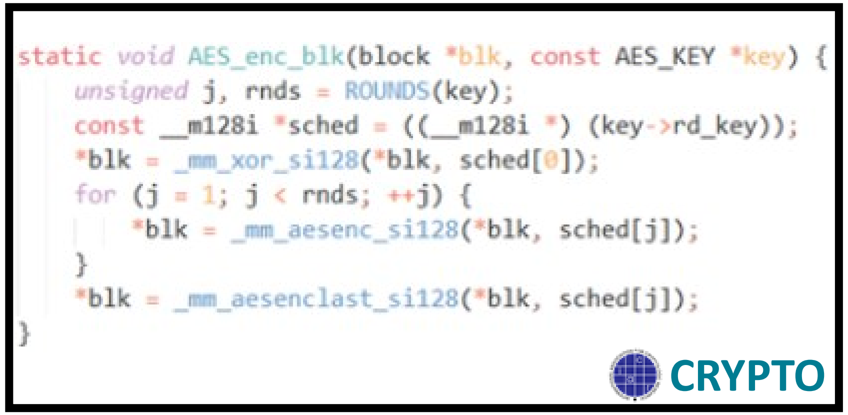
Beyond Software Watermarking:
Traitor-Tracing for Pseudorandom Functions

Rishab Goyal, Sam Kim, Brent Waters, and David J. Wu

Software Watermarking

[NSS99, BGIRSVY01, HMW07, CHNVW16]



```
static void AES_enc_blk(block *blk, const AES_KEY *key) {
    unsigned j, rnds = ROUNDS(key);
    const __m128i *sched = ((__m128i *) (key->rd_key));
    *blk = _mm_xor_si128(*blk, sched[0]);
    for (j = 1; j < rnds; ++j) {
        *blk = _mm_aesenc_si128(*blk, sched[j]);
    }
    *blk = _mm_aesenclast_si128(*blk, sched[j]);
}
```


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]

```
static void AES_enc_blk(block *blk, const AES_KEY *key) {
    unsigned j, rnds = ROUNDS(key);
    const __m128i *sched = ((__m128i *) (key->rd_key));
    *blk = _mm_xor_si128(*blk, sched[0]);
    for (j = 1; j < rnds; ++j) {
        *blk = _mm_aesenc_si128(*blk, sched[j]);
    }
    *blk = _mm_aesenclast_si128(*blk, sched[j]);
}
```

CRYPTO

Embed a “mark” within a program

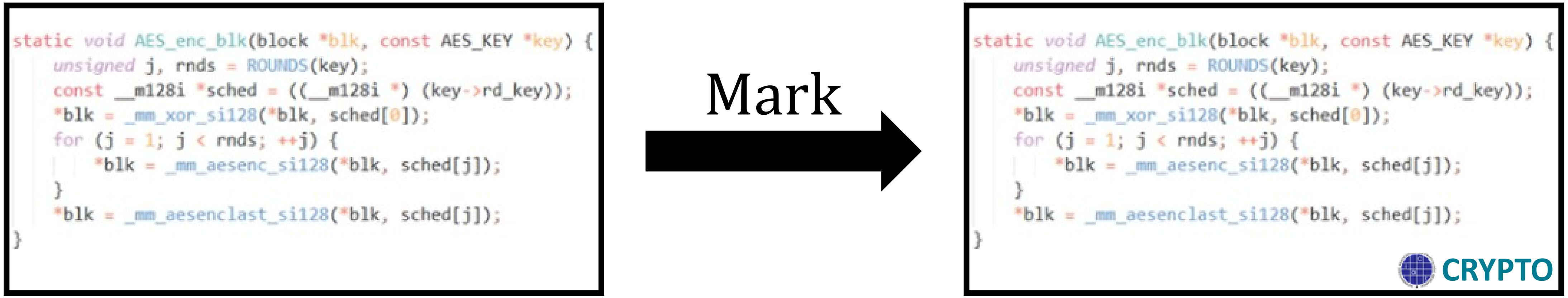
If mark is removed, then program is destroyed

Two main algorithms:

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

Software Watermarking

[NSS99, BGIRSVY01, HMW07, CHNVW16]



