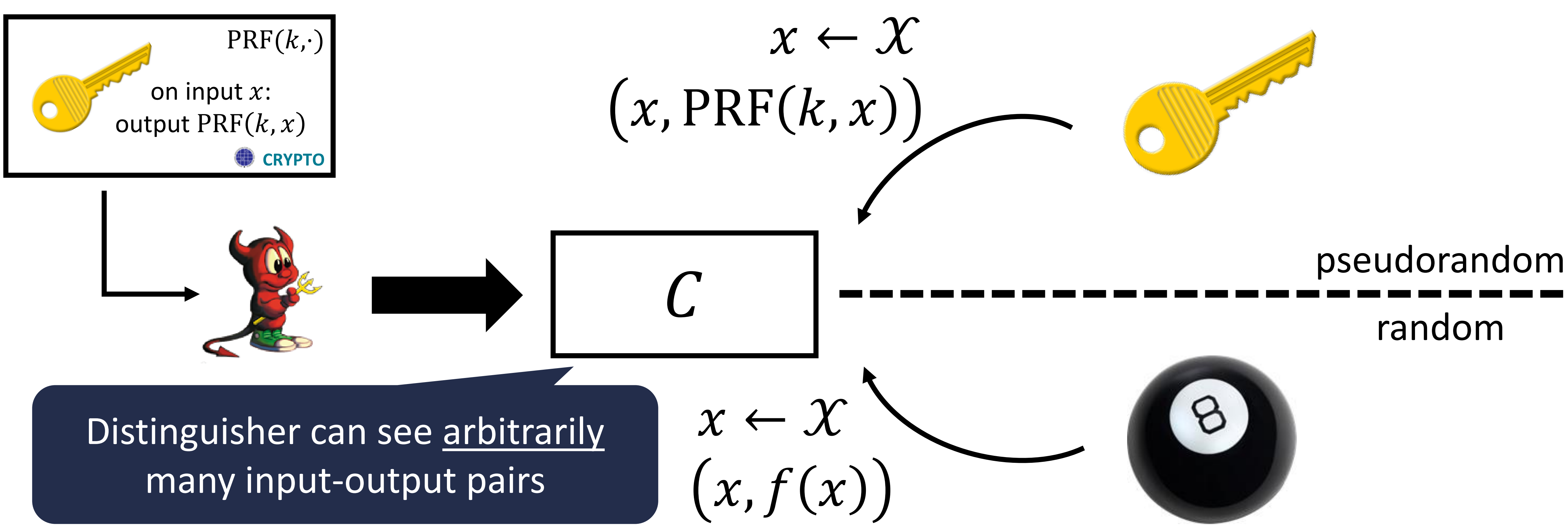
on randomly sampled inputs, then mark should be preserved



Traceable PRFs

Marking security (informal):

if program C can break weak pseudorandomness
of $\mathrm{PRF}(k,\cdot)$, then mark should be preserved



Traceable PRFs

$\mathsf{Setup}(1^\lambda) \to (\mathsf{msk}, \mathsf{tk})$

msk: master PRF key
tk: tracing key (can be public or secret)

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$

embeds $\mathsf{id} \in \{0,1\}^\ell$ into the key

$\mathsf{Eval}(\mathsf{sk}, x) \to y$

sk can be either msk or $\mathsf{sk}_{\mathsf{id}}$

$\mathsf{Trace}^D(\mathsf{tk}) \to T \subseteq \{0,1\}^\ell$

tracing algorithm given oracle access to weak PRF distinguisher

Traceable PRFs

$\mathrm{Setup}(1^\lambda) \to (\mathrm{msk}, \mathrm{tk})$

msk: master PRF key
tk: tracing key (can be public or secret)

Tracing key is sampled with PRF key (tracing algorithm needs to be able to sample PRF evaluations)

