Watermarking Cryptographic Functionalities from Standard Lattice Assumptions

Sam Kim and David J. Wu

Stanford University

Digital Watermarking

Often used to identify owner of content and prevent unauthorized distribution

Digital Watermarking

- Content is (mostly) viewable

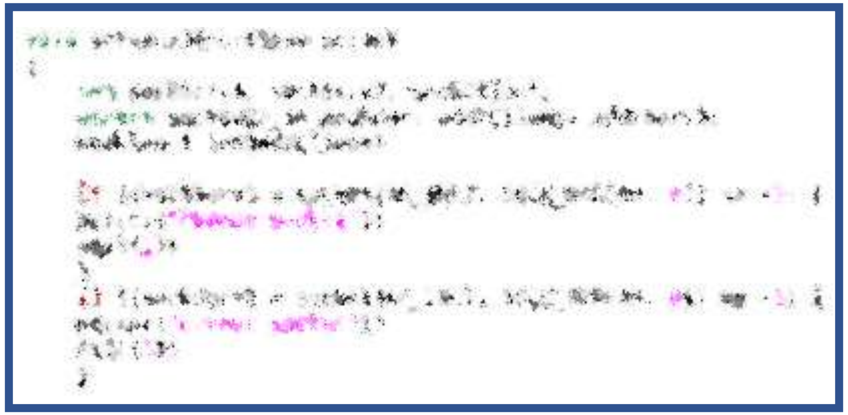
Digital Watermarking

- Content is (mostly) viewable
- Watermark difficult to remove (without destroying the image)

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17]

Embed a “mark” within a program



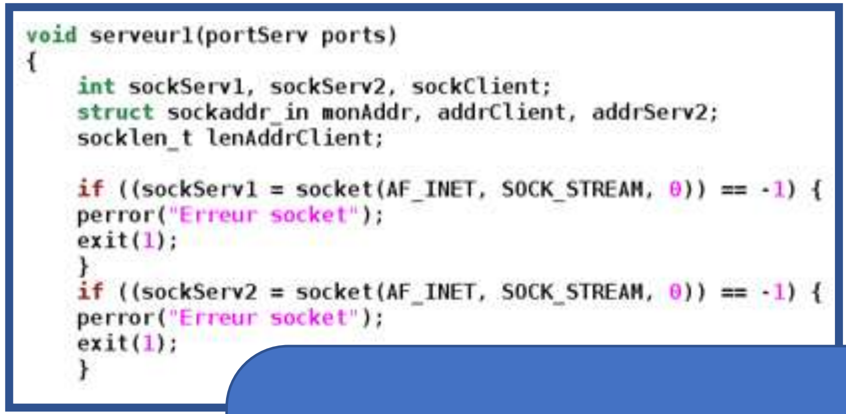
If mark is removed, then program is destroyed

Two main algorithms:

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

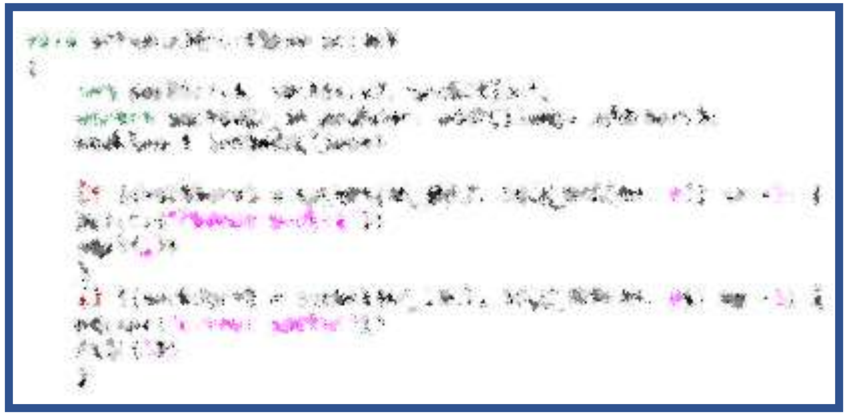
Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17]



```
void serveur1(portServ ports)
{
    int sockServ1, sockServ2, sockClient;
    struct sockaddr_in monAddr, addrClient, addrServ2;
    socklen_t lenAddrClient;

    if ((sockServ1 = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Erreur socket");
    exit(1);
    }
    if ((sockServ2 = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Erreur socket");
    exit(1);
    }
```



Embe

Both marking and verification require a secret watermarking key wsk

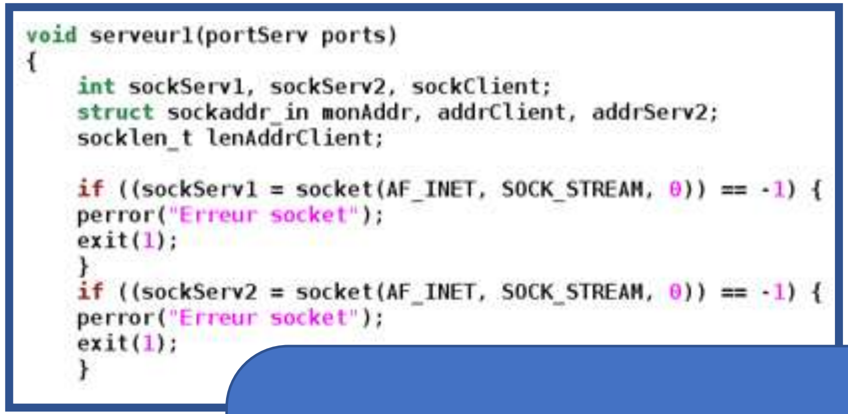
If mark is removed, then program is destroyed

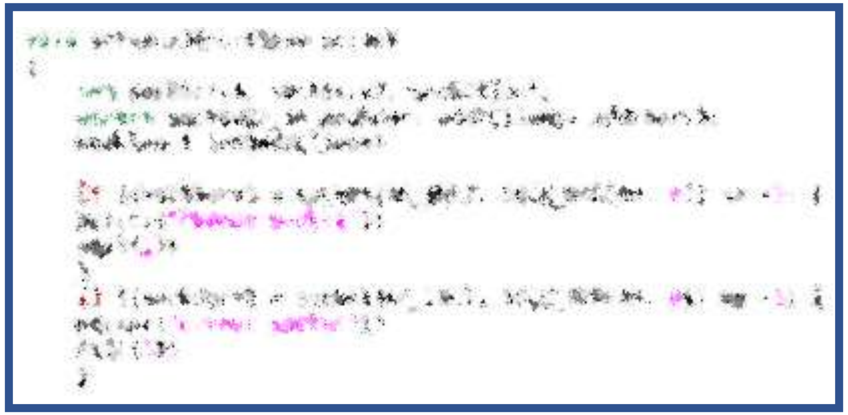
Two mai

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

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17]





Embe

Extends to setting where watermark can be an (arbitrary) string [See paper]

If mark is removed, then program is destroyed

Two mai alg s:

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

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17]

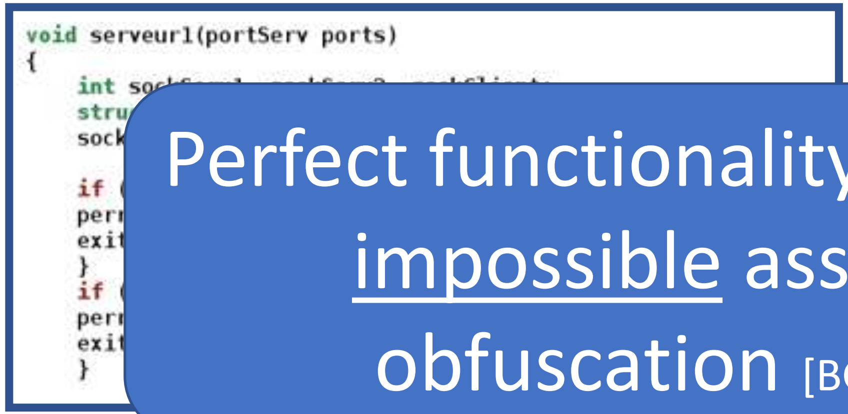
Functionality-preserving: On input a program (modeled as a circuit C), the Mark algorithm outputs a circuit C' where:

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

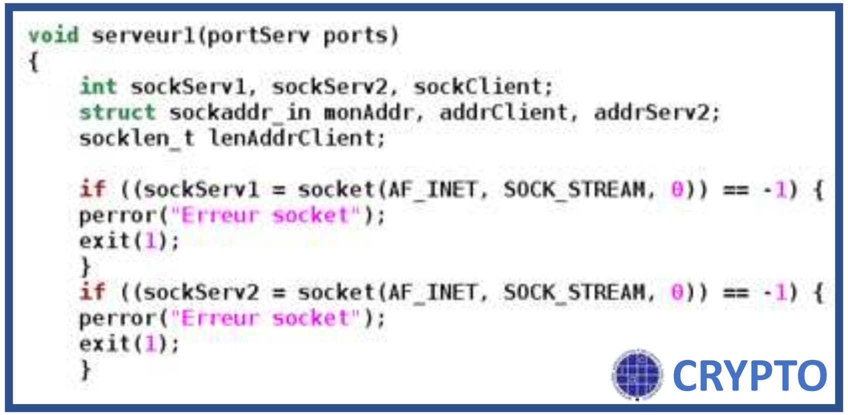
on all but a negligible fraction of inputs x

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17]



Perfect functionality-preserving <u>impossible</u> assuming obfuscation [BGIRSVY12]



Functionality-preserving: On input a program (modeled as a circuit C), the Mark algorithm outputs a circuit C' where:

