Watermarking PRFs from Lattices via Extractable PRFs

Sam Kim and David J. Wu

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]

```
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

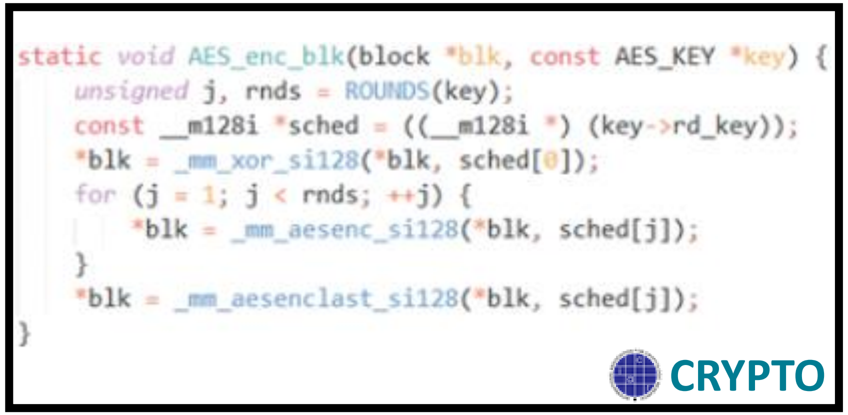
If mark is removed, then program is destroyed

Two main algorithms (simplified):

- $\mathrm{Mark}(C) \to C'$: Takes a circuit C and outputs a marked circuit C'
- $\mathrm{Verify}(C') \to \{0,1\}$: Tests whether a circuit C' is marked or not

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]



Embe

Notion extend to setting where watermark can be any string

If mark is removed, then program is destroyed

Two mai algo s (simplified):

- $\mathrm{Mark}(C) \to C'$: Takes a circuit C and outputs a marked circuit C'
- $\mathrm{Verify}(C') \to \{0,1\}$: Tests whether a circuit C' is marked or not

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]

```
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]);
}
```

Mark


```
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

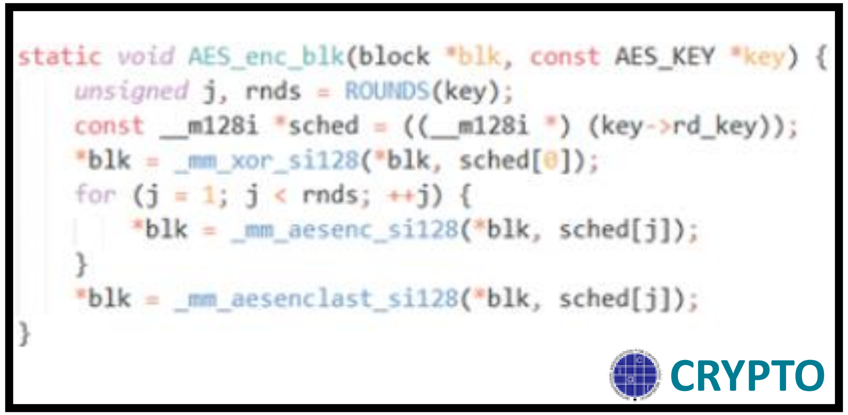
Functionality-preserving: On input a circuit C, the Mark algorithm outputs a circuit C' where

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

on almost all inputs x

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]

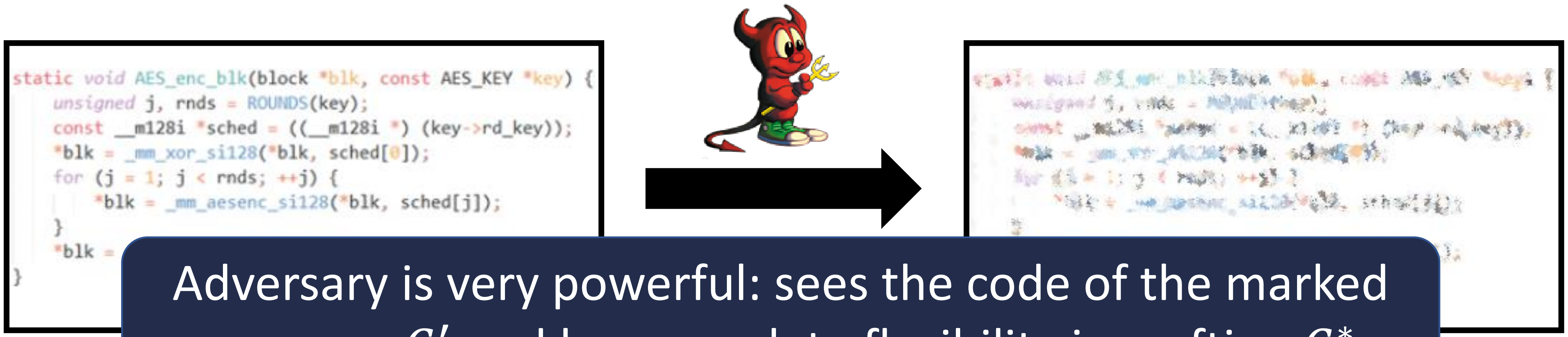


Unremovability: Given a marked program C', no efficient adversary can construct a circuit C^* where

- $C^*(x) = C'(x)$ on almost all inputs x
- The circuit C^* is unmarked: $\text{Verify}(C^*) = 0$

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]



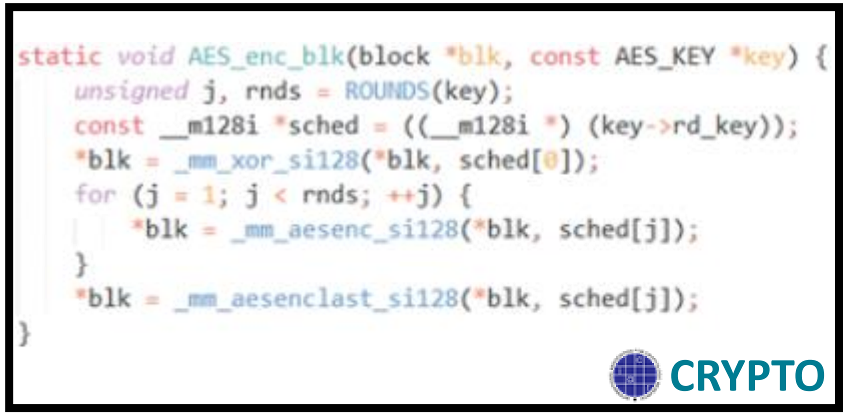
Adversary is very powerful: sees the code of the marked program C' and has complete flexibility in crafting C^*

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

- $C^*(x) = C'(x)$ on almost all inputs x
- The circuit C^* is unmarked: $\text{Verify}(C^*) = 0$

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]