$\mathrm{KeyGen}(\mathrm{msk}, \mathrm{id}) \to \mathrm{sk}_{\mathrm{id}}$

$\mathrm{Eval}(\mathrm{sk}, x) \to y$

sk can be either msk or $\mathrm{sk}_{\mathrm{id}}$

$\mathrm{Trace}^D(\mathrm{tk}) \to T \subseteq \{0,1\}^\ell$

tracing algorithm given oracle access to weak PRF distinguisher

Traceable PRFs

Correctness: marked and unmarked keys agree almost everywhere

$$\Pr_{x \leftarrow \mathcal{X}}[\text{Eval}(\text{msk}, x) = \text{Eval}(\text{sk}_{\text{id}}, x)] = 1 - \text{negl}(\lambda)$$

Pseudorandomness: $\text{Eval}(\text{msk}, \cdot)$ is pseudorandom

Tracing Security:

id →

$\text{sk}_{\text{id}} \leftarrow \text{KeyGen}(\text{msk}, \text{id})$ ←

D →

if D breaks weak pseudorandomness of $\text{Eval}(\text{msk}, \cdot)$ with advantage ε, then $\text{Trace}^{D}(\text{tk})$ outputs id with probability $\approx \varepsilon$

single-key setting

Traceable PRFs

Traceable PRF directly implies secret-key traitor tracing (via nonce-based encryption)

$$\mathrm{Encrypt}(k,m) \coloneqq (r, \mathrm{PRF}(k,r) \oplus m)$$

Instantiate PRF with a traceable PRF

Not the case if we start with watermarkable PRF!

Tracing Security:

id $\rightarrow$

$\mathrm{sk}_{\mathrm{id}} \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{id})$ $\leftarrow$

D $\rightarrow$

single-key setting

if D breaks weak pseudorandomness of $\mathrm{Eval}(\mathrm{msk}, \cdot)$ with advantage ε, then $\mathrm{Trace}^D(\mathrm{tk})$ outputs id with probability $\approx \varepsilon$

Traceable PRFs

Our results:

Assuming LWE, there exists a single-key traceable PRF with secret tracing

This talk

Assuming indistinguishability obfuscation and injective one-way functions, there exists a fully collusion-resistant traceable PRF with public tracing

Notably: assumptions are the same as those needed for watermarkable PRFs (and rely on similar building blocks)

Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**

(analog of private linear broadcast encryption from traitor tracing) [BSW06]

Constrained PRF key: can be used to evaluate at all points $x \in \mathcal{X}$ where $C(x) = 1$

Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

PRF key

Privacy: index associated with a domain element is hidden

Linear constraint family:

- Some PRF inputs are associated with a (secret) index t between 0 and 2^ℓ
- Constrained key associated with $\text{id} \in [0, 2^\ell - 1]$ and can be used to evaluate on inputs whose index t satisfies $t \leq \text{id}$ (or no index)

Constructing Traceable PRFs

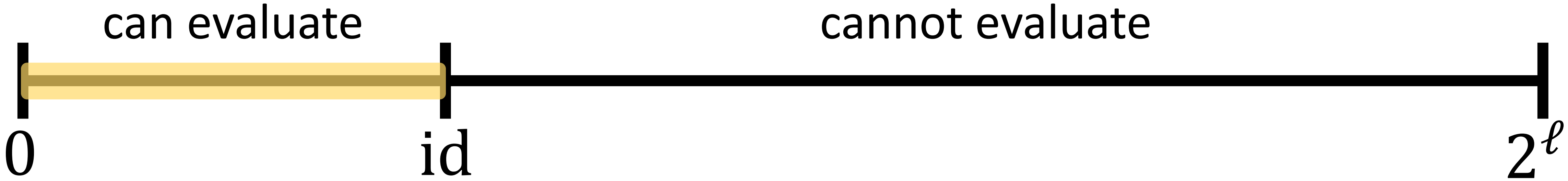
Rely on intermediate notion: **private linear constrained PRF**

(analog of private linear broadcast encryption from traitor tracing) [BSW06]

x input point

"hidden" index t

Can decrypt input points with indices $t \leq \mathrm{id}$



index t (for PRF domain element)