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

on all but a negligible fraction of inputs x

Watermarking Programs

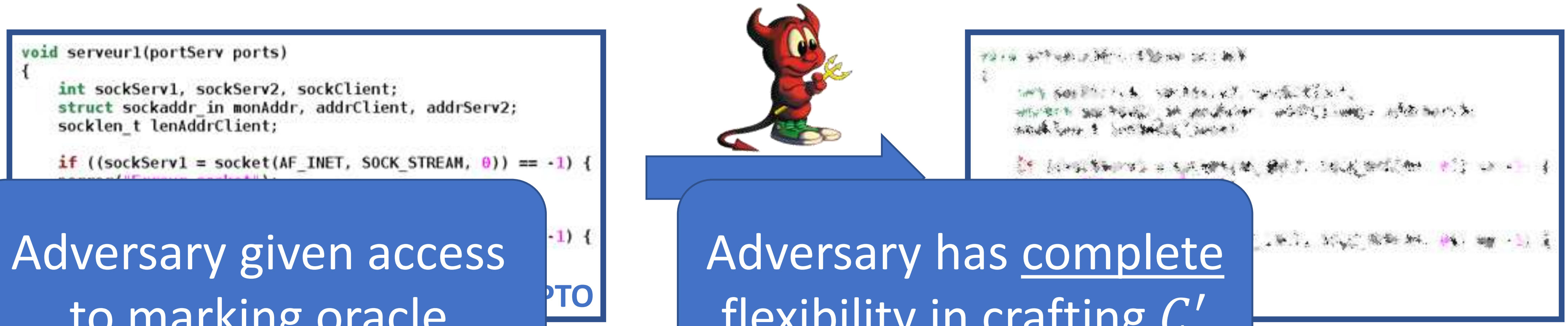
[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17]

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

- $C'(x) = C(x)$ on all but a negligible fraction of inputs x
- $\text{Verify}(\text{wsk}, C') = 1$

Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17]



Adversary given access to marking oracle

Adversary has <u>complete</u> flexibility in crafting C'

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

- $C'(x) = C(x)$ on all but a negligible fraction of inputs x
- $\mathrm{Verify}(\mathrm{wsk}, C') = 1$

[Also require <u>unforgeability</u>; see paper for details]

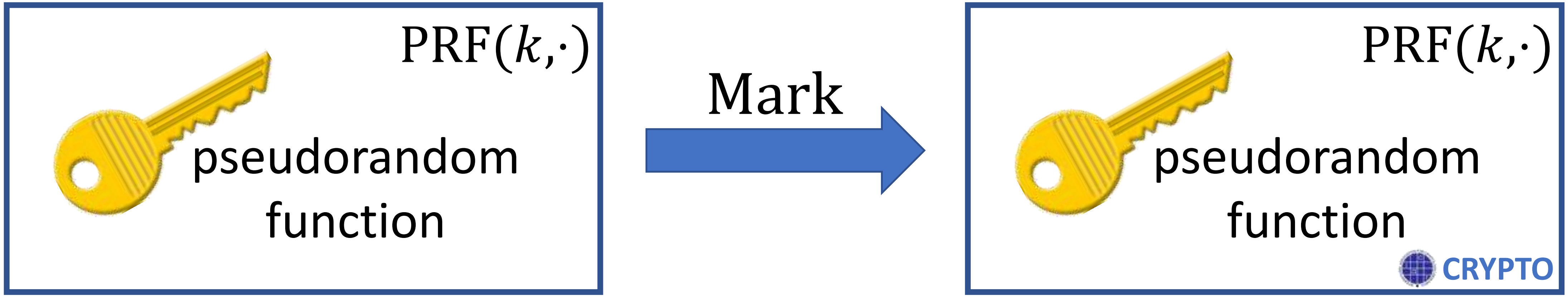
Watermarking Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17]

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

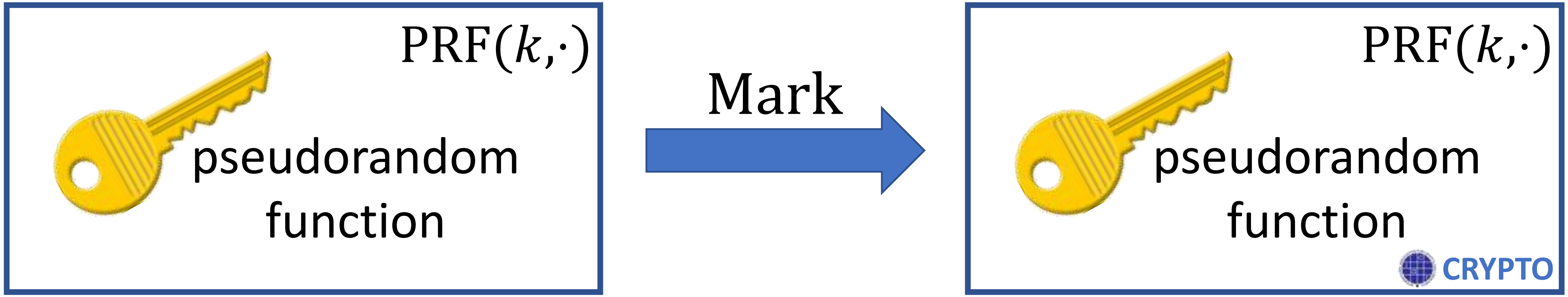
Watermarking Cryptographic Programs

[NSS99, BGIRSVY01, HMW07, YF11, Nis13, CHNVW16, BLW17]



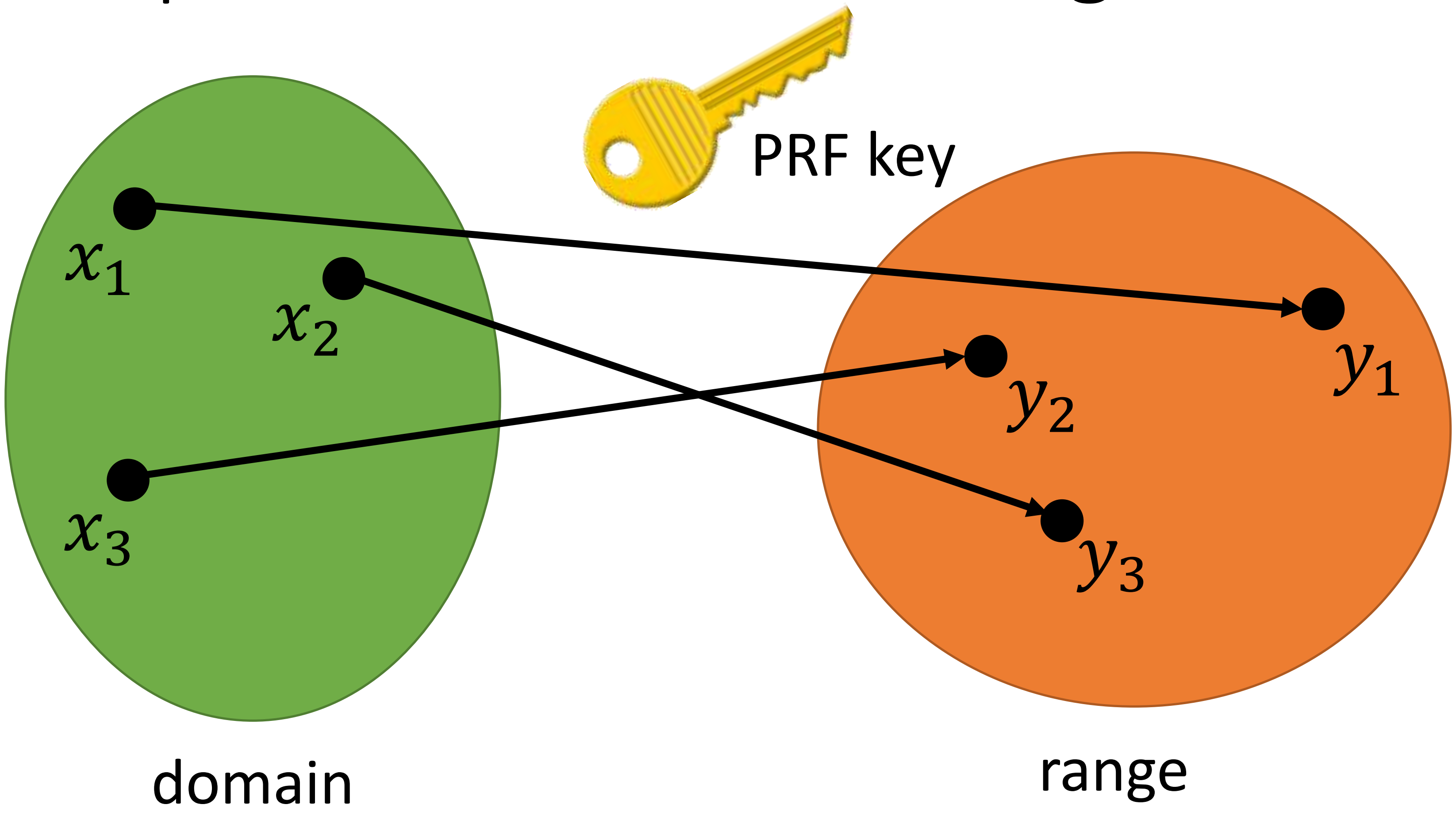
- Focus of this work: watermarking PRFs [CHNVW16, BLW17]
- Enables watermarking of symmetric primitives built from PRFs (e.g., encryption, MACs, etc.)