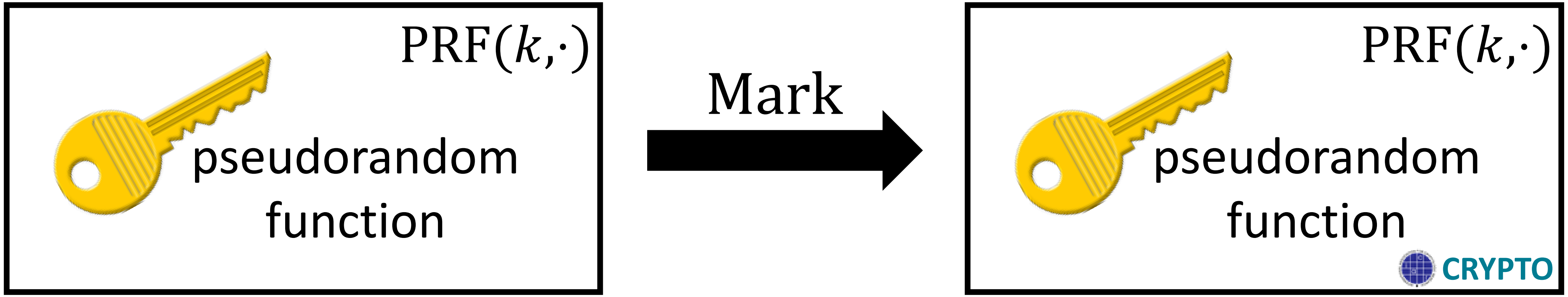


Learning the original (unmarked) function gives a way to remove the watermark

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

Watermarking Cryptographic Programs

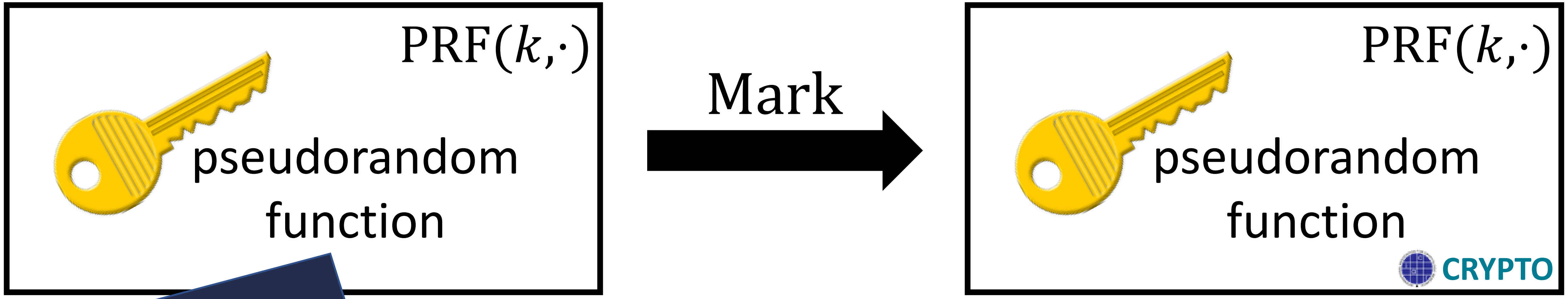
[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]



- Focus of this work: watermarking PRFs [CHNVW16, BLW17, KW17, QWZ18]

Watermarking Cryptographic Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]

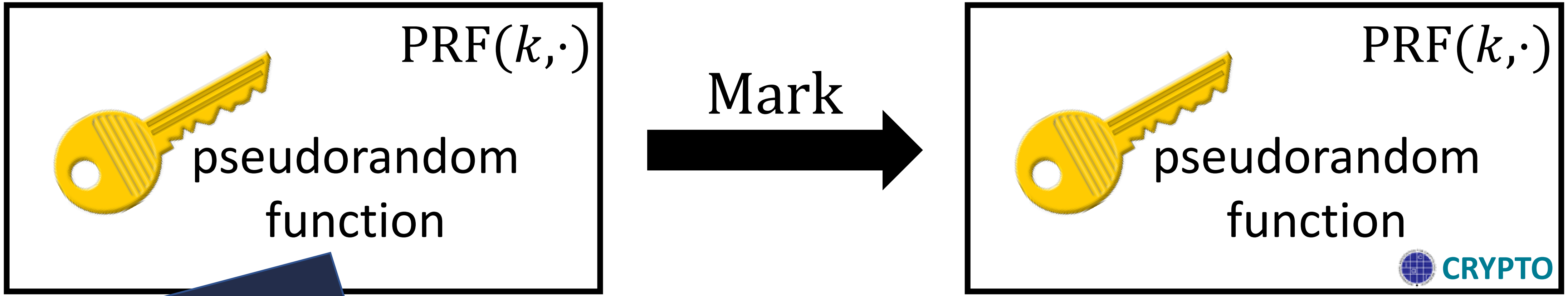


A function whose input-output behavior is unpredictable (looks like a random function) – e.g., AES

ng PRFs [CHNVW16, BLW17, KW17, QWZ18]

Watermarking Cryptographic Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17, KW17, QWZ18, GKMWW19]



Program has PRF key k hard-wired inside it and on input x, outputs $\mathrm{PRF}(k, x)$

ng PRFs [CHNVW16, BLW17, KW17, QWZ18]

Watermarkable PRFs

[CHNVW16]: Watermark PRFs from **iO + OWFs**

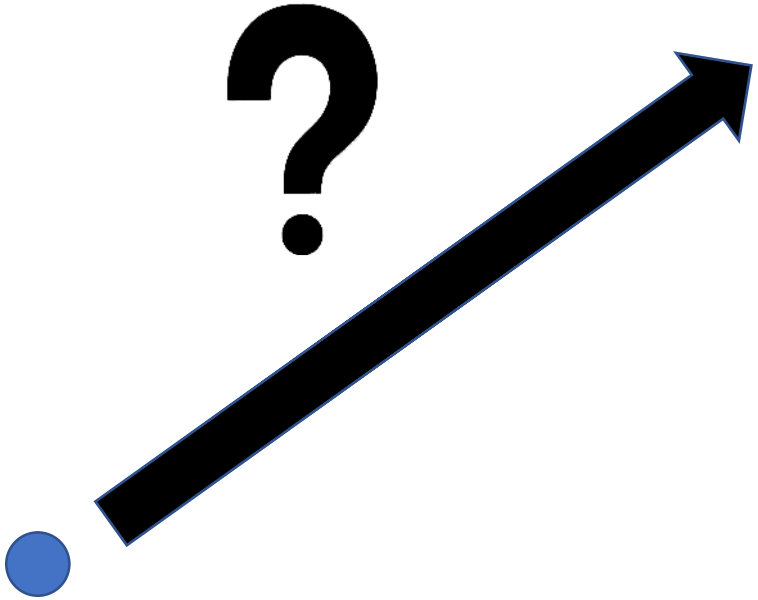
Publicly verifiable

Can we watermark PRFs from standard assumptions?