Constructing Traceable PRFs

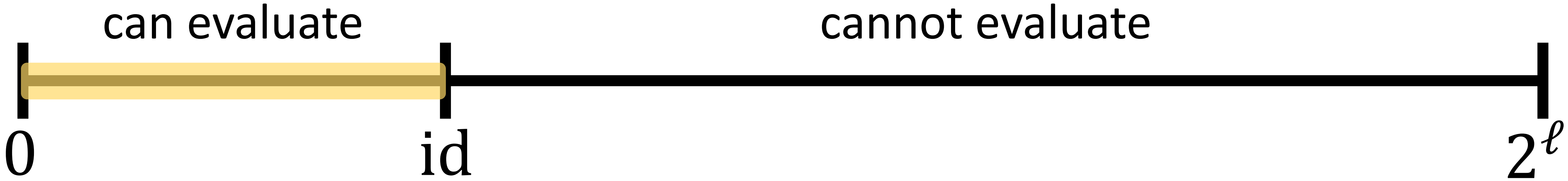
Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

Normal hiding: domain element with index 0 indistinguishable from random domain elements

Identity hiding: domain elements with index i and j are indistinguishable without key for $i \leq \mathrm{id} < j$

Pseudorandomness: PRF outputs on inputs with index 2^ℓ are pseudorandom

(all of the properties should hold given constrained keys)



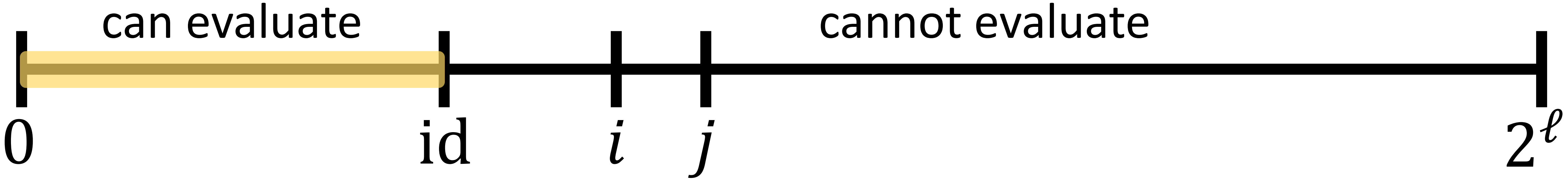
There exists a sampling algorithm to sample inputs with a specified index (could be secret-key algorithm)

Constructing Traceable PRFs

Tracing idea:

Assumption: Distinguisher D can break weak pseudorandomness with advantage ε

Implication: There must be a jump somewhere, and can only appear at id



Normal hiding

Inputs with index 0 are indistinguishable from random inputs, so decoder has advantage ε

Identity hiding

Distinguishing advantage changes negligibly when $\mathrm{id} \notin [i, j-1]$

Pseudorandomness

Inputs with index 2^ℓ are pseudorandom, so decoder has advantage 0

Constructing Private Linear Constrained PRF

Starting point: standard constrained PRF

Let domain $\mathcal{X} = \{0,1\}^{\ell}$

Problem: indices for domain element are <u>public</u>

$$C_{\text{id}}(t) = \begin{cases} 0, & t > \text{id} \\ 1, & t \le \text{id} \end{cases}$$

Can decrypt input points with tags $t \le \text{id}$

Constructing Private Linear Constrained PRF

Starting point: standard constrained PRF

Solution: Encrypt indices

Let domain $\mathcal{X} = \mathcal{CT}$ (ciphertext space for symmetric encryption scheme)

$$C_{k,\text{id}}(\text{ct}) = \begin{cases} 0, & \text{Decrypt}(k, \text{ct}) > \text{id} \\ 1, & \text{otherwise} \end{cases}$$

k: decryption key

Can decrypt input points corresponding to inputs that encrypt index greater than id

Constructing Private Linear Constrained PRF

Starting point: standard constrained PRF

Let domain $\mathcal{X} = \mathcal{CT}$

Problem: constrained key might leak k which leaks indices

$$C_{k,\text{id}}(\text{ct}) = \begin{cases} 0, & \text{Decrypt}(k, \text{ct}) > \text{id} \\ 1, & \text{otherwise} \end{cases}$$

k: decryption key

Can decrypt input points corresponding to inputs that encrypt index greater than id