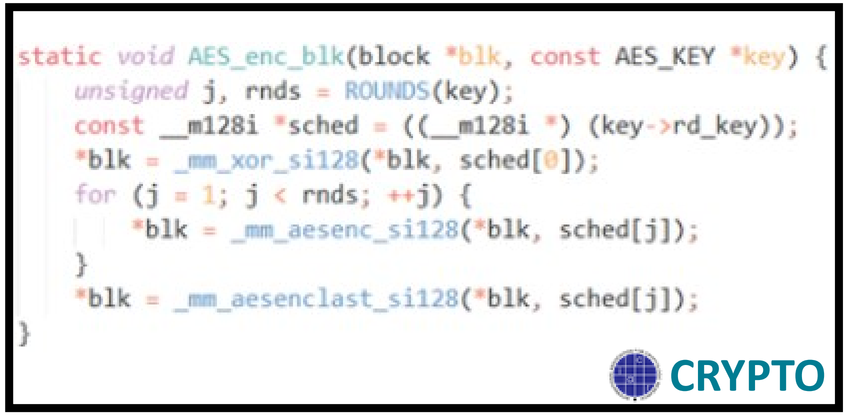
Functionality-preserving: On input a circuit C (and mark m), the Mark algorithm outputs a circuit C' where

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

on almost all inputs x

Software Watermarking

[NSS99, BGIRSVY01, HMW07, CHNVW16]



```
static void AES_enc_blk(block *blk, const AES_KEY *key) {
    unsigned j, rnds = ROUNDS(key);
    const __m128i *sched = ((__m128i *) (key->rd_key));
    *blk = _mm_xor_si128(*blk, sched[0]);
    for (j = 1; j < rnds; ++j) {
        *blk = _mm_aesenc_si128(*blk, sched[j]);
    }
    *blk = _mm_aesenclast_si128(*blk, sched[j]);
}
```


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

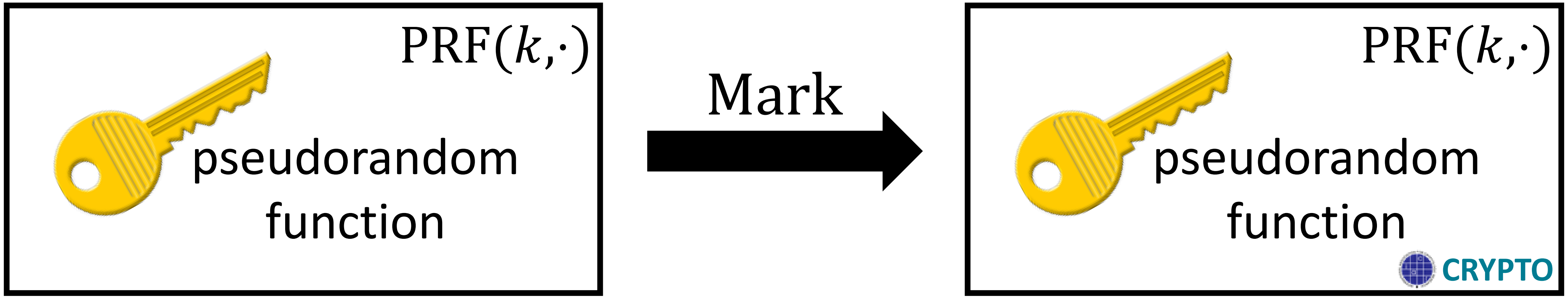
- $C^*(x) = C'(x)$ on almost all inputs x
- The circuit C^* does not preserve the mark: $\text{Extract}(C^*) \neq m$

Software Watermarking

[NSS99, BGIRSVY01, HMW07, CHNVW16]