[KW17]: Watermark PRFs from standard assumptions (LWE)

Secretly verifiable

Watermarkable PRFs



Publicly Verifiable Watermarking [CHNVW16]

Secretly Verifiable Watermarking from LWE [KW17]

A Naïve Attempt at Public Verifiability

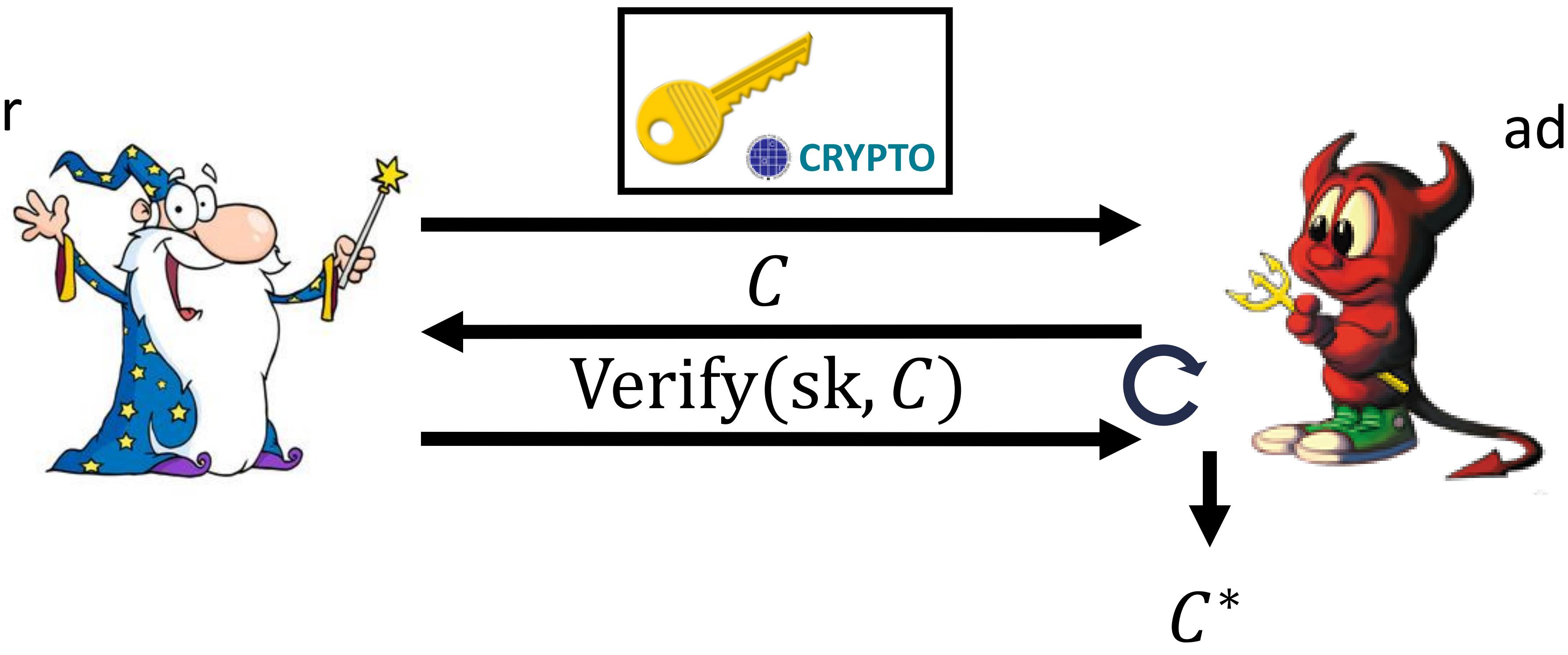
Just make the verification key public!

Problem: Knowledge of the verification key allows adversary to trivially remove watermark

In fact: Even <u>oracle access</u> to the verification key is sufficient to break unremovability (“verifier rejection” problem)

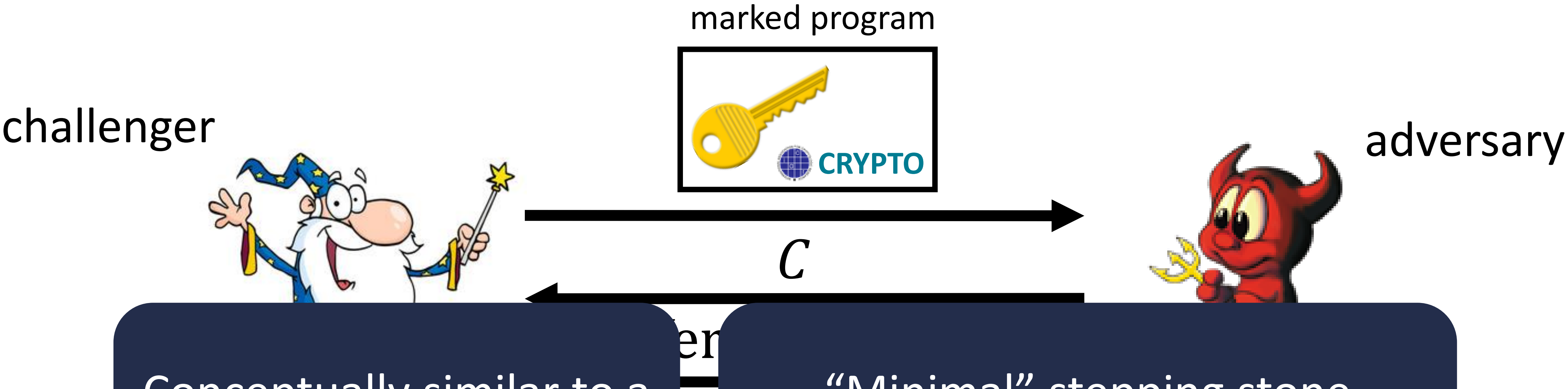
Secretly Verifiable Watermarking from LWE [KW17]

Between Public and Secret Verification



Intermediate notion: Secret verification, but <u>security</u> in the presence of a verification oracle

Between Public and Secret Verification



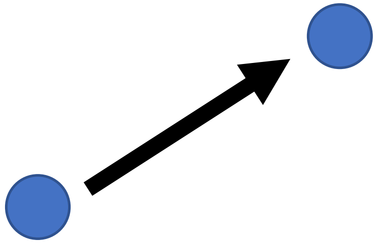
Conceptually similar to a CCA security notion

"Minimal" stepping stone towards public verifiability

Intermediate notion: Secret verification, but <u>security</u> in the presence of a verification oracle

Watermarkable PRFs

Secretly Verifiable Watermarking from CCA [QWZ18]



Secretly Verifiable Watermarking from LWE [KW17]

Publicly Verifiable Watermarking [CHNVW16]

Watermarkable PRFs

Secretly Verifiable Watermarking from CCA [QWZ18]

Secretly Verifiable Watermarking from LWE [KW17]

Good: Achieves security in the presence of a verification oracle

Limitation: Knowledge of the verification key breaks PRF security (even *unmarked* keys)

Security Against the Authority

[QWZ18]

sk

watermarking authority

x

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

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

pseudorandom function

After seeing single query (on <u>any</u> x), authority can distinguish output of PRF from output of random function

Security Against the Authority

[QWZ18]

Implication: Knowledge of the
verification key completely breaks PRF security

(notion still seems far publicly-verifiable setting)

After seeing single query (on <u>any</u> x), authority can
distinguish output of PRF from output of random function

Don’t We Have to Trust the Authority Anyways?

sk

watermarking authority

x

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

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

pseudorandom function

Not necessarily: marking algorithm can be implemented using a two-party computation, so authority never needs to see *any* PRF keys in the clear

Don’t We Have to Trust the Authority Anyways?