Main Result



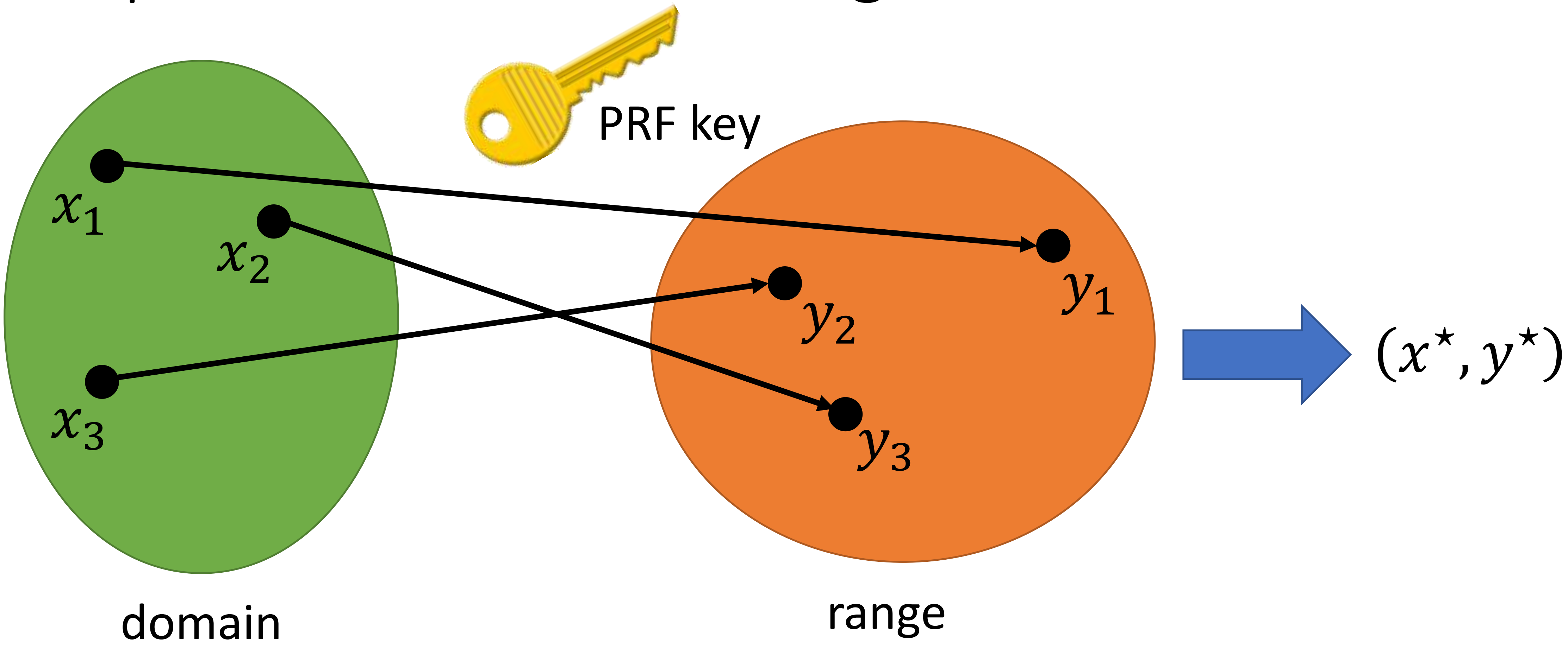
This work: Under *standard lattice assumptions*, there exists a (secretly)-verifiable watermarkable family of PRFs.

Blueprint for Watermarking PRFs [CHNVW16, BLW17]



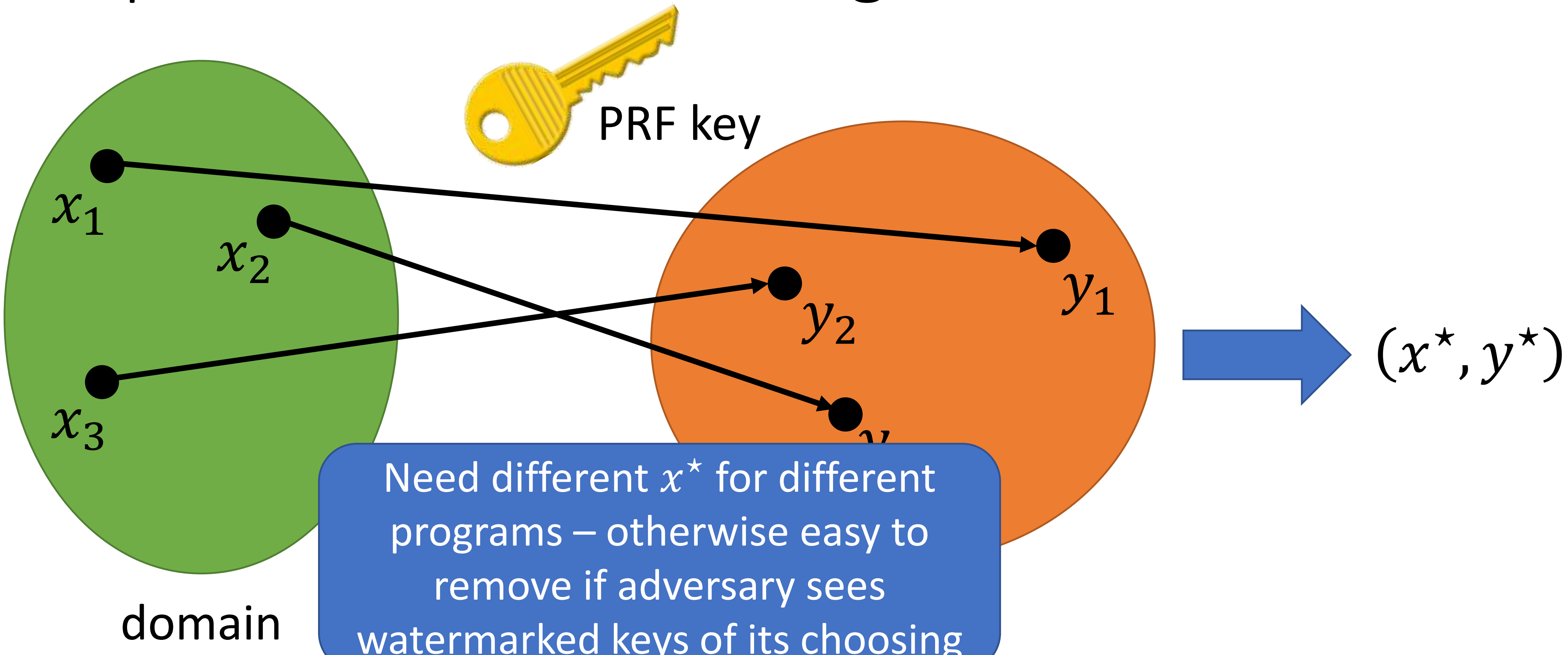
Step 1: Evaluate PRF on test points x_1, x_2, x_3 (part of the watermarking secret key)

Blueprint for Watermarking PRFs [CHNVW16, BLW17]



Step 2: Derive a pair $(x^\star, y^\star)$ from y_1, y_2, y_3

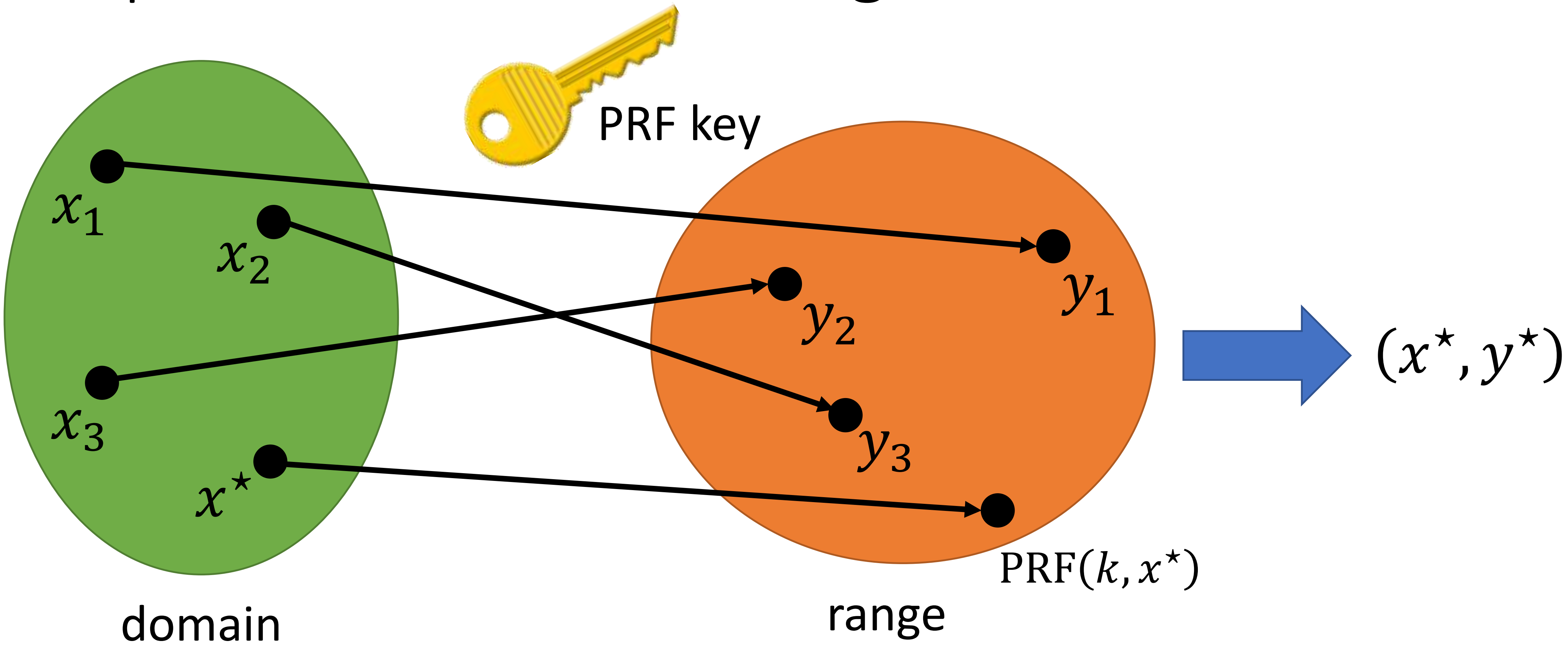
Blueprint for Watermarking PRFs [CHNVW16, BLW17]



Need different $x^\star$ for different programs – otherwise easy to remove if adversary sees watermarked keys of its choosing