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

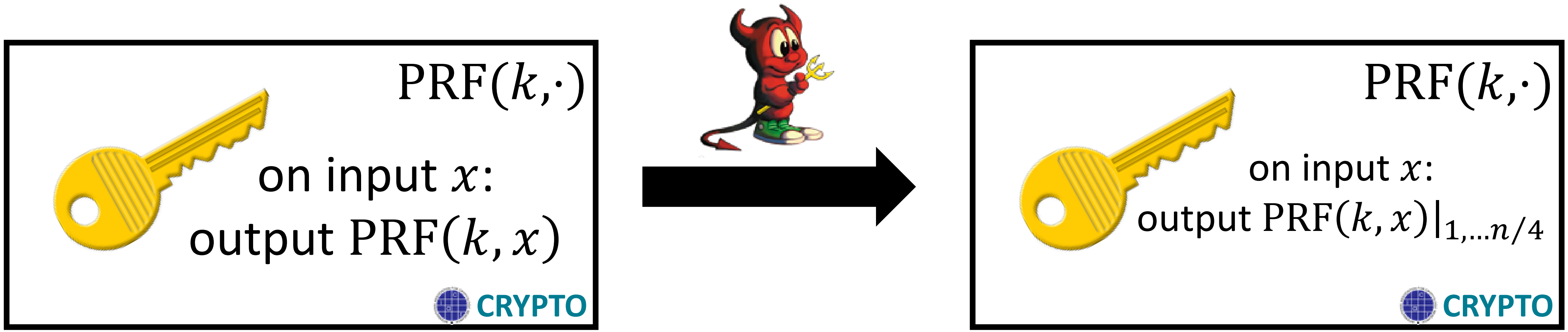
Watermarking Cryptographic Programs



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

Suffices for watermarking other symmetric primitives:
(e.g., MAC signing key, symmetric decryption key)