sk

watermarking authority

x $\longrightarrow$

$\mathrm{PRF}(k, x)$ $\longleftarrow$

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

pseudorandom function

This work: New watermarkable PRF that provides security even against the watermarking authority

Our Results

New **secretly verifiable** watermarking for PRF from LWE

- Unremovability holds in the presence of the verification oracle

- **weak pseudorandomness** *even against authority*
  (T-restricted pseudorandomness)

- As secure as any other PRF family from LWE
  - Relies on worst-case lattice problems with **nearly-polynomial** ($n^{\omega(1)}$) approximation factors

Our Results

New **secretly verifiable** wate

- Unremovability holds in

- **weak pseudorandomne**

Previous constructions (with message-embedding) required <u>private</u> constrained PRFs (which requires quasi-polynomial or sub-exponential approximation factors)

- As secure as any other PRF family from LWE
  - Relies on worst-case lattice problems with **nearly-polynomial** ($n^{\omega(1)}$) approximation factors

- New abstraction: **extractable PRF**

Starting Point: Puncturable PRF

[BW13, BGI14, KPTZ13]

Punctured key k_{x^*} can be used to evaluate PRF on all points $x \neq x^*$
(value at x^* is pseudorandom even given k_{x^*})

Private puncturing: punctured key k_{x^*} also hides x^*

Programmability: program $F(k_{x^*}, x^*) \coloneqq y^*$

From Puncturing to Watermarking

[BLW17, KW17]

PRF key k

punctured key k_{x^*}

Marking algorithm:

1. Derive a special point (x^*, y^*) from input/output behavior of PRF
2. Define a marked circuit to be $F(k_{x^*}, \cdot)$

Verification algorithm:

1. Test if $C(x^*) = y^*$

Security: Punctured point x^* is hidden

Intuition: Binary Search Attack

programmed key k_{x^*}

$\vdots$

Verify(vk,·)

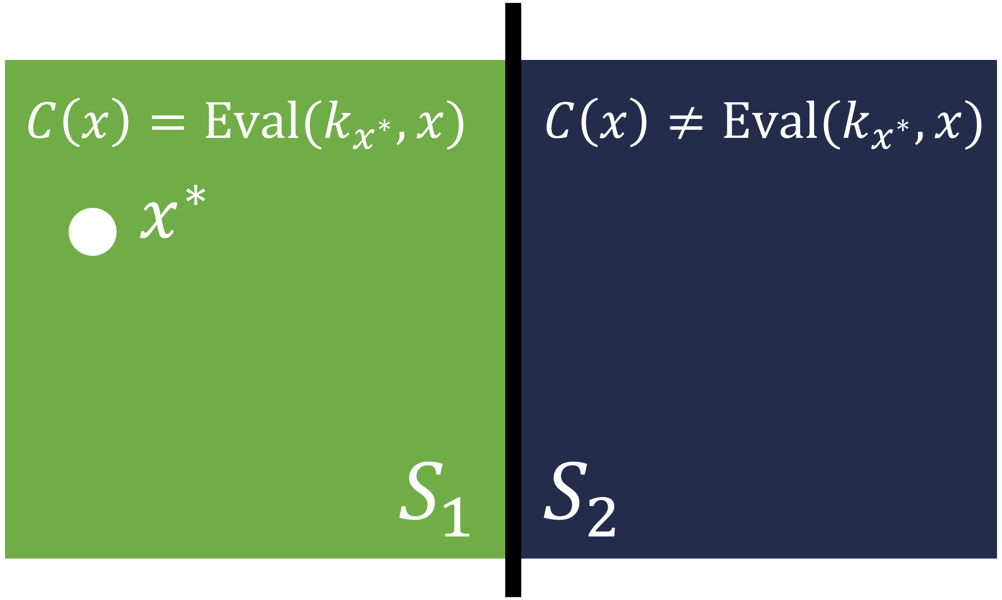
PRF Domain:

x^*

Intuition: Binary Search Attack

programmed key k_{x^*}

PRF Domain:



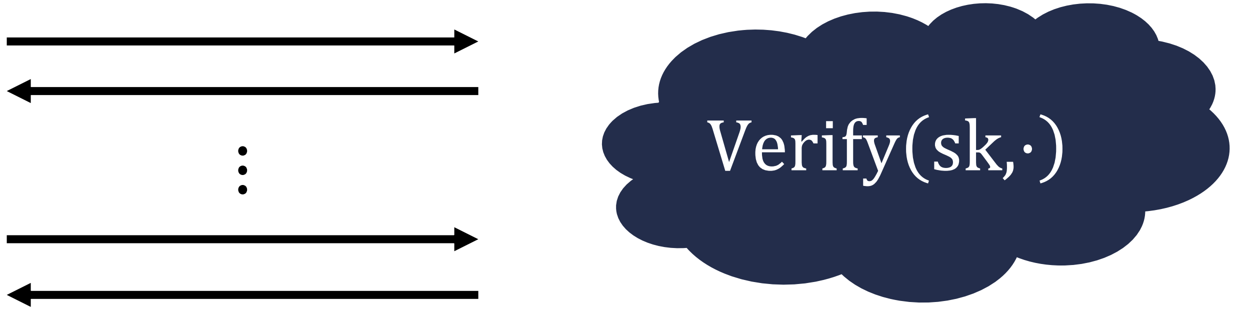
Intuitively:

if $\mathrm{Verify}(\mathrm{sk}, C) = 1$, then $x^* \notin S_2$

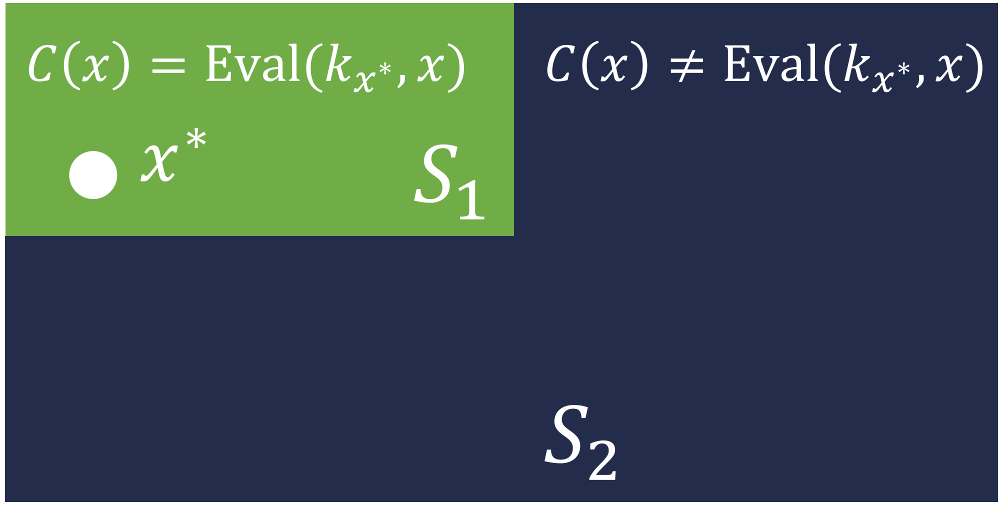
if $\mathrm{Verify}(\mathrm{sk}, C) = 0$, then $x^* \in S_2$