Constructing Private Linear Constrained PRF

Starting point: standard constrained PRF

Let domain $\mathcal{X} = \mathcal{CT}$

Solution: use a <u>private</u> constrained PRF (constrained key hides constraint) [BLW17, CC17]

→ Constrain_C →

$$C_{k,\text{id}}(\text{ct}) = \begin{cases} 0, & \text{Decrypt}(k, \text{ct}) > \text{id} \\ 1, & \text{otherwise} \end{cases}$$

k: decryption key

Can decrypt input points corresponding to inputs that encrypt index greater than id

Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**

(analog of private linear broadcast encryption from traitor tracing) [BSW06]

Normal hiding: domain element with index 0 indistinguishable from random domain elements

Holds as long as encryption scheme has pseudorandom ciphertexts

(and constrained key hides secret key)

$$C_{k,\mathrm{id}}(\mathrm{ct}) = \begin{cases} 0, & \mathrm{Decrypt}(k, \mathrm{ct}) > \mathrm{id} \\ 1, & \text{otherwise} \end{cases}$$

Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**
(analog of private linear broadcast encryption from traitor tracing) [BSW06]

Identity hiding: domain elements with index i and j are indistinguishable without key for $i \leq \mathrm{id} < j$

Holds as long as encryption scheme is semantically secure
(and constrained key hides secret key)

$$C_{k,\mathrm{id}}(\mathrm{ct}) = \begin{cases} 0, & \mathrm{Decrypt}(k, \mathrm{ct}) > \mathrm{id} \\ 1, & \text{otherwise} \end{cases}$$

Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**

(analog of private linear broadcast encryption from traitor tracing) [BSW06]

Pseudorandomness: PRF outputs on inputs with index 2^ℓ are pseudorandom

Holds by constrained security of constrained PRF

(constraint function always false if $\mathrm{id} = 2^\ell$)

$$C_{k,\mathrm{id}}(\mathrm{ct}) = \begin{cases} 0, & \mathrm{Decrypt}(k, \mathrm{ct}) > \mathrm{id} \\ 1, & \text{otherwise} \end{cases}$$

Constructing Traceable PRFs

Rely on intermediate notion: **private linear constrained PRF**

(analog of private linear broadcast encryption from traitor tracing) [BSW06]

$\rightarrow$ Constrain_C $\rightarrow$

$$C_{k,\text{id}}(\text{ct}) = \begin{cases} 0, & \text{Decrypt}(k, \text{ct}) > \text{id} \\ 1, & \text{otherwise} \end{cases}$$

Public tracing: need a way to sample *PRF evaluations* (both inputs *and* outputs)

Unclear how to do so via private constrained PRFs, possible using indistinguishability obfuscation (with full collusion-resistance)

LWE

↓

single-key

private constrained PRF

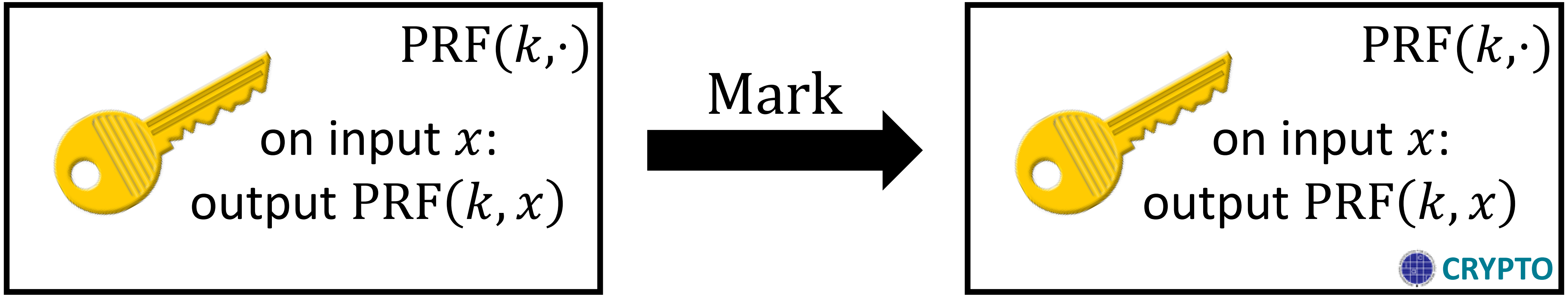
\+

symmetric encryption

→ single-key private linear constrained PRF (with secret sampling)