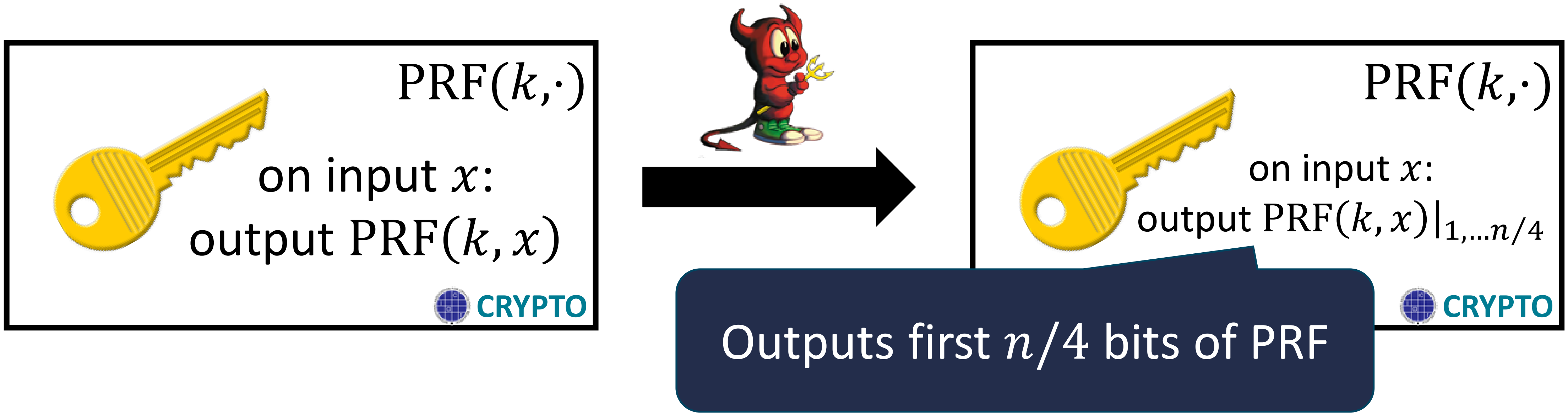
A Closer Look at Watermarking Security



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

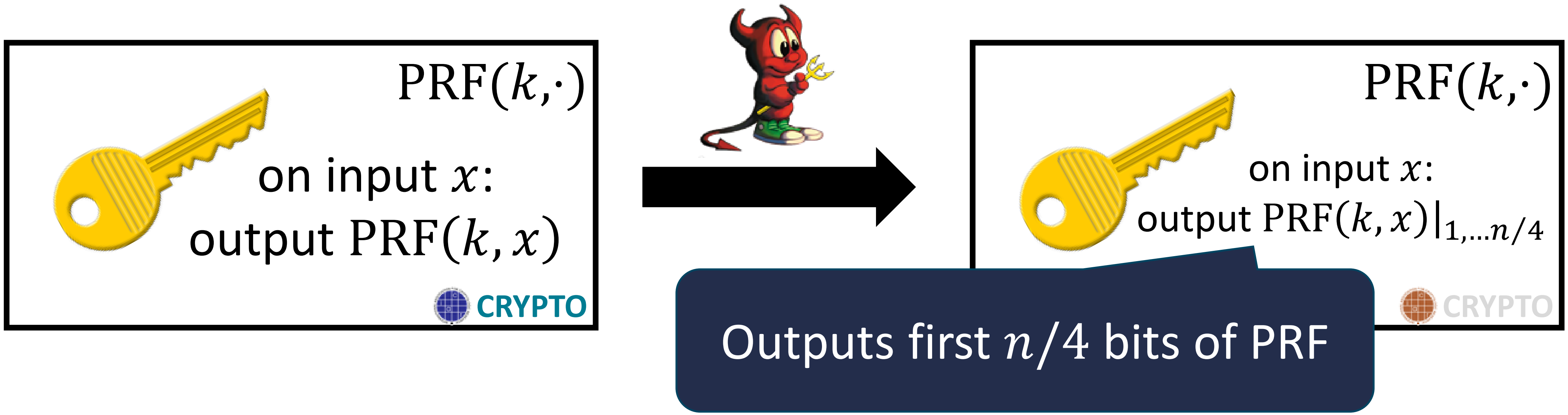


Unremovability: Given a program C' with mark m, no efficient adversary can construct a circuit C^* where

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

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: $\text{Extract}(C^*) \neq m$

Adversary's circuit does not preserve functionality

No guarantees on whether the mark is preserved or not!

A Closer Look at Watermarking Security

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

on input x:
output $\mathrm{PRF}(k,x)$

CRYPTO

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

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

CRYPTO

Outputs first $n/4$ bits of PRF

Existing watermarking constructions are unable to recover the watermark from this type of program

ogram C' with mark m, no efficient

rcuit C^* where

ost all inputs x

t 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

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

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

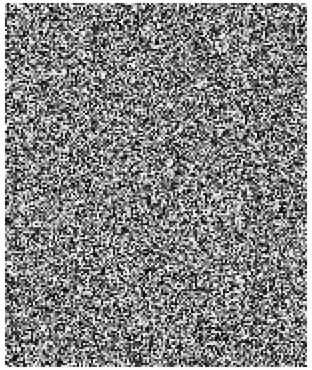
CRYPTO

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

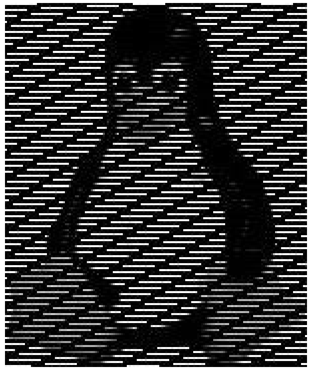
Is this a problem?

For building blocks like PRFs, we do not necessarily need to recover <u>exact</u> 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" to break the application, but may <u>not</u> preserve watermark

A Closer Look at Watermarking Security

Typically in cryptography:

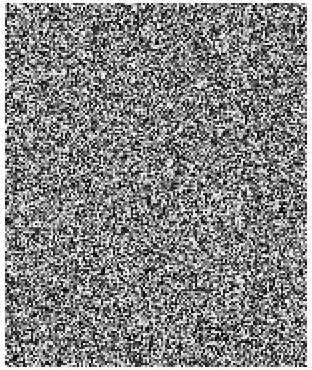
adversary's goals are separate from honest parties' goals

Encryption:

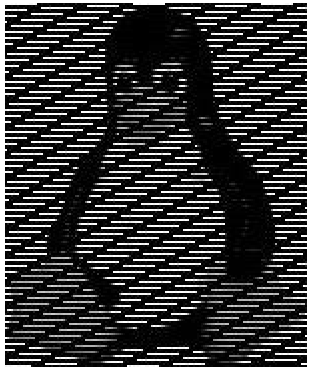
- **Correctness:** recover message from ciphertext
- **Security:** learn *anything* about message from ciphertext

For building blocks like PRFs, we do not necessarily need to recover <u>exact</u> 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" to break the application, but may <u>not</u> preserve watermark

A Closer Look at Watermarking Security

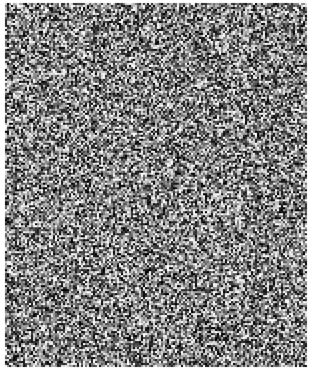
$\mathrm{PRF}(k,\cdot)$

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

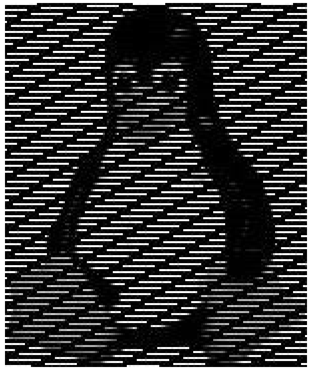
Watermarking cryptographic programs:

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

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



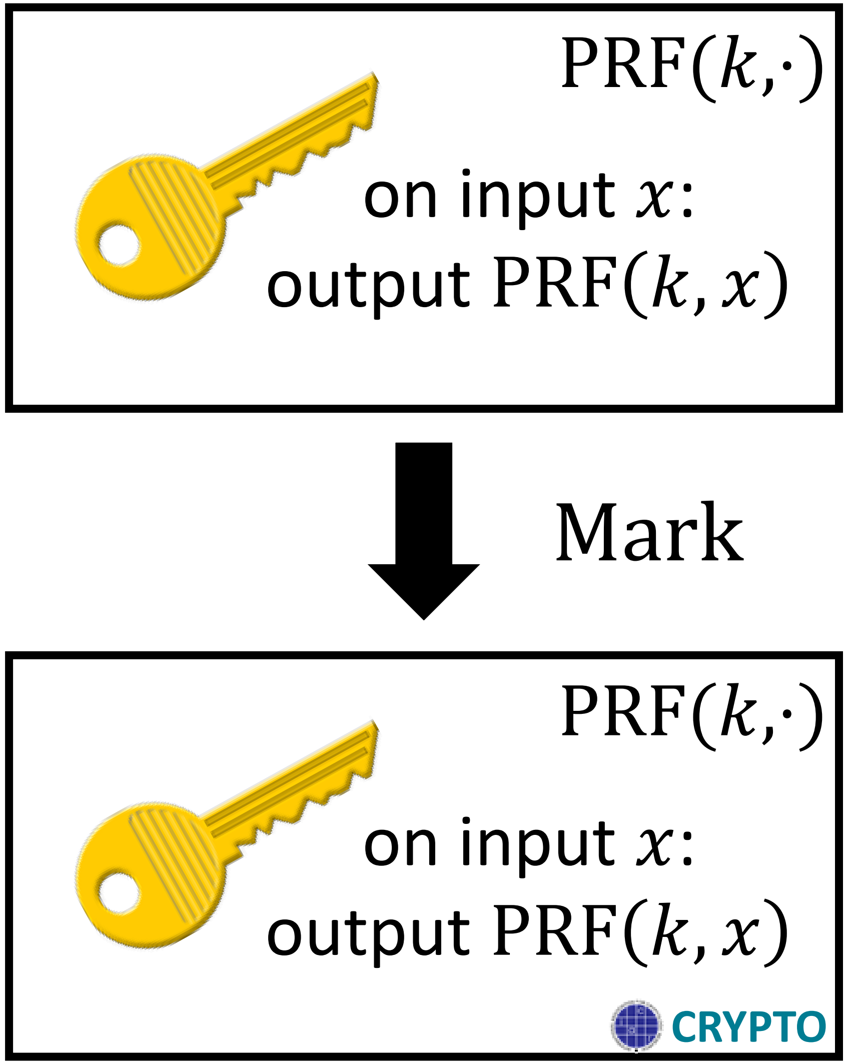
Encrypted image
(PRF in counter mode)



Partial decryption
(using program on left)

Adversary's program is "good enough" to break the application, but may <u>not</u> preserve watermark