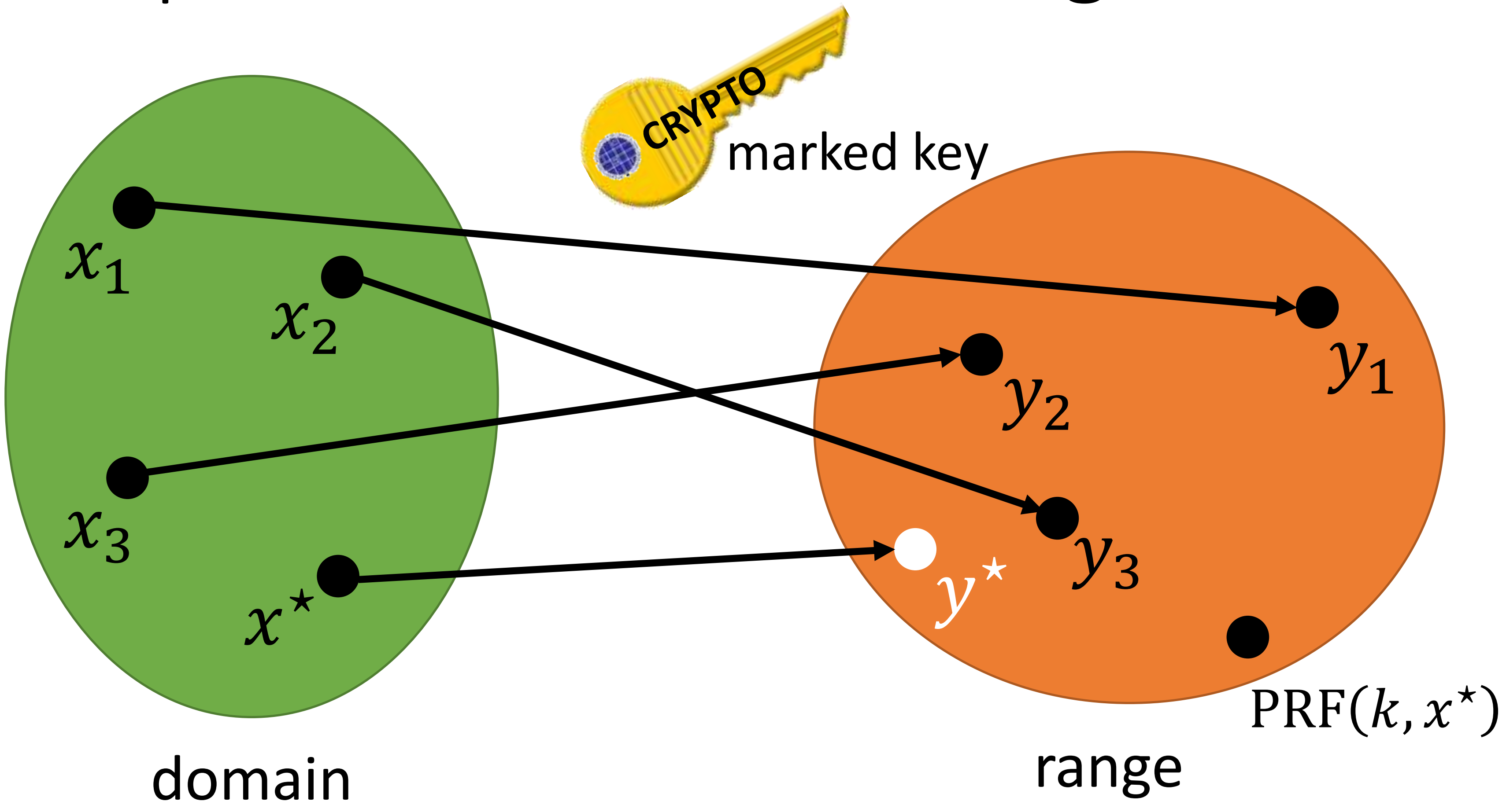
Step 2: Derive a pair $(x^\star, y^\star)$ from y_1, y_2, y_3

Blueprint for Watermarking PRFs [CHNVW16, BLW17]



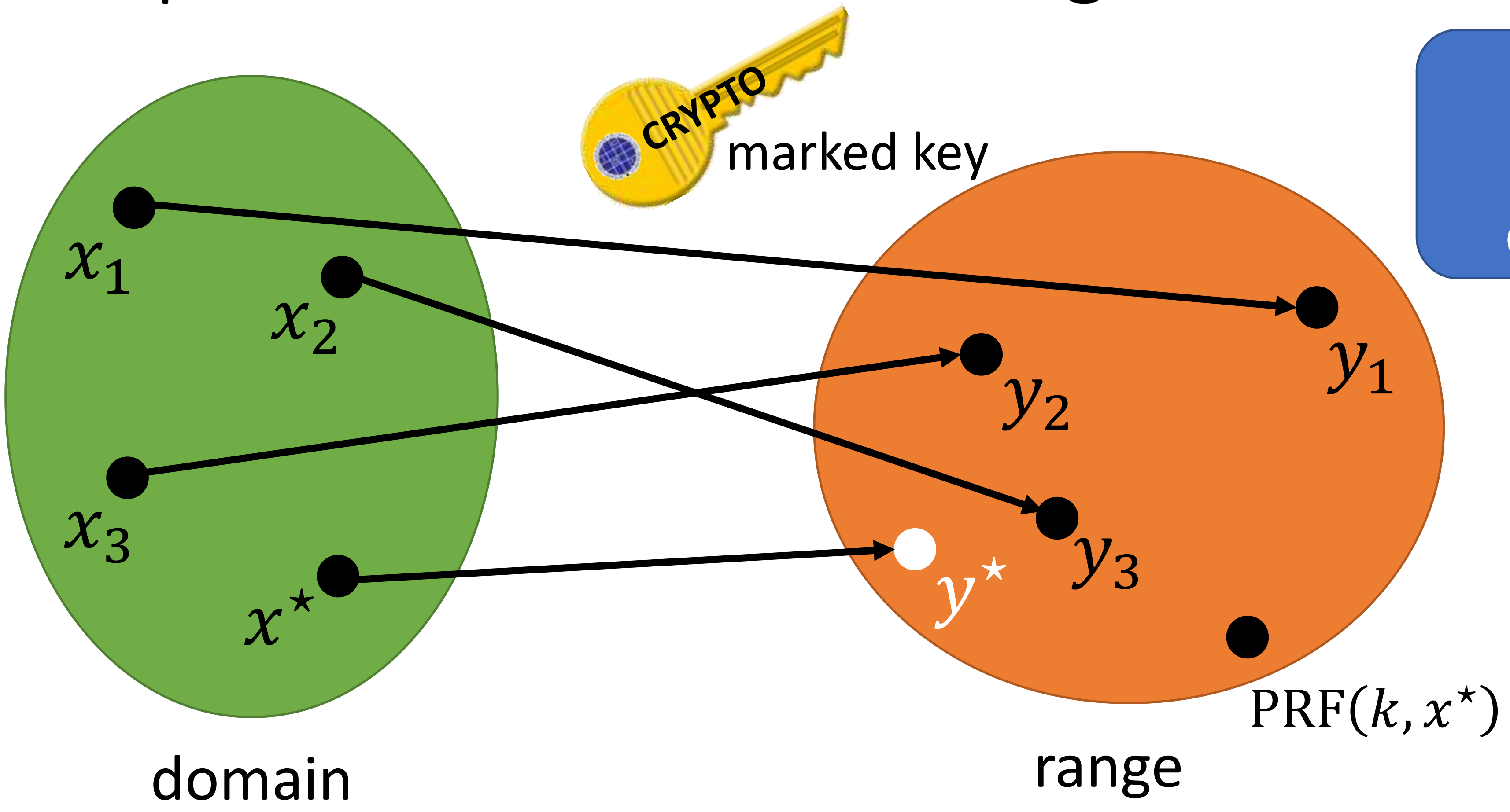
Step 3: "Marked key" is a circuit that implements the PRF at all points, except at $x^\star$, the output is changed to $y^\star$

Blueprint for Watermarking PRFs [CHNVW16, BLW17]



Step 3: "Marked key" is a circuit that implements the PRF at all points, except at $x^\star$, the output is changed to $y^\star$

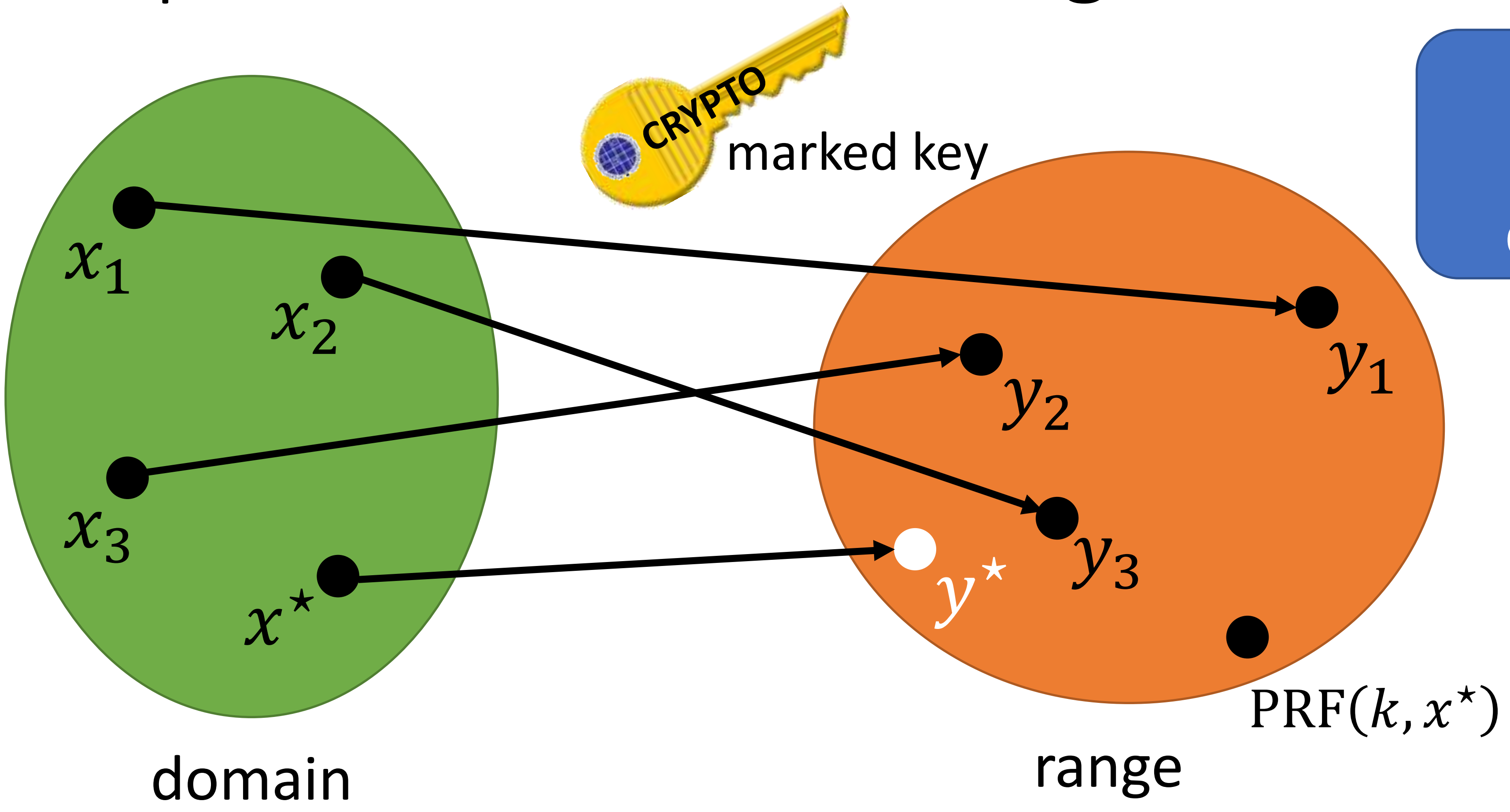
Blueprint for Watermarking PRFs [CHNVW16, BLW17]