→ single-key traceable PRF (with secret tracing)

Traceable PRF Summary



Unremovability: Any program that can *distinguish* PRF outputs (on random inputs) must preserve the watermark

More generally: when considering software watermarking, should not always tie “functionality preserving” to “input-output preservation”

Traceable PRF Summary

Rely on intermediate notion: **private linear constrained PRF**

(analog of private linear broadcast encryption from traitor tracing) [BSW06]

→ Constrain_C →

$$C_{k,\text{id}}(\text{ct}) = \begin{cases} 0, & \text{Decrypt}(k, \text{ct}) > \text{id} \\ 1, & \text{otherwise} \end{cases}$$

LWE → single-key private constrained PRF + symmetric encryption → single-key private linear constrained PRF (with secret sampling) → single-key traceable PRF (with secret tracing)

https://eprint.iacr.org/2020/316

Private Constrained PRFs from Lattices

Overview of Brakerski-Vaikuntanathan and Brakerski-Tsabury-Vaikuntanathan-Wee constructions

Lattice-Based PRFs

[BPR12, BLMR13, BP14]

Learning with errors (LWE):

$$(\boldsymbol{A}, \boldsymbol{s}^T\boldsymbol{A} + \boldsymbol{e}^T) \approx (\boldsymbol{A}, \boldsymbol{u}^T)$$

$$\boldsymbol{A} \leftarrow \mathbb{Z}_q^{n\times m}, \boldsymbol{s} \leftarrow \mathbb{Z}_q^n, \boldsymbol{e} \leftarrow \chi^m, \boldsymbol{u} \leftarrow \mathbb{Z}_q^m$$

Learning with rounding (LWR) [BPR12]:

Replace error with <u>deterministic</u> rounding

$$\left(\boldsymbol{A}, \lfloor \boldsymbol{s}^T\boldsymbol{A} \rceil_p\right) \approx (\boldsymbol{A}, \boldsymbol{u}^T)$$

$$\boldsymbol{A} \leftarrow \mathbb{Z}_q^{n\times m}, \boldsymbol{s} \leftarrow \mathbb{Z}_q^n, \boldsymbol{u} \leftarrow \mathbb{Z}_p^m$$

Lattice-Based PRFs

[BPR12, BLMR13, BP14]

Learning with rounding (LWR) [BPR12]:

replace error with deterministic rounding

$$\left(\boldsymbol{A}, \lfloor \boldsymbol{s}^T \boldsymbol{A} \rceil_p\right) \approx (\boldsymbol{A}, \boldsymbol{u}^T)$$

General blueprint for lattice PRFs:

PRF family define by collection of public parameters: $\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell \in \mathbb{Z}_q^{n \times m}$

PRF key: $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$

PRF evaluation at x: $\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell, x \mapsto \boldsymbol{A}_x$

$$\mathrm{PRF}(\boldsymbol{s}, x) \coloneqq \lfloor \boldsymbol{s}^T \boldsymbol{A}_x \rceil_p$$

multiple ways to derive $\boldsymbol{A}_x$ from $\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell$

The GSW FHE Scheme

[GSW13]

pk: $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$ sk: $\boldsymbol{s} \in \mathbb{Z}_q^n$

Public key is an **LWE matrix** (columns are LWE samples)

$$\boldsymbol{s}^T\boldsymbol{A} = \boldsymbol{e}^T \approx \boldsymbol{0}^T$$

Ciphertext for $x \in \{0,1\}$:

$$\boldsymbol{A}_x = \boldsymbol{A}\boldsymbol{R} + x\boldsymbol{G}$$ where $\boldsymbol{R}$ is random short matrix