Intuition: Binary Search Attack

programmed key k_{x^*}



PRF Domain:



Intuitively:

if $\mathrm{Verify}(\mathrm{sk}, C) = 1$, then $x^* \notin S_2$

if $\mathrm{Verify}(\mathrm{sk}, C) = 0$, then $x^* \in S_2$

Eventually, adversary recovers special point x^*

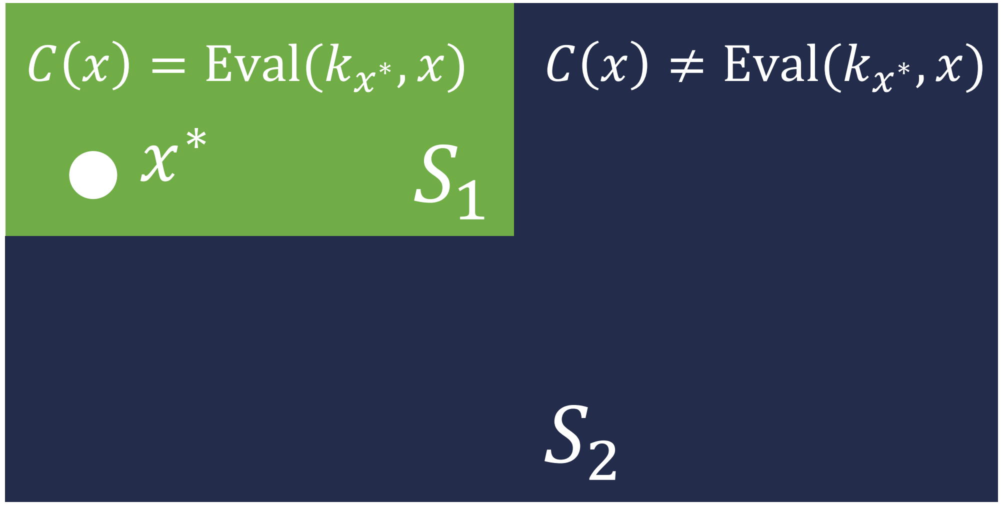
Intuition: Binary Search Attack

programmed key k_{x^*}

Very similar to a "verifier rejection" attack encountered in settings like designated-verifier proof systems, CCA-security, etc.

Solution: Make the set of "valid" circuits detectable (i.e., cannot change too many points and still preserve mark)

PRF Domain:



Intuitively:

if $\text{Verify}(\text{sk}, C) = 1$, then $x^* \notin S_2$

if $\text{Verify}(\text{sk}, C) = 0$, then $x^* \in S_2$

Eventually, adversary recovers special point x^*

Our Notion: Extractable PRF

Punctured key k_{x^*} can be used to evaluate PRF on all points $x \neq x^*$

Private puncturing: punctured key k_{x^*} also hides x^*

Programmability: program $F(k_{x^*}, x^*) \coloneqq y^*$

<u>**Extractability**</u>: point $F(k_{x^*}, z) \coloneqq \mathrm{Encode}(k)$ encode original PRF key k

Our Notion: Extractable PRF

PRF key k

punctured key k_{x^*}

Punctured key k_{x^*} can be used to evaluate PRF on all points $x \neq x^*$

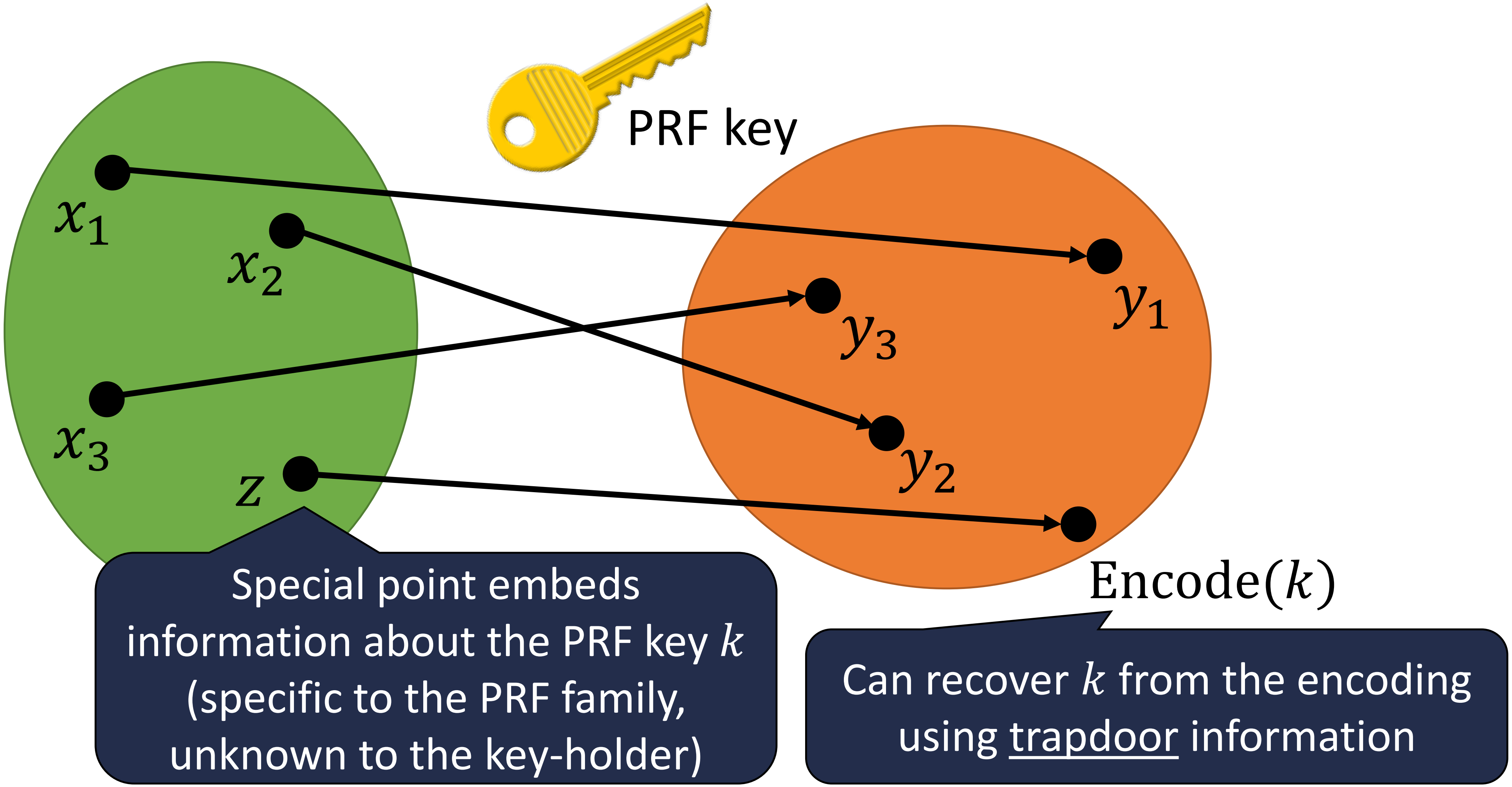
Private puncturing: puncture

Programmability: program F

Decode with trapdoor td
(part of watermarking sk)