Traceable PRFs



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

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

Traceable PRFs

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

on input x:
output $\text{PRF}(k,x)$

Mark

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

on input x:
output $\text{PRF}(k,x)$

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

Traceable PRF: analog for PRFs

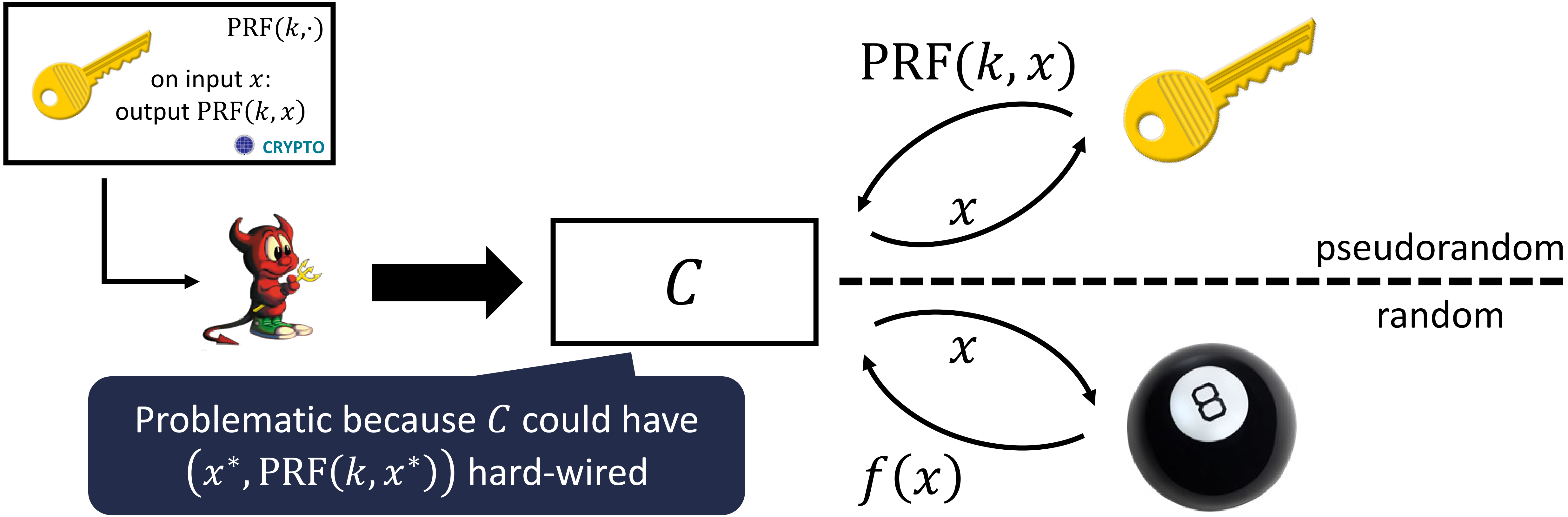
Marking security (informal):
if program C can *distinguish* $\text{PRF}(k,\cdot)$ from random, then mark should be preserved

Traceable PRFs

Marking security (informal):

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

from random, then mark should be preserved

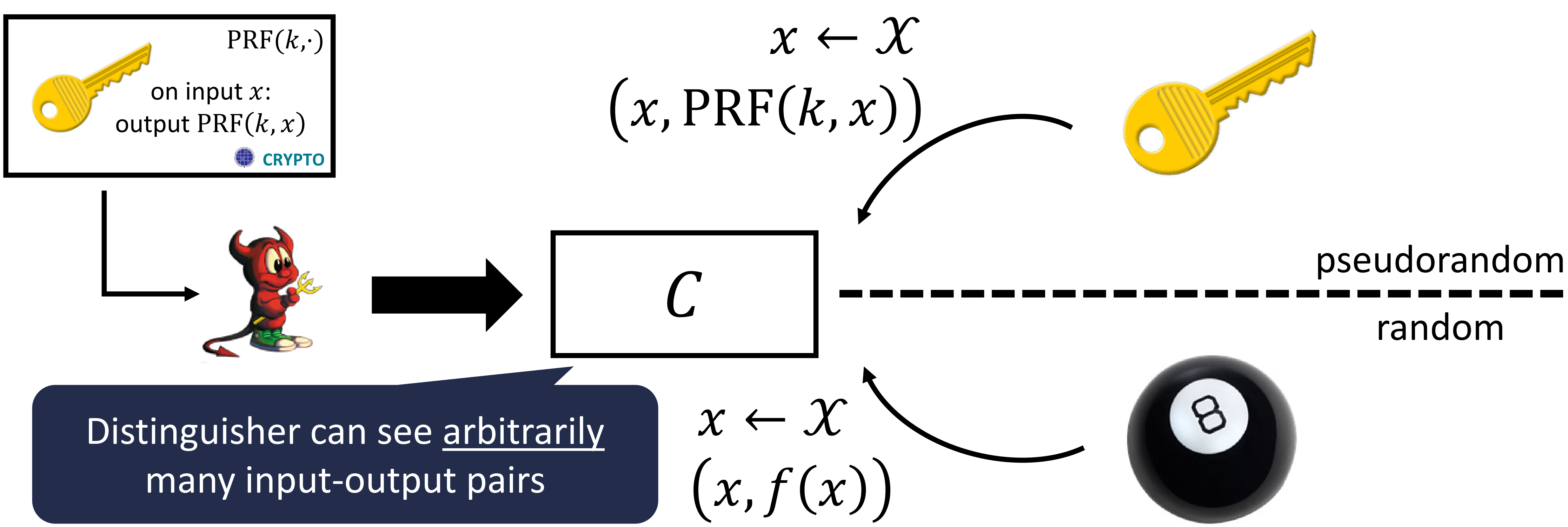


Traceable PRFs

Marking security (informal):