Decryption:

$$\boldsymbol{s}^T\boldsymbol{A}_x = \boldsymbol{s}^T\boldsymbol{A}\boldsymbol{R} + x \cdot \boldsymbol{s}^T\boldsymbol{G} \approx x \cdot \boldsymbol{s}^T\boldsymbol{G}$$

The GSW/BGG+ Homomorphisms

[GSW13, BGGHNSVV14]

$$\boldsymbol{A}_1 = \boldsymbol{A}\boldsymbol{R}_1 + x_1\boldsymbol{G} \quad \cdots \quad \boldsymbol{A}_\ell = \boldsymbol{A}\boldsymbol{R}_\ell + x_\ell\boldsymbol{G}$$

Input-independent evaluation:

$$\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell, f \mapsto \boldsymbol{A}_f$$

Function of $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell, f, x$

$$\boldsymbol{A}_f = \boldsymbol{A}\boldsymbol{R}_{f,x} + f(x)\boldsymbol{G} \quad \text{where} \quad \boldsymbol{R}_{f,x} = [\boldsymbol{R}_1 \mid \cdots \mid \boldsymbol{R}_\ell]\boldsymbol{H}_{f,x}$$

and $\boldsymbol{H}_{f,x}$ is short

Input-dependent evaluation:

$$[\boldsymbol{A}\boldsymbol{R}_1 \mid \cdots \mid \boldsymbol{A}\boldsymbol{R}_\ell]\boldsymbol{H}_{f,x} = \boldsymbol{A}\boldsymbol{R}_{f,x}$$

$$[\boldsymbol{A}_1 - x_1\boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - x_\ell\boldsymbol{G}]\boldsymbol{H}_{f,x} = \boldsymbol{A}_f - f(x)\boldsymbol{G}$$

Lattice-Based Constrained PRFs

[BV15]

Domain: $\mathcal{X} = \{0,1\}^\rho$

Let $U_x(f) \coloneqq f(x)$ be a universal circuit where $|f| = \ell$

Public parameters: $\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell \leftarrow \mathbb{Z}_q^{n \times m}$

PRF key: $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$

PRF evaluation at x:

$$\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell, x \mapsto \boldsymbol{A}_{U_x}$$

$$\mathrm{PRF}(\boldsymbol{s}, x) \coloneqq \lfloor \boldsymbol{s}^T \boldsymbol{A}_{U_x} \rceil_p$$

to argue pseudorandomness, need to also multiply by $G^{-1}(\boldsymbol{D})$ where $\boldsymbol{D}$ is part of public parameters

can evaluate at x where $f(x) = 0$

Constrained key for f:

$$\boldsymbol{s}^T [\boldsymbol{A}_1 - f_1 \cdot \boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - f_\ell \cdot \boldsymbol{G}] + \boldsymbol{e}^T$$

Constrained evaluation at x:

$$\boldsymbol{s}^T [\boldsymbol{A}_1 - f_1 \cdot \boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - f_\ell \cdot \boldsymbol{G}] \boldsymbol{H}_{U_x, f} + \boldsymbol{e}^T \boldsymbol{H}_{U_x, f}$$

$$\approx \boldsymbol{s}^T (\boldsymbol{A}_{U_x} - f(x) \cdot \boldsymbol{G})$$

$$= \boldsymbol{s}^T \boldsymbol{A}_{U_x} \quad \text{when} \quad f(x) = 0$$

Lattice-Based Constrained PRFs

[BV15]

Domain: $\mathcal{X} = \{0,1\}^\rho$

Let $U_x(f) \coloneqq f(x)$ be a universal circuit where $|f| = \ell$

Public parameters: $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell \leftarrow \mathbb{Z}_q^{n \times m}$

PRF key: $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$

PRF evaluation at x:

$$\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell, x \mapsto \boldsymbol{A}_{U_x}$$

$$\mathrm{PRF}(\boldsymbol{s}, x) \coloneqq \lfloor \boldsymbol{s}^T \boldsymbol{A}_{U_x} \rceil_p$$