Defer implementation details for now…

Step 3: "Marked key" is a circuit that implements the PRF at all points, except at $x^\star$, the output is changed to $y^\star$

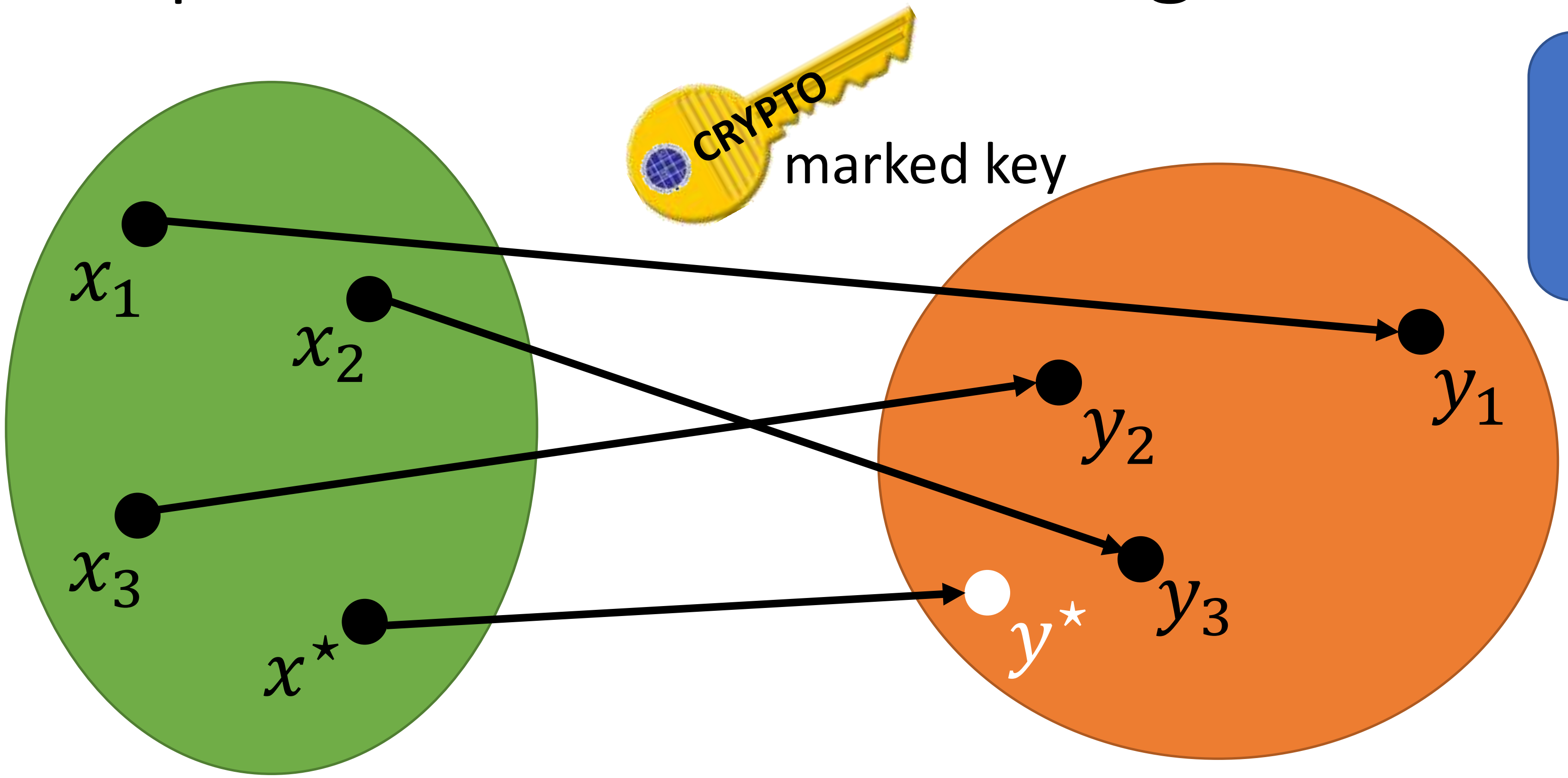
Blueprint for Watermarking PRFs [CHNVW16, BLW17]



Defer implementation details for now…

Verification: Evaluate function at x_1, x_2, x_3, derive $(x^\star, y^\star)$ and check if the value at $x^\star$ matches $y^\star$

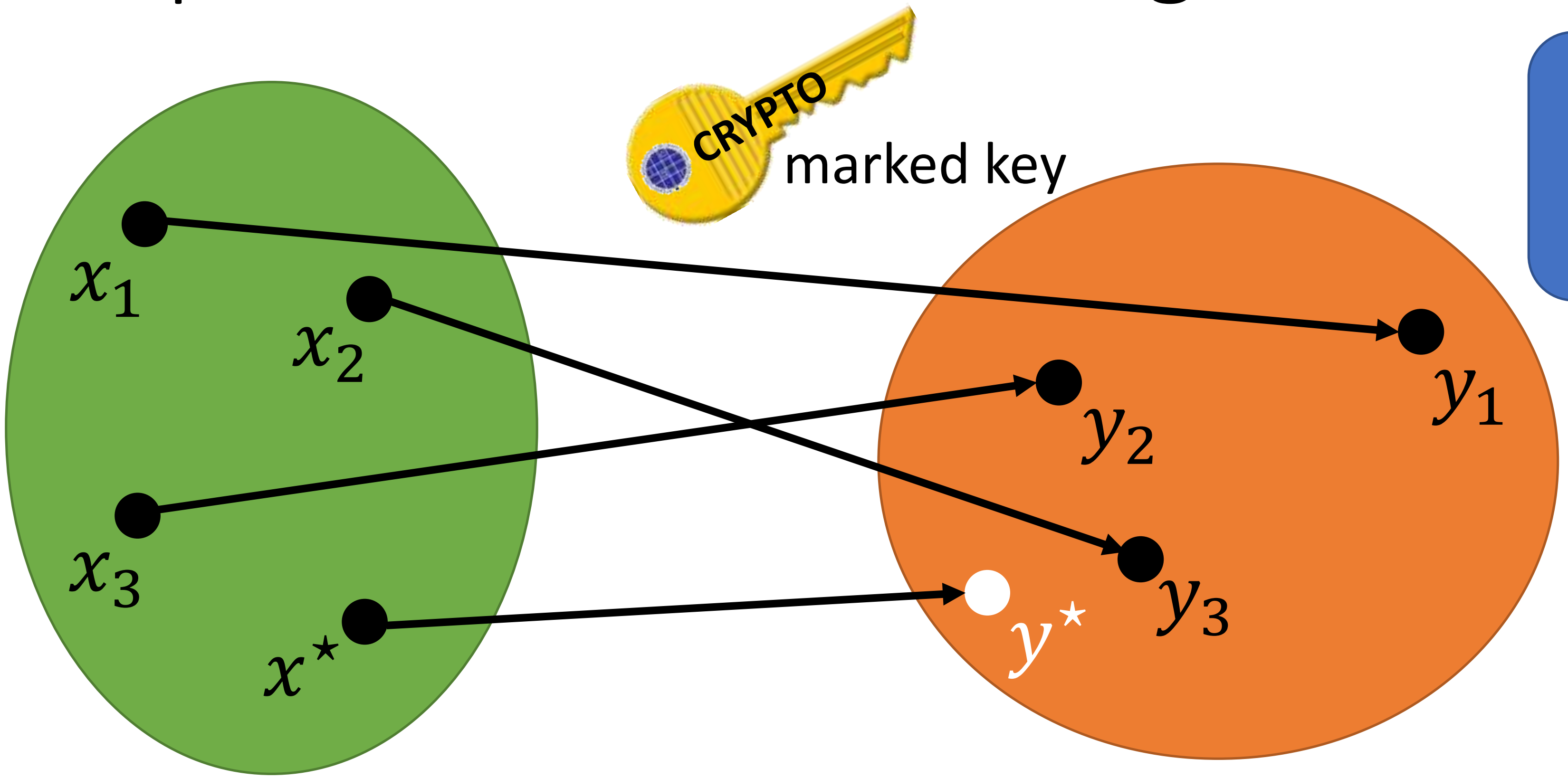
Blueprint for Watermarking PRFs [CHNVW16, BLW17]



Defer implementation details for now…

☑ Functionality-preserving: function differs at a single point

Blueprint for Watermarking PRFs [CHNVW16, BLW17]