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

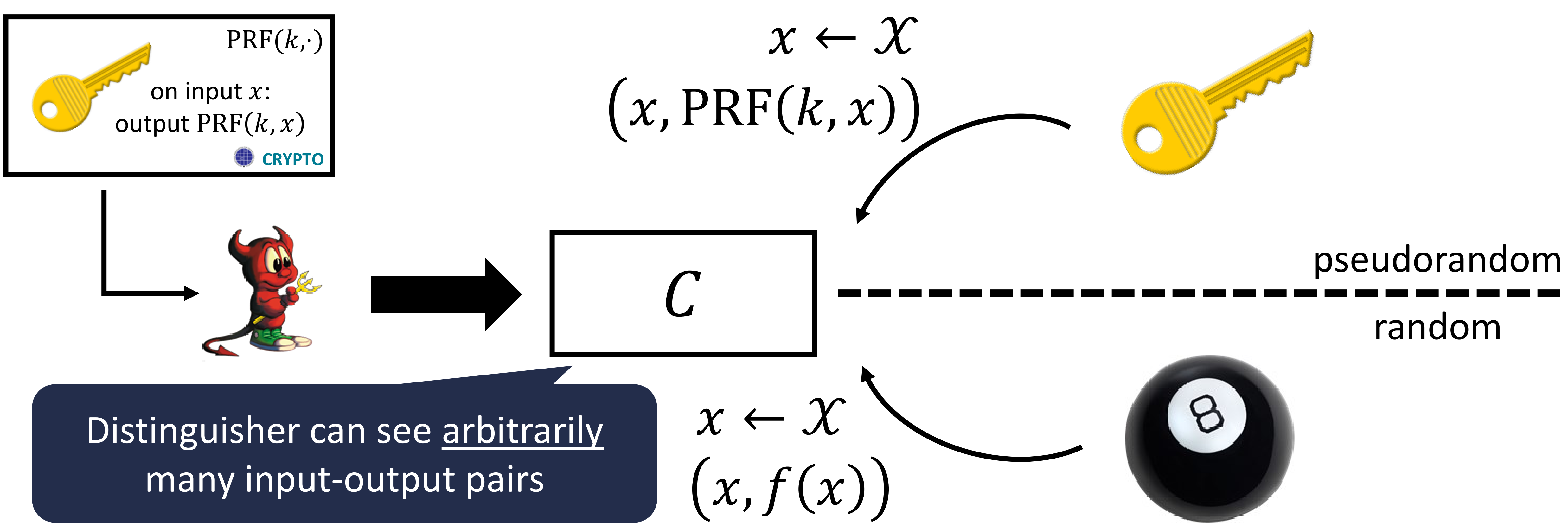
on randomly sampled inputs, then mark should be preserved



Traceable PRFs

Marking security (informal):

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



Traceable PRFs

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

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

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

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

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

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

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

tracing algorithm given <u>oracle</u> access to weak PRF distinguisher

Traceable PRFs

Correctness: marked and unmarked keys agree almost everywhere

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

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

Tracing Security:

$\xrightarrow{\quad \mathrm{id} \quad}$

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

$\xrightarrow{\quad D \quad}$

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

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

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

Instantiate PRF with a traceable PRF

Not the case if we start with watermarkable PRF!

Tracing Security:

id →

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

D →

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]

PRF key

Constrained key

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]

x input point

index t

Can evaluate inputs with indices $t \leq \mathrm{id}$

Linear constraint family:

- Input points are associated with a (secret) index t between 0 and 2^ℓ
- Constrained key associated with $\mathrm{id} \in [0, 2^\ell - 1]$

Constructing Traceable PRFs

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

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

x input point

index t

Can evaluate inputs with indices $t \leq \mathrm{id}$

can evaluate | cannot evaluate

0 | id | 2^{ℓ}

privacy: index is hidden

index t (for PRF domain element)