<u>Extractability</u>: point $F(k_{x^*}, z) \coloneqq \text{Encode}(k)$ encode original PRF key k

Our Notion: Extractable PRF



Extraction to Watermarking

Marking algorithm:

1. Derive a special point (x^*, y^*) from input/output behavior of PRF
2. Define a marked circuit to be $F(k_{x^*}, \cdot)$

Verification algorithm:

1. Test if $C(x^*) = y^*$
2. Extract key k and test if $C(\cdot) \approx F(k, \cdot)$

   (output unmarked if key extraction fails)
3. Accept only if both conditions satisfied

Adversary can rule out only a small fraction of domain

In fact: extractability enables a simpler marking procedure

Extraction to Watermarking

Marking algorithm:

1. Derive a special point (x^*, y^*) from input/output behavior of PRF
2. Define a marked circuit to be $F(k_{x^*}, \cdot)$

Verification algorithm:

1. Test if $C(x^*) = y^*$
2. Extract key k
3. Accept only if

Instead of programming the value at x^*, <u>puncture</u> the PRF at x^*: circuit is <u>marked</u> if $C(x^*) \neq F(k, x^*)$ where k is the extracted key

In fact: extractability enables a <u>simpler</u> marking procedure

Extraction to Watermarking

Marking algorithm:

1. Puncture key at x^* to obtain a key k_{x^*}
2. Define a marked key to be $F(k_{x^*}, \cdot)$

To remove watermark, need to fix the value of the PRF at the punctured point (i.e., guess a pseudorandom value)

Verification algorithm:

1. Extract key k and test if $C(\cdot) \approx F(k, \cdot)$

   (output unmarked if key extraction fails)

2. Output marked if $C(x^*) \neq F(k, x^*)$ and unmarked otherwise

In fact: extractability enables a simpler marking procedure

Extraction to Watermarking

Marking algorithm:

1. Puncture key at x^* to obtain a key k_{x^*}
2. Define a marked key to be $F(k_{x^*}, \cdot)$

To remove watermark, need to fix the value of the PRF at the punctured point (i.e., guess a pseudorandom value)

Verification algorithm:

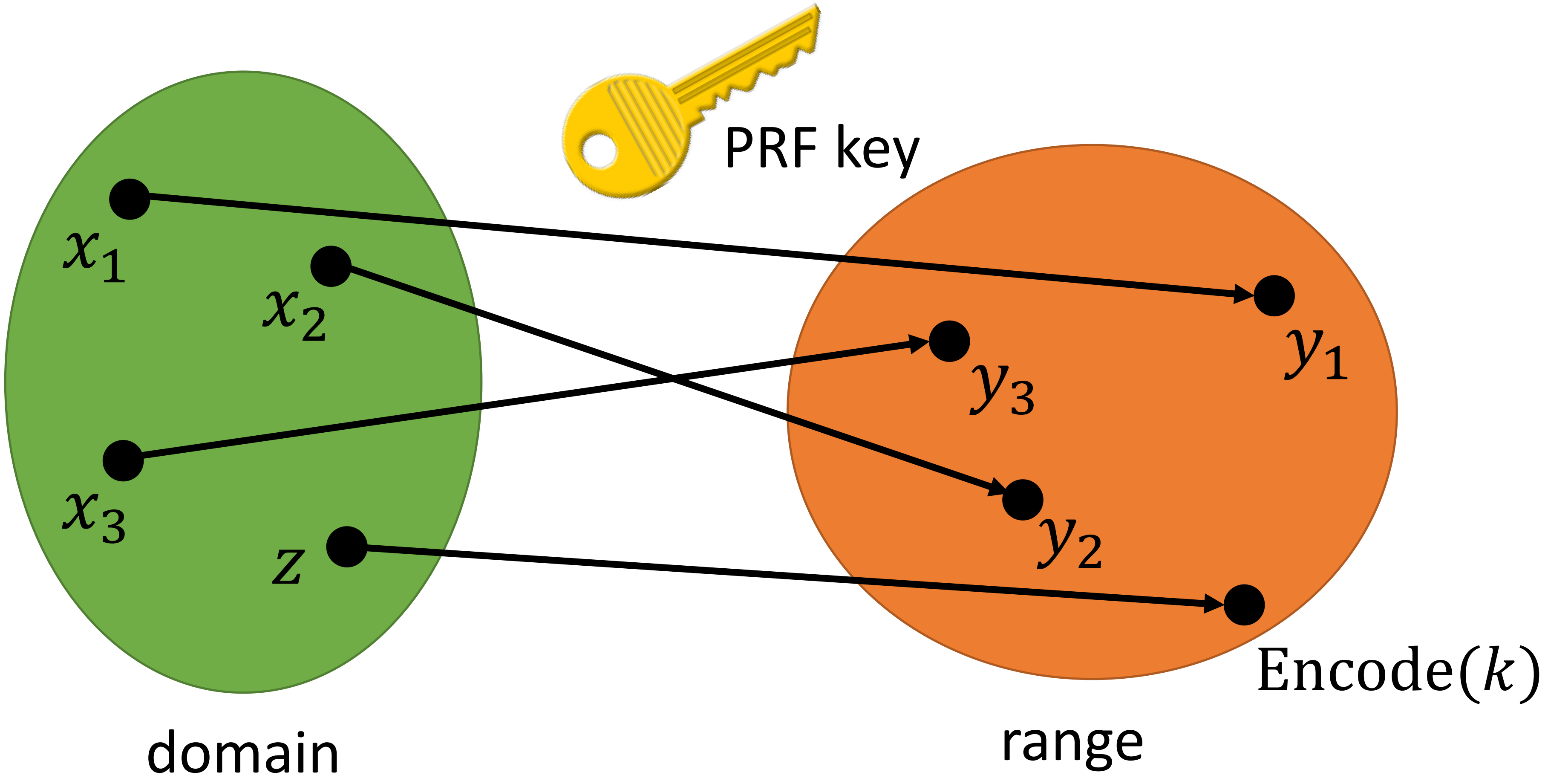
1. Extract key k and test if $C(\cdot) \approx F(k, \cdot)$

(output unmarked if key extraction fails)