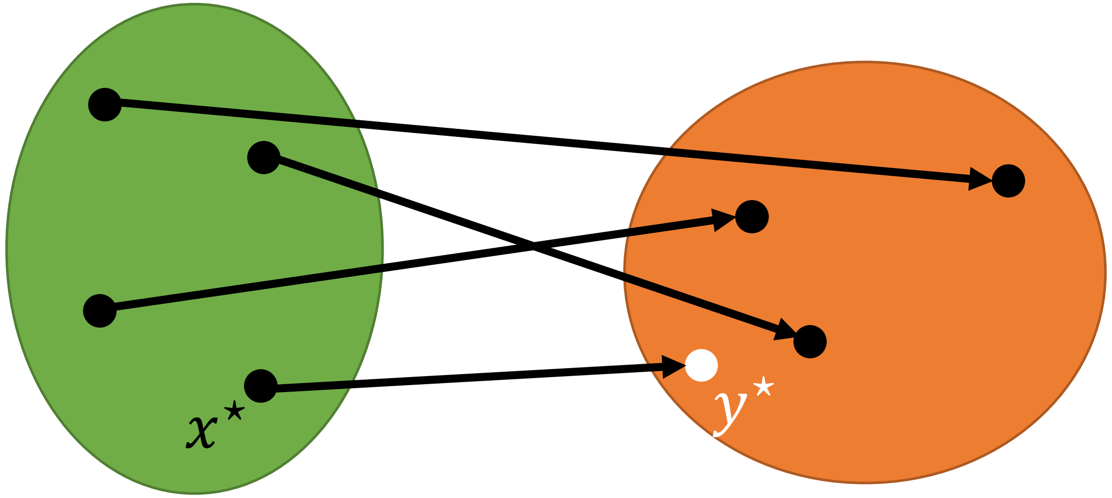


Defer implementation details for now…

☑ Functionality-preserving: function differs at a single point

☑ Unremovable: as long as adversary cannot tell that $(x^\star, y^\star)$ is “special”

Blueprint for Watermarking PRFs [CHNVW16, BLW17]



Prior solutions: use obfuscation to hide $(x^\star, y^\star)$

How to implement this functionality?

Blueprint for Watermarking PRFs [CHNVW16, BLW17]

Obfuscated program:

$P_{(x^\star, y^\star)}(x)$:
- if $x = x^\star$, output $y^\star$
- else, output $\mathrm{PRF}(k, x)$

Prior solutions: use obfuscation to hide $(x^\star, y^\star)$

Obfuscated program has PRF key embedded inside and outputs $\mathrm{PRF}(k, x)$ on all inputs $x \neq x^\star$ and $y^\star$ when $x = x^\star$

How to implement this functionality?

Blueprint for Watermarking PRFs [CHNVW16, BLW17]

Obfuscated program:

$P_{(x^\star, y^\star)}(x)$:

- if $x = x^\star$, output $y^\star$
- else, output $\mathrm{PRF}(k, x)$

Prior solutions: use obfuscation to hide $(x^\star, y^\star)$

Obfuscated program has PRF key embedded inside and outputs $\mathrm{PRF}(k, x)$ on all inputs $x \neq x^\star$ and $y^\star$ when $x = x^\star$

Essentially relies on secretly *re-programming* the value at $x^\star$

functionality?

Blueprint for Watermarking PRFs [CHNVW16, BLW17]

Obfuscated program:

$P_{(x^\star, y^\star)}(x)$:
- if $x = x^\star$, output $y^\star$
- else, output $\mathrm{PRF}(k, x)$

Prior solutions: use obfuscation to hide $(x^\star, y^\star)$

Obfuscated program has PRF key embedded inside and outputs $\mathrm{PRF}(k, x)$ on all inputs $x \neq x^\star$ and $y^\star$ when $x = x^\star$

Key technical challenge: How to hide $(x^\star, y^\star)$ within the watermarked key (without obfuscation)?

Blueprint for Watermarking PRFs [CHNVW16, BLW17]

Obfuscated program:

$P_{(x^\star, y^\star)}(x)$:

- if $x = x^\star$, output $y^\star$
- else, output $\mathrm{PRF}(k, x$

Prior solutions: use obfuscation to hide $(x^\star, y^\star)$

F key

ts

$x^\star$

Has an obfuscation flavor: need to embed a secret inside a piece of code that cannot be removed

Key technical challenge: How to hide $(x^\star, y^\star)$ within the watermarked key (without obfuscation)?

Blueprint for Watermarking PRFs [CHNVW16, BLW17]

Obfuscated program:

$P_{(x^\star, y^\star)}(x)$:

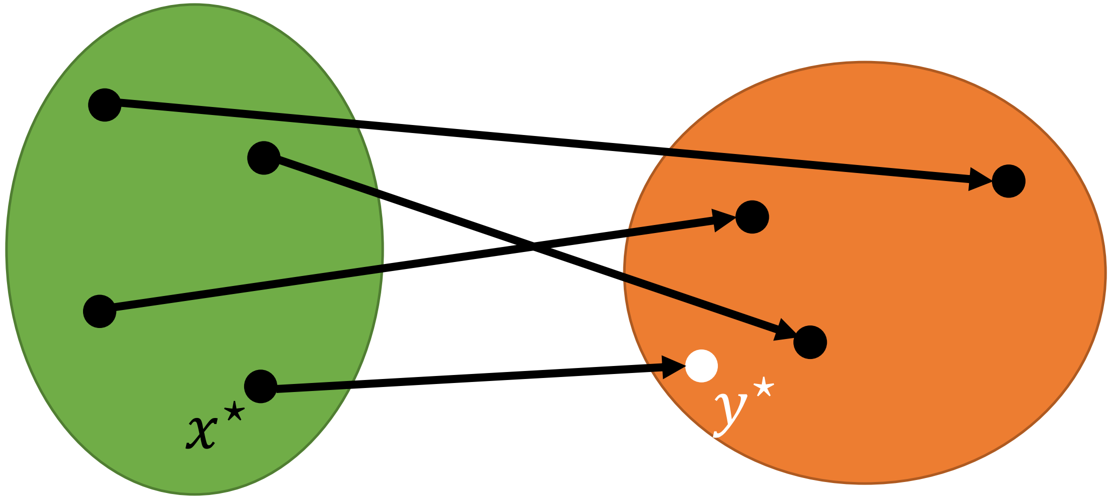
- if $x = x^\star$, output $y^\star$
- else, output $\mathsf{PRF}(k, x)$

Prior solutions: use obfuscation to hide $(x^\star, y^\star)$

Obfuscated program has PRF key embedded inside and outputs $\mathrm{PRF}(k, x)$ on all inputs $x \neq x^\star$ and $y^\star$ when $x = x^\star$

This work: Under *standard lattice assumptions*, there exists a (secretly)-verifiable watermarkable family of PRFs.

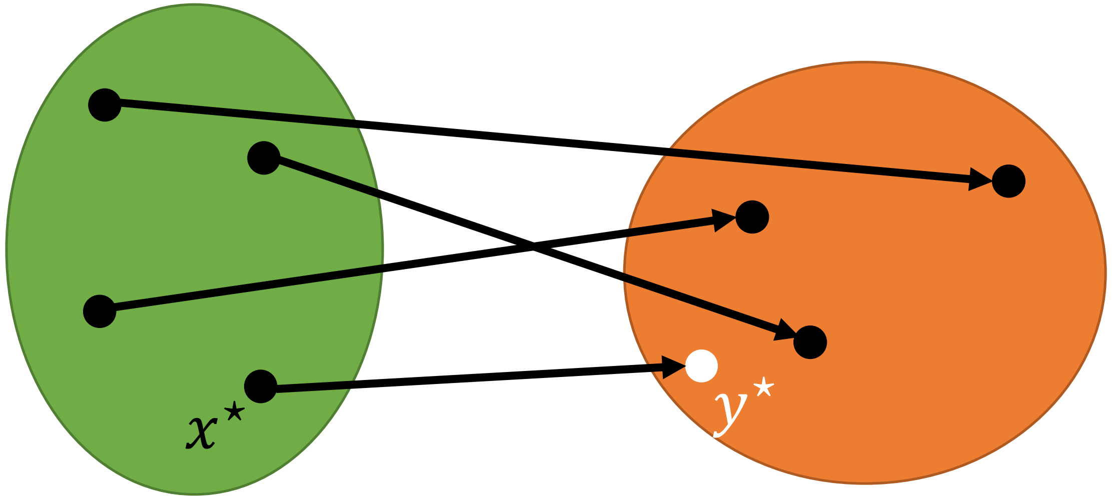
Starting Point: Private Puncturable PRFs [BLW17, BKM17, CC17]



- Watermarked PRF implements PRF at all but a single point
- Structurally very similar to a *puncturable PRF* [BW13, BGI13, KPTZ13]

Puncturable PRF:

Starting Point: Private Puncturable PRFs [BLW17, BKM17, CC17]



- Watermarked PRF implements PRF at all but a single point
- Structurally very similar to a

Can be used to evaluate the PRF on all points $x \neq x^\star$

Puncturable PRF:

PRF key

$\text{Puncture}_{x^\star}$

punctured key

Starting Point: Private Puncturable PRFs [BLW17, BKM17, CC17]

PRF key

$\text{Puncture}_{x^\star}$

punctured key

Recall general approach for watermarking:

1. Derive $(x^\star, y^\star)$ from input/output behavior of PRF
2. Give out a key that agrees with PRF everywhere, except has value $y^\star$ at $x = x^\star$

PRF key punctured at $x^\star$

However, punctured key does not necessarily hide $x^\star$, which allows adversary to remove watermark

Starting Point: Private Puncturable PRFs [BLW17, BKM17, CC17]

Punctured keys typically do not provide flexibility in programming value at punctured point: difficult to test if a program is watermarked or not

Re

1. … or of PRF
2. Give … ey that agrees with PRF everywhere, except has value $y^\star$ at $x = x^\star$

PRF key punctured at $x^\star$

However, punctured key does not necessarily hide $x^\star$, which allows adversary to remove watermark

Starting Point: Private Puncturable PRFs [BLW17, BKM17, CC17]

Problem 1: Punctured keys do not hide the punctured point $x^\star$

- Use *private* puncturable PRFs

Problem 2: Difficult to test whether a value is the result of using a punctured key to evaluate at the punctured point

Starting Point: Private Puncturable PRFs [BLW17, BKM17, CC17]

In existing lattice-based private puncturable PRF constructions [BKM17, CC17], value of punctured key at punctured point is a *deterministic* function of the PRF key

Problem 1: P

- Use *pri*

Problem 2: Difficult to test whether a value is the result of using a punctured key to evaluate at the punctured point