Computing $\boldsymbol{H}_{U_x,f}$ requires knowledge of f (construction does not hide the constraint)

Constrained evaluation at x:

$$\boldsymbol{s}^T [\boldsymbol{A}_1 - f_1 \cdot \boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - f_\ell \cdot \boldsymbol{G}] \boldsymbol{H}_{U_x,f} + \boldsymbol{e}^T \boldsymbol{H}_{U_x,f}$$

$$\approx \boldsymbol{s}^T \left(\boldsymbol{A}_{U_x} - f(x) \cdot \boldsymbol{G} \right)$$

$$\approx \boldsymbol{s}^T \boldsymbol{A}_{U_x} \quad \text{when} \quad f(x) = 0$$

Lattice-Based Private Constrained PRFs

[BTVW17]

Approach: encrypt the function f using an FHE scheme, and homomorphically evaluate U_x

$$\hat{f} \coloneqq \text{Encrypt}(\text{pk}, f) \qquad \left|\hat{f}\right| = L$$

$$\widehat{U}_x\left(\hat{f}\right) \coloneqq \text{FHE.Eval}\left(\text{pk}, U_x, \hat{f}\right)$$ Homomorphic evaluation of U_x on f

Constrained key for f:

$$\boldsymbol{s}^T\left[\boldsymbol{A}_1 - \hat{f}_1 \cdot \boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - \hat{f}_L \cdot \boldsymbol{G}\right] + \boldsymbol{e}^T$$

Constrained evaluation at x:

$$\boldsymbol{s}^T\left[\boldsymbol{A}_1 - \hat{f}_1 \cdot \boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - \hat{f}_L \cdot \boldsymbol{G}\right]\boldsymbol{H}_{\widehat{U}_x,\hat{f}} + \boldsymbol{e}^T\boldsymbol{H}_{\widehat{U}_x,\hat{f}}$$

$$\approx \boldsymbol{s}^T\left(\boldsymbol{A}_{\widehat{U}_x} - \widehat{U}_x\left(\hat{f}\right) \cdot \boldsymbol{G}\right)$$

Problem: $\widehat{U}_x\left(\hat{f}\right)$ is a bit of the encryption of $f(x)$, not $f(x)$

Lattice-Based Private Constrained PRFs

[BTVW17]

Straightforward to generalize homomorphic operations to matrix-valued functions:

$$\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell, f \mapsto \boldsymbol{A}_f$$
$$[\boldsymbol{A}_1 - x_1\boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - x_\ell\boldsymbol{G}]\boldsymbol{H}_{f,x} = \boldsymbol{A}_f - f(x)\boldsymbol{G}$$

$$f: \{0,1\}^\ell \mapsto b \in \{0,1\}$$

$\Downarrow$

$$\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell, f \mapsto \boldsymbol{A}_f$$
$$[\boldsymbol{A}_1 - x_1\boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - x_\ell\boldsymbol{G}]\boldsymbol{H}_{f,x} = \boldsymbol{A}_f - \boldsymbol{X}_f$$

$$f: \{0,1\}^\ell \mapsto \boldsymbol{X}_f \in \mathbb{Z}_q^{n\times m}$$

Idea: compute $\boldsymbol{X}_f$ bit-by-bit, and multiply encoding of the k^{th} bit of the j^{th} component of $\boldsymbol{X}_f$ by $G^{-1}(2^k\boldsymbol{E}_j)$, where $\boldsymbol{E}_j$ is 1 in the j^{th} component and 0 everywhere else

The GSW FHE Scheme

[GSW13]

Recall GSW decryption:

Ciphertext for $x \in \{0,1\}$: $\boldsymbol{A}_x = \boldsymbol{AR} + x\boldsymbol{G}$

Decryption: $\boldsymbol{s}^T \boldsymbol{A}_x = \boldsymbol{s}^T \boldsymbol{AR} + x \cdot \boldsymbol{s}^T \boldsymbol{G} = x \cdot \boldsymbol{s}^T \boldsymbol{G} + \text{error}$

Property: Multiplying secret key with ciphertext yields encoding of the plaintext message