2. Output marked if $C(x^*) \neq F(k, x^*)$ and unmarked otherwise

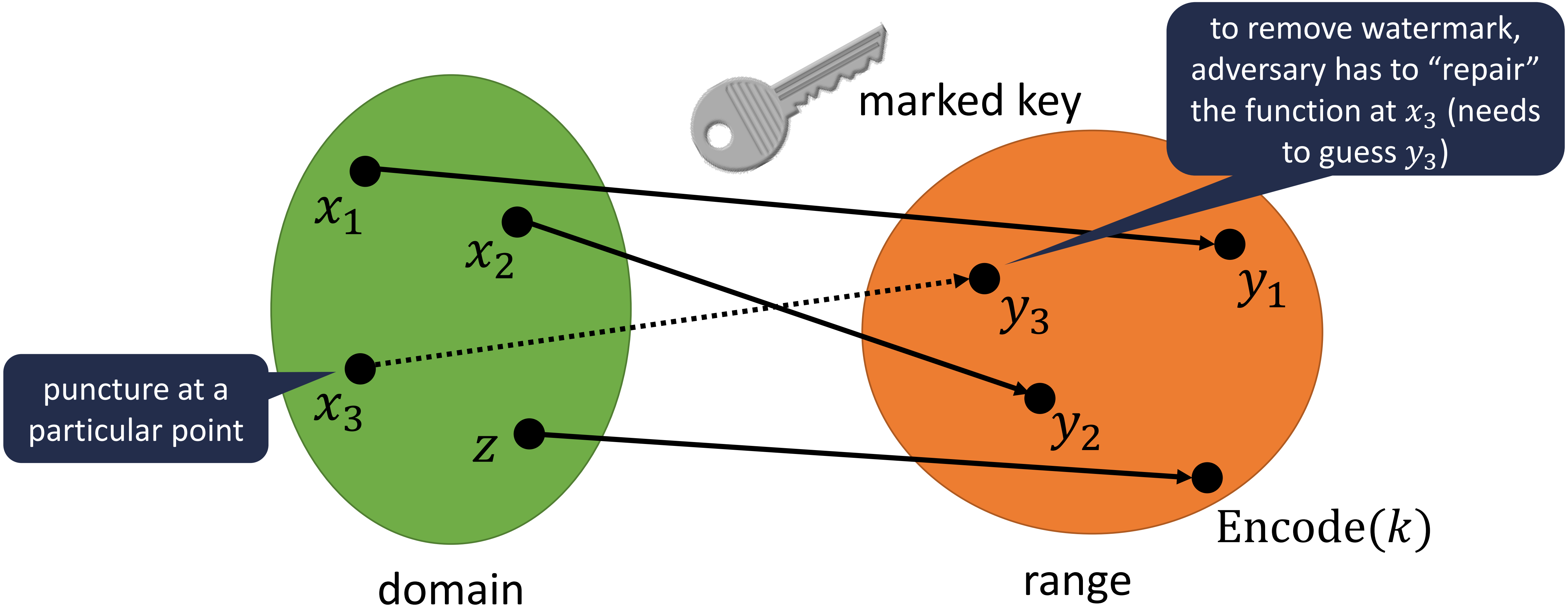
Advantage: no longer require private puncturing
(can base on weaker assumptions)