Starting Point: Private Puncturable PRFs [BLW17, BKM17, CC17]

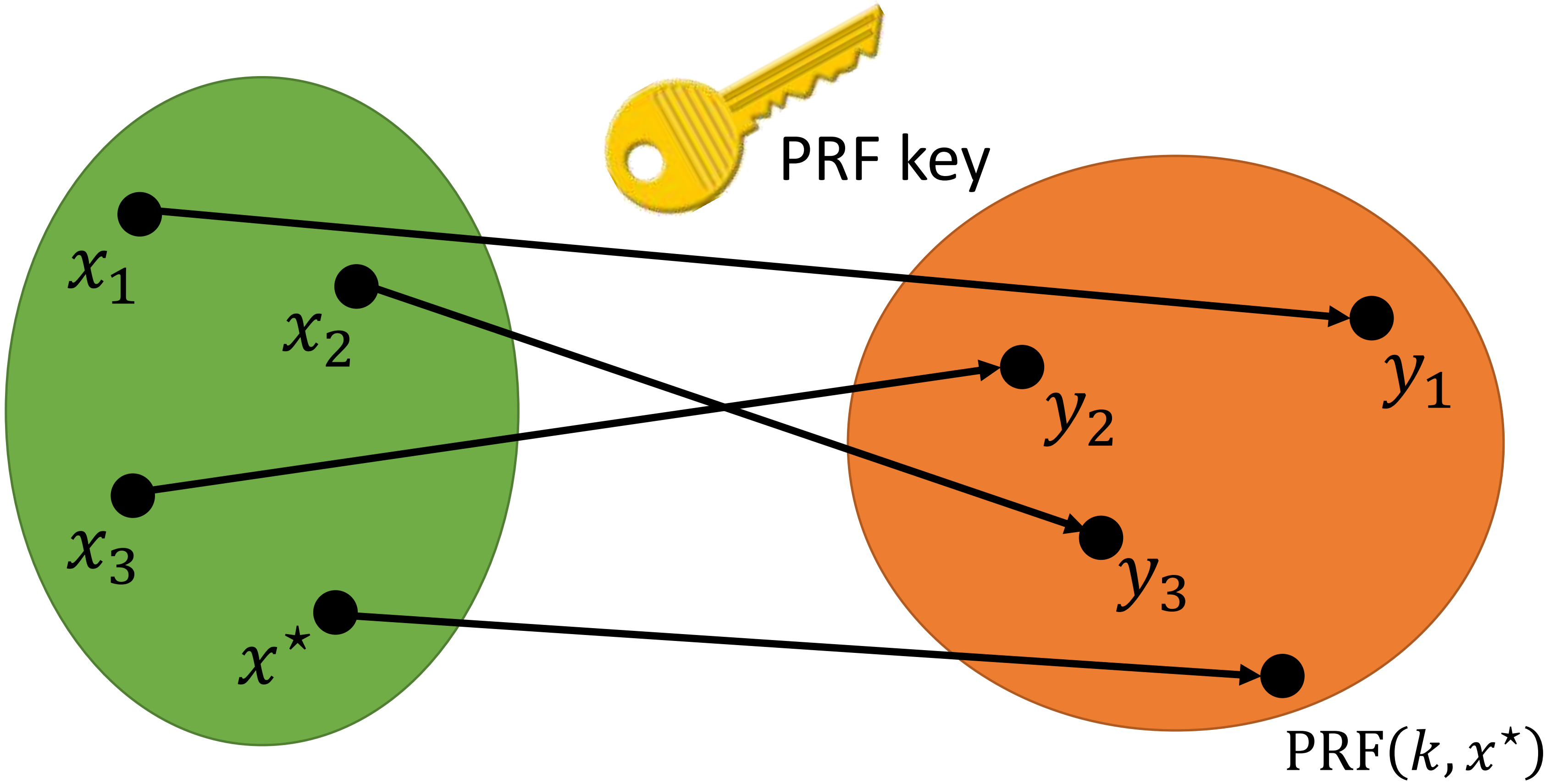
Problem 1: Punctured keys do not hide the punctured point $x^\star$

- Use *privately* puncturable PRFs

Problem 2: Difficult to test whether a value is the result of using a punctured key to evaluate at the punctured point

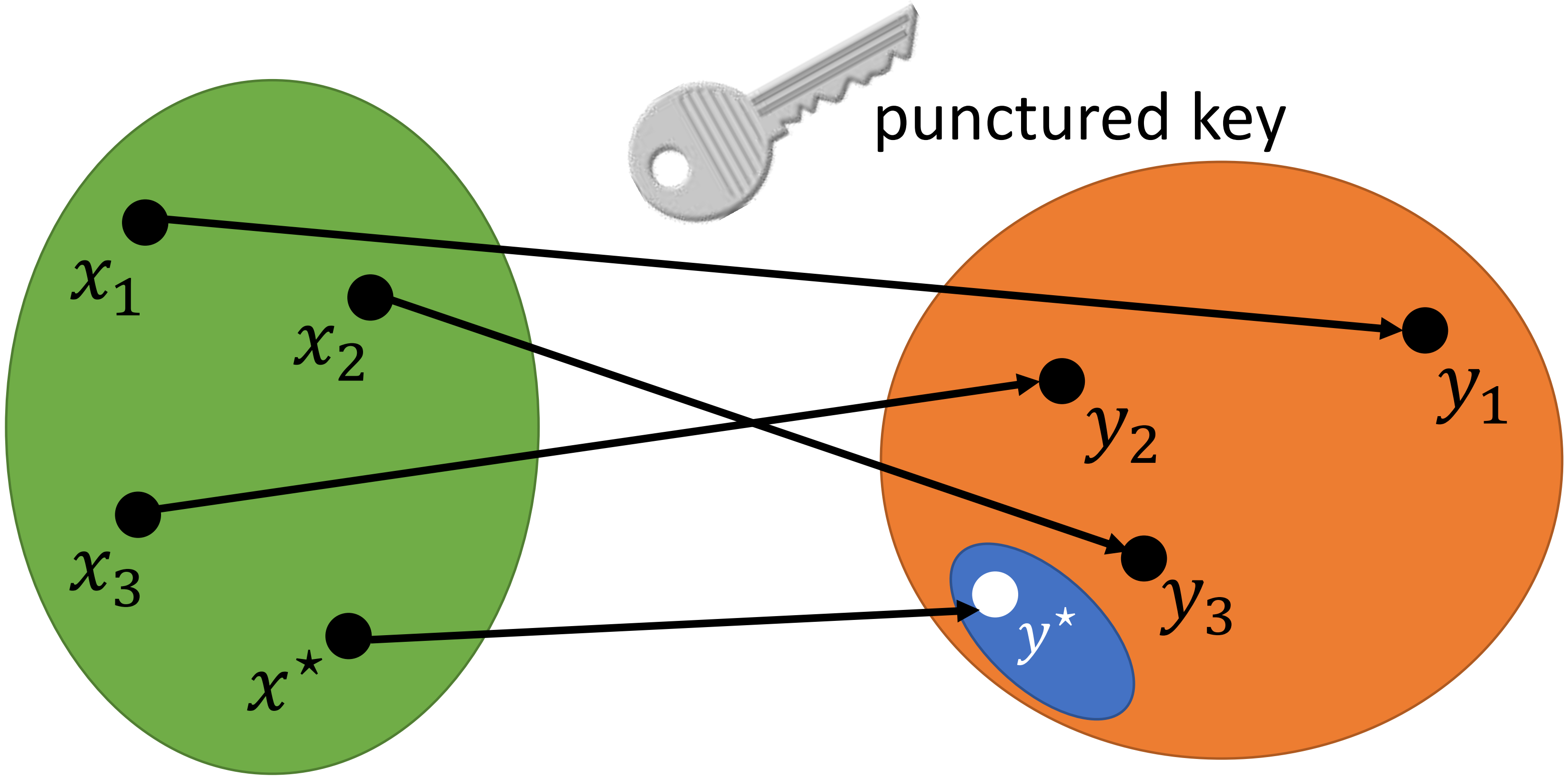
- Relax programmability requirement

Private Translucent PRFs



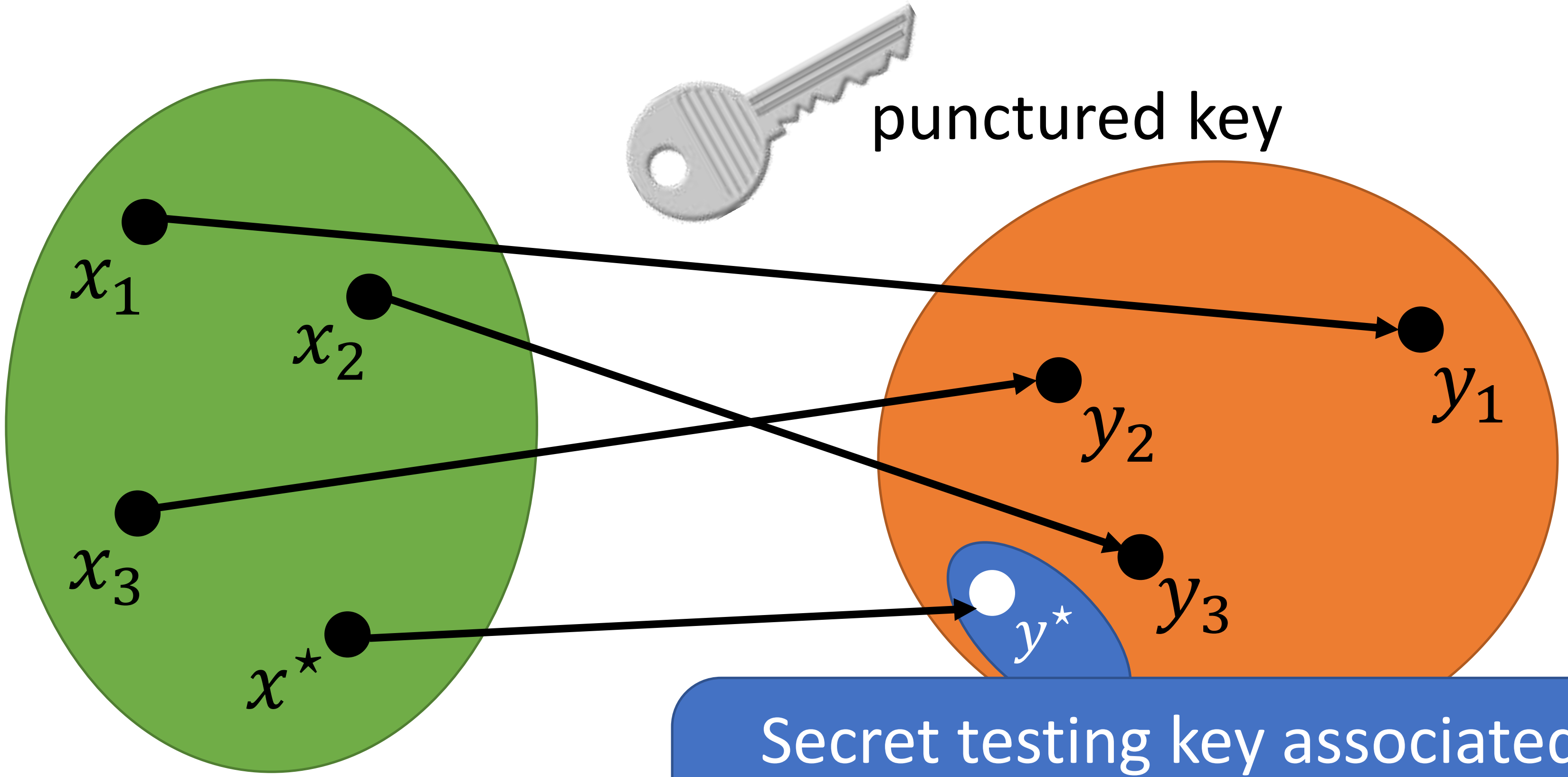
Private puncturable PRF *family* with the property that output of any punctured key on a punctured point lies in a sparse, hidden subspace