Constructing Traceable PRFs

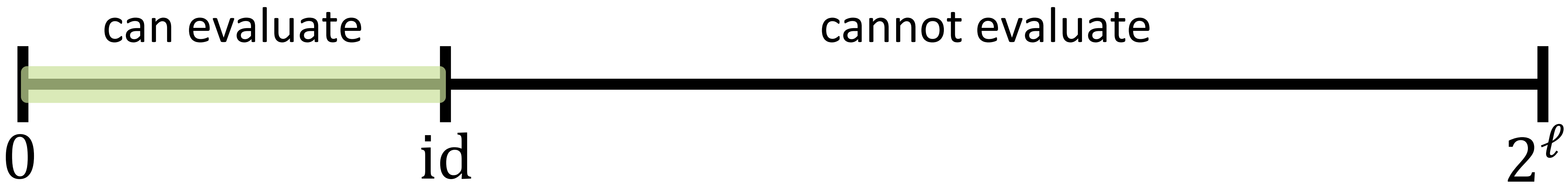
Rely on intermediate notion: **private linear constrained PRF**

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

x input point

index t

Can evaluate inputs with indices $t \leq \mathrm{id}$



There exists a sampling algorithm to sample inputs with a specified index

Constructing Traceable PRFs

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

Normal hiding

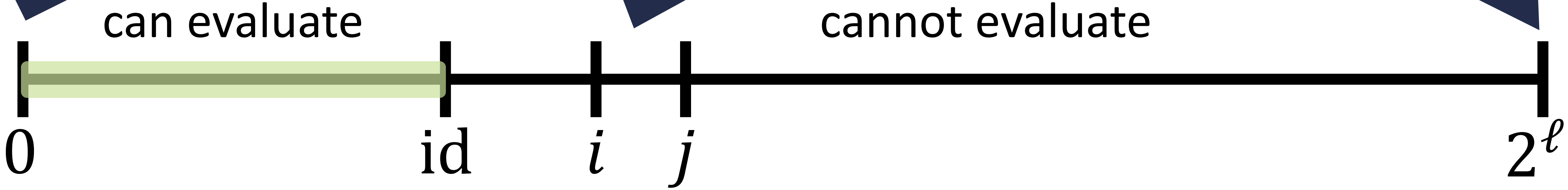
Inputs with index 0 are indistinguishable from random inputs

Identity hiding

PRF inputs with indices i, j are indistinguishable without key for $i \leq \mathrm{id} < j$

Pseudorandomness

PRF outputs on inputs with index 2^ℓ are pseudorandom



There exists a sampling algorithm to sample inputs with a specified index

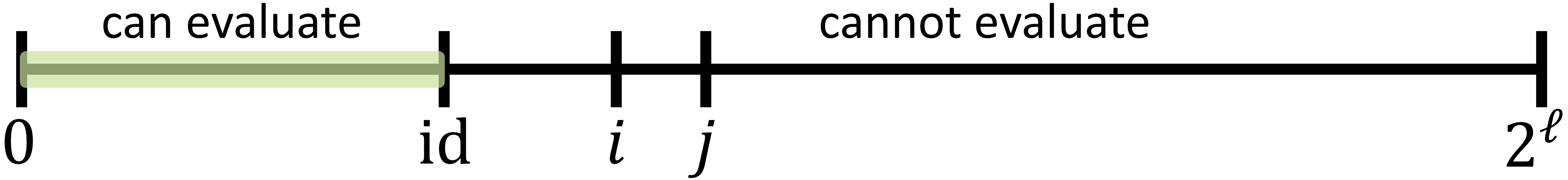
Constructing Traceable PRFs

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

Can trace using algorithm for oracle jump-finding problem [NWZ16]



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 (for circuit constraints)

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

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

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

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

Constructing Private Linear Constrained PRF

Starting point: standard constrained PRF (for circuit constraints)

Solution: Encrypt indices

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

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

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 (for circuit constraints)

Problem: constrained key might leak k which leaks indices

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

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 Private Linear Constrained PRF

Starting point: standard constrained PRF (for circuit constraints)

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

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

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

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]

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

LWE

↓

single-key
private constrained PRF

+

symmetric encryption

→

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

→

single-key
traceable PRF
(with secret tracing)

Traceable PRF Summary

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

on input x:
output $\mathrm{PRF}(k,x)$

Mark $\longrightarrow$

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

on input x:
output $\mathrm{PRF}(k,x)$

CRYPTO

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”

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