Lattice-Based Private Constrained PRFs

[BTVW17]

Approach: encrypt the function f using an FHE scheme, and homomorphically evaluate U_x

$$\hat{f} := \text{Encrypt}(\text{pk}, f) \qquad |\hat{f}| = L$$

$$\widehat{U}_x(\hat{f}) := \text{FHE.Eval}(\text{pk}, U_x, \hat{f})$$

Homomorphic evaluation of U_x on f

Define $\widehat{U}_x(\hat{f})$ to output the GSW ciphertext (matrix-valued) obtained from homomorphic evaluation

Constrained key for f:

$$\boldsymbol{s}^T[\boldsymbol{A}_1 - \hat{f}_1 \cdot \boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - \hat{f}_L \cdot \boldsymbol{G}] + \boldsymbol{e}^T$$

Constrained evaluation at x:

$$\boldsymbol{s}^T[\boldsymbol{A}_1 - \hat{f}_1 \cdot \boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - \hat{f}_L \cdot \boldsymbol{G}]\boldsymbol{H}_{\widehat{U}_x, \hat{f}} + \boldsymbol{e}^T \boldsymbol{H}_{\widehat{U}_x, \hat{f}}$$

$$\approx \boldsymbol{s}^T\left(\boldsymbol{A}_{\widehat{U}_x} - \widehat{U}_x(\hat{f})\right)$$

$$\approx \boldsymbol{s}^T(\boldsymbol{A}_{\widehat{U}_x} - f(x) \cdot \boldsymbol{G})$$

Insight: If $\boldsymbol{s}$ is also the secret key for the GSW encryption scheme, then

$$\boldsymbol{s}^T \widehat{U}_x(\hat{f}) = f(x) \cdot \boldsymbol{s}^T \boldsymbol{G} + \text{error}$$

Lattice-Based Private Constrained PRFs

[BTVW17]

Approach: encrypt the function f using an FHE scheme, and homomorphically evaluate U_x

$$\hat{f} \coloneqq \mathrm{Encrypt}(\mathrm{pk}, f) \qquad \left|\hat{f}\right| = L$$

$$\widehat{U}_x(\hat{f}) \coloneqq \mathrm{FHE.Eval}(\mathrm{pk}, U_x, \hat{f})$$

Homomorphic evaluation of U_x on f

Define $\widehat{U}_x(\hat{f})$ to output the GSW ciphertext (matrix-valued) obtained from homomorphic evaluation

Constrained key for f:

$$\boldsymbol{s}^T[\boldsymbol{A}_1 - \hat{f}_1 \cdot \boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - \hat{f}_L \cdot \boldsymbol{G}] + \boldsymbol{e}^T$$

Constrained evaluation at x:

$$\boldsymbol{s}^T[\boldsymbol{A}_1 - \hat{f}_1 \cdot \boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - \hat{f}_L \cdot \boldsymbol{G}]\boldsymbol{H}_{\widehat{U}_x, \hat{f}} + \boldsymbol{e}^T \boldsymbol{H}_{\widehat{U}_x, \hat{f}}$$

$$\approx \boldsymbol{s}^T\left(\boldsymbol{A}_{\widehat{U}_x} - \widehat{U}_x(\hat{f})\right)$$

$$\approx \boldsymbol{s}^T\left(\boldsymbol{A}_{\widehat{U}_x} - f(x) \cdot \boldsymbol{G}\right)$$

Some tweaks needed to argue security
(see [BTVW17] for full details)

Summary

Input-independent evaluation:

$$\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell, f \mapsto \boldsymbol{A}_f$$

PRF evaluation

Input-dependent evaluation:

$$[\boldsymbol{A}_1 - x_1\boldsymbol{G} \mid \cdots \mid \boldsymbol{A}_\ell - x_\ell\boldsymbol{G}]\boldsymbol{H}_{f,x} = \boldsymbol{A}_f - f(x)\boldsymbol{G}$$

constrained evaluation

Constraint privacy:

- Encrypt constraint using GSW FHE scheme
- LWE secret reused for PRF secret key and FHE secret key

Thank you!