Extraction to Watermarking



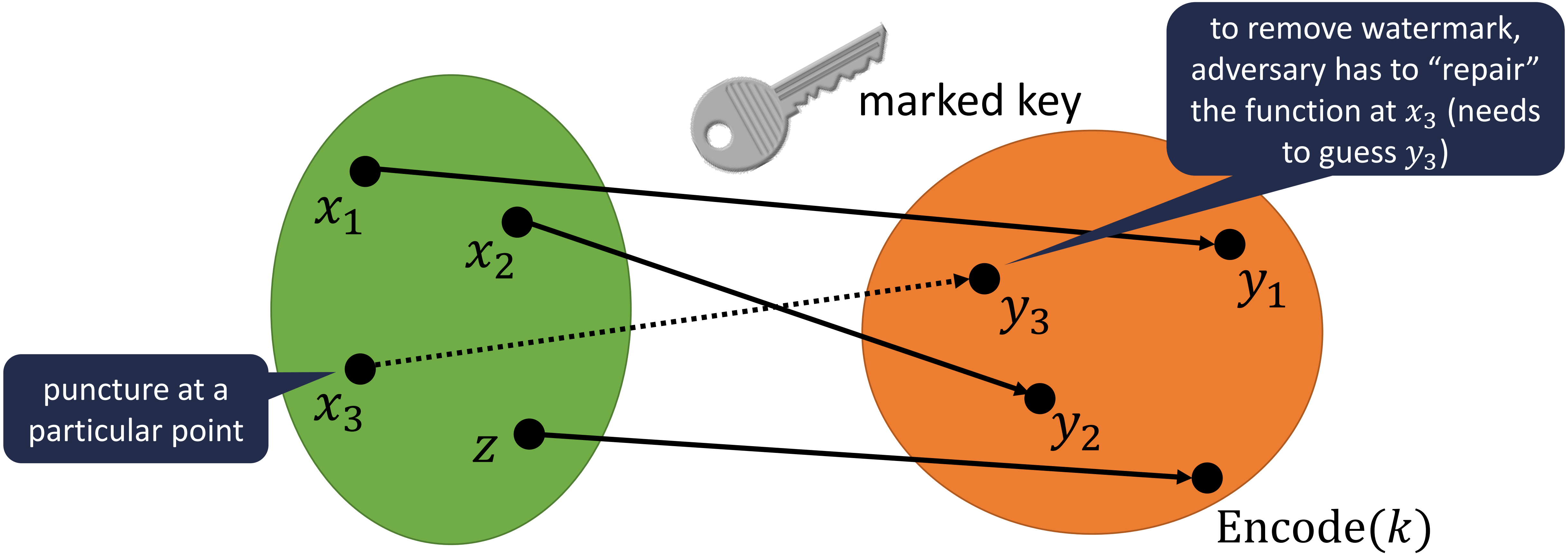
Real PRF key

Extraction to Watermarking



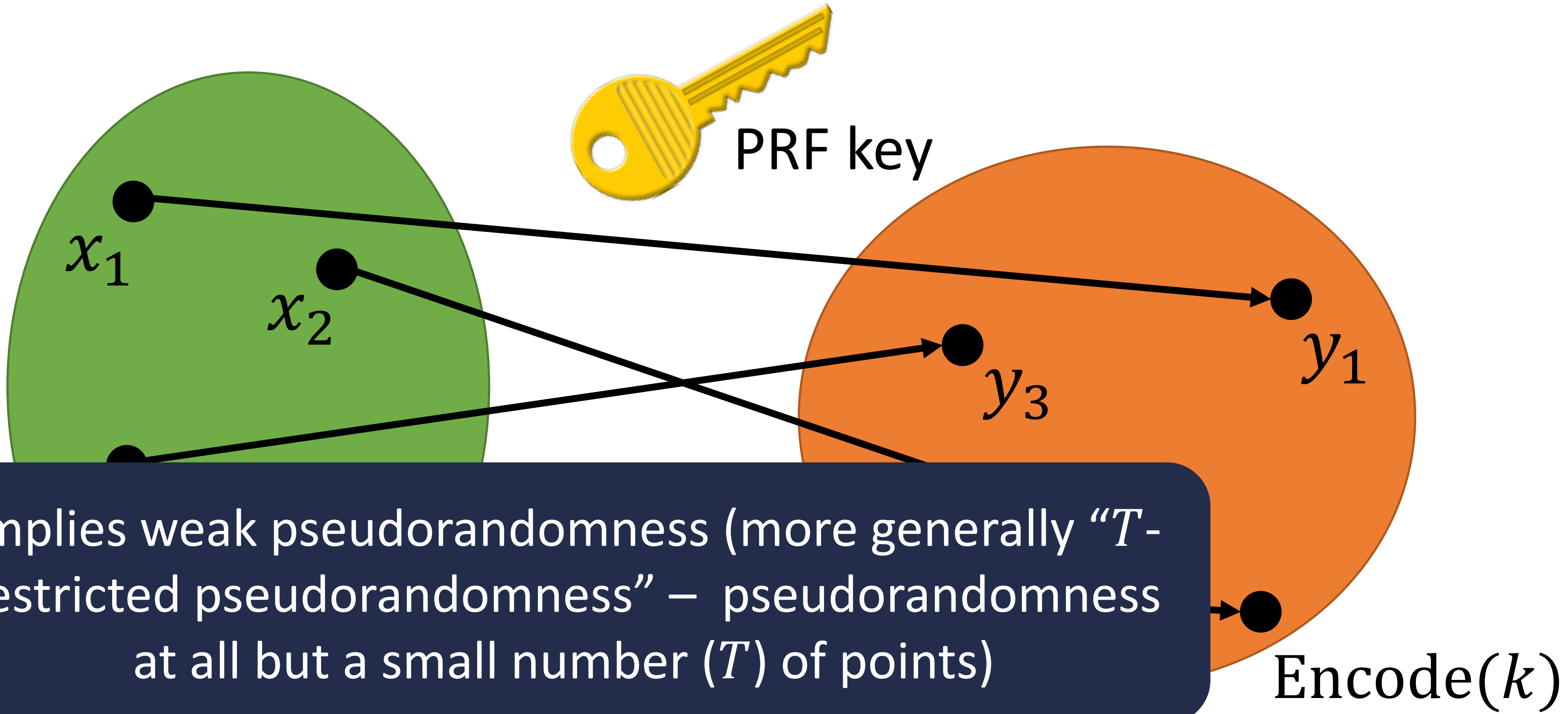
Marked Key

Extraction to Watermarking



Preventing verifier rejection: Queries on circuits that are far away from marked key will always reject, so binary search is no longer effective

Security Against the Authority



Implies weak pseudorandomness (more generally "T-restricted pseudorandomness" – pseudorandomness at all but a small number (T) of points)

PRF keys are pseudorandom everywhere except at z (even given the extraction trapdoor)

Summary

Puncturable Extractable PRF → Watermarkable PRF

High-level overview:

- **Marking:** Puncture PRF key at x^*
- **Verification:** Extract key from circuit, and check correctness of value at x^*

Unremovability: Key-extraction succeeds if circuit if adversary's circuit is close to original PRF; removing the mark requires "patching" PRF at punctured point

Summary

Puncturable Extractable PRF → Watermarkable PRF

Property holds even in the presence of the verification oralce

ey at x^*
from circuit, and check correctness of

Unremovability: Key-extraction succeeds if circuit if adversary's circuit is close to original PRF; removing the mark requires "patching" PRF at punctured point

Constructing Extractable PRFs

Structure of lattice PRFs [BV15]:

PRF on ℓ-bit inputs (e.g., domain $\{0,1\}^\ell$)

$\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell \in \mathbb{Z}_q^{n \times m}$ public matrices (one for each bit of input)

PRF secret key: $\boldsymbol{s} \in \mathbb{Z}_q^n$ (LWE secret)

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

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

Constructing Extractable PRFs

Structure of lattice PRFs [BV15]:

PRF on ℓ-bit inputs (e.g., domain $\{0,1\}^\ell$)

$\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell \in \mathbb{Z}_q^{n \times m}$ public matrices (one for each bit of input)

PRF secret key: $\boldsymbol{s} \in \mathbb{Z}_q^n$ (LWE secret)

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

$\boldsymbol{A}_x$: matrix derived from $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell, x$

Constructing Extractable PRFs

Goal: embed a trapdoor at z such that evaluation at z allows key recovery

Lattice trapdoors [Ajt99, GPV08, AP09, MP12]: can sample

random matrix $\boldsymbol{D} \in \mathbb{Z}_q^{n \times m}$

trapdoor $\mathrm{td}_{\boldsymbol{D}}$

such that LWE is easy with respect to $\boldsymbol{D}$:

given $\boldsymbol{s}^T \boldsymbol{D} + \boldsymbol{e}^T$ and $\mathrm{td}_{\boldsymbol{D}}$, can recover LWE secret $\boldsymbol{s}$

Idea: hide a lattice trapdoor in the public parameters

Constructing Extractable PRFs

$\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell \in \mathbb{Z}_q^{n \times m}$ public matrices (one for each bit of input)

PRF secret key: $\boldsymbol{s} \in \mathbb{Z}_q^n$ (LWE secret)

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

Embed trapdoor at $z \in \{0,1\}^\ell$:

Compute $\boldsymbol{A}_z$ from $\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell$

Let $\boldsymbol{W} = \boldsymbol{D} - \boldsymbol{A}_z$

Include $\boldsymbol{W}$ in the public parameters

Constructing Extractable PRFs

$\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell \in \mathbb{Z}_q^{n \times m}$ public matrices (one for each bit of input)

PRF secret key: $\boldsymbol{s} \in \mathbb{Z}_q^n$ (LWE secret)

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

$\boldsymbol{W}$ hides $\boldsymbol{A}_z$ (and thus, z) since $\boldsymbol{D}$ is statistically close to uniform

Include $\boldsymbol{W}$ in the public parameters

PRF evaluation at input x: $\mathrm{PRF}(\boldsymbol{s}, x) \coloneqq \lfloor \boldsymbol{s}^T (\boldsymbol{A}_x + \boldsymbol{W}) \rceil_p$

Constructing Extractable PRFs

$\boldsymbol{A_1}, \dots, \boldsymbol{A}_\ell \in \mathbb{Z}_q^{n \times m}$ public matrices (one for each bit of input)

PRF secret key: $\boldsymbol{s} \in \mathbb{Z}_q^n$ (LWE secret)

PRF evaluation at input x: $\mathrm{PRF}(\boldsymbol{s}, x) := \lfloor \boldsymbol{s}^T \boldsymbol{A}_x \rceil_p$

Embed trapdoor at $z \in \{0,1\}^\ell$:

Co

Le

In r

Value everywhere else is still pseudorandom

Value at z is $\lfloor \boldsymbol{s}^T (\boldsymbol{A_z} + \boldsymbol{W}) \rceil_p = \lfloor \boldsymbol{s}^T \boldsymbol{D} \rceil_p$, so can extract $\boldsymbol{s}$ using trapdoor $\mathrm{td}_{\boldsymbol{D}}$

PRF evaluation at input x: $\mathrm{PRF}(\boldsymbol{s}, x) := \lfloor \boldsymbol{s}^T (\boldsymbol{A}_x + \boldsymbol{W}) \rceil_p$

Summary

Puncturable Extractable PRF → Watermarkable PRF

Puncturable extractable PRF can be built from LWE
(with a nearly polynomial modulus-to-noise ratio)

Yields new watermarking scheme from LWE with
security in the presence of verification oracle

Extensions: Message-embedding, mark-unforgeability [See paper…]

Open Problems

Extractable PRFs from generic techniques?

More applications of extractable PRFs?

Publicly-verifiable watermarking scheme for PRFs?

Thank you!

`http://eprint.iacr.org/2018/986`