Private Translucent PRFs



Private puncturable PRF *family* with the property that output of any punctured key on a punctured point lies in a sparse, hidden subspace

Private Translucent PRFs

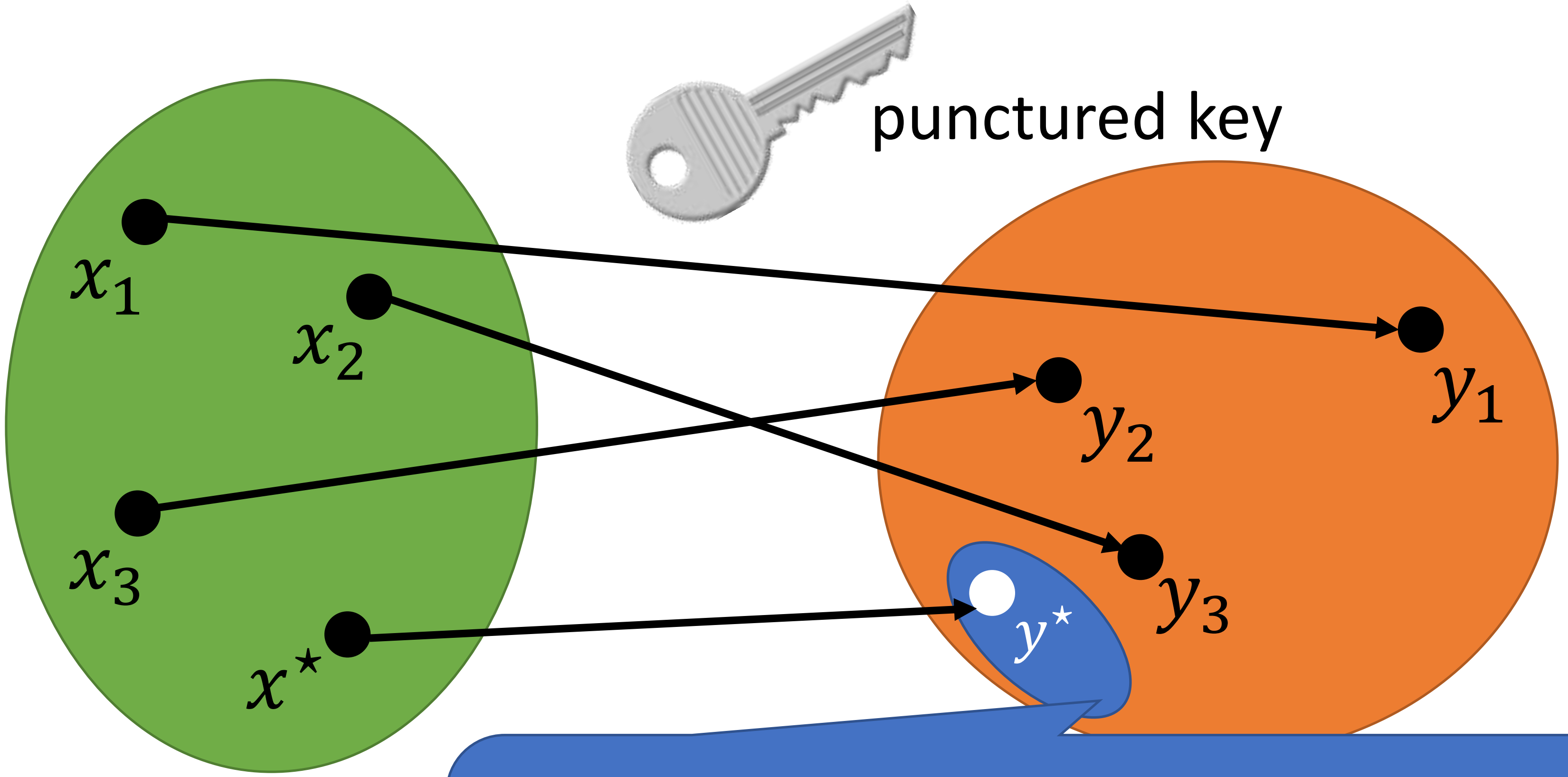


Secret testing key associated with the PRF family can be used to test for membership in the hidden subspace

Private puncturable PRF *family*

punctured key on a punctured point lies in a sparse, hidden subspace

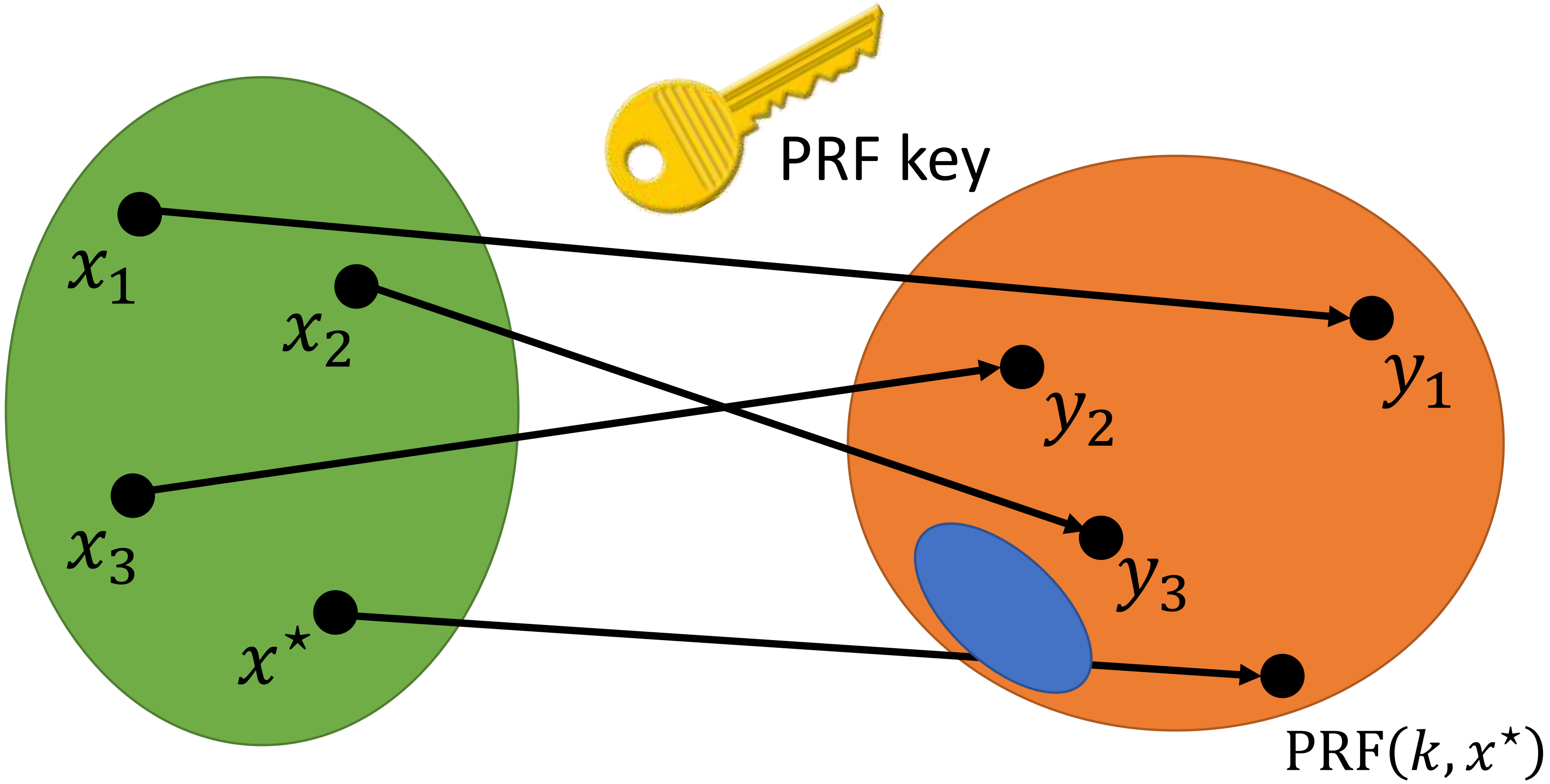
Private Translucent PRFs



- Values in special set looks indistinguishable from a random value (without secret testing key)
- Indistinguishable even though it is easy to sample values from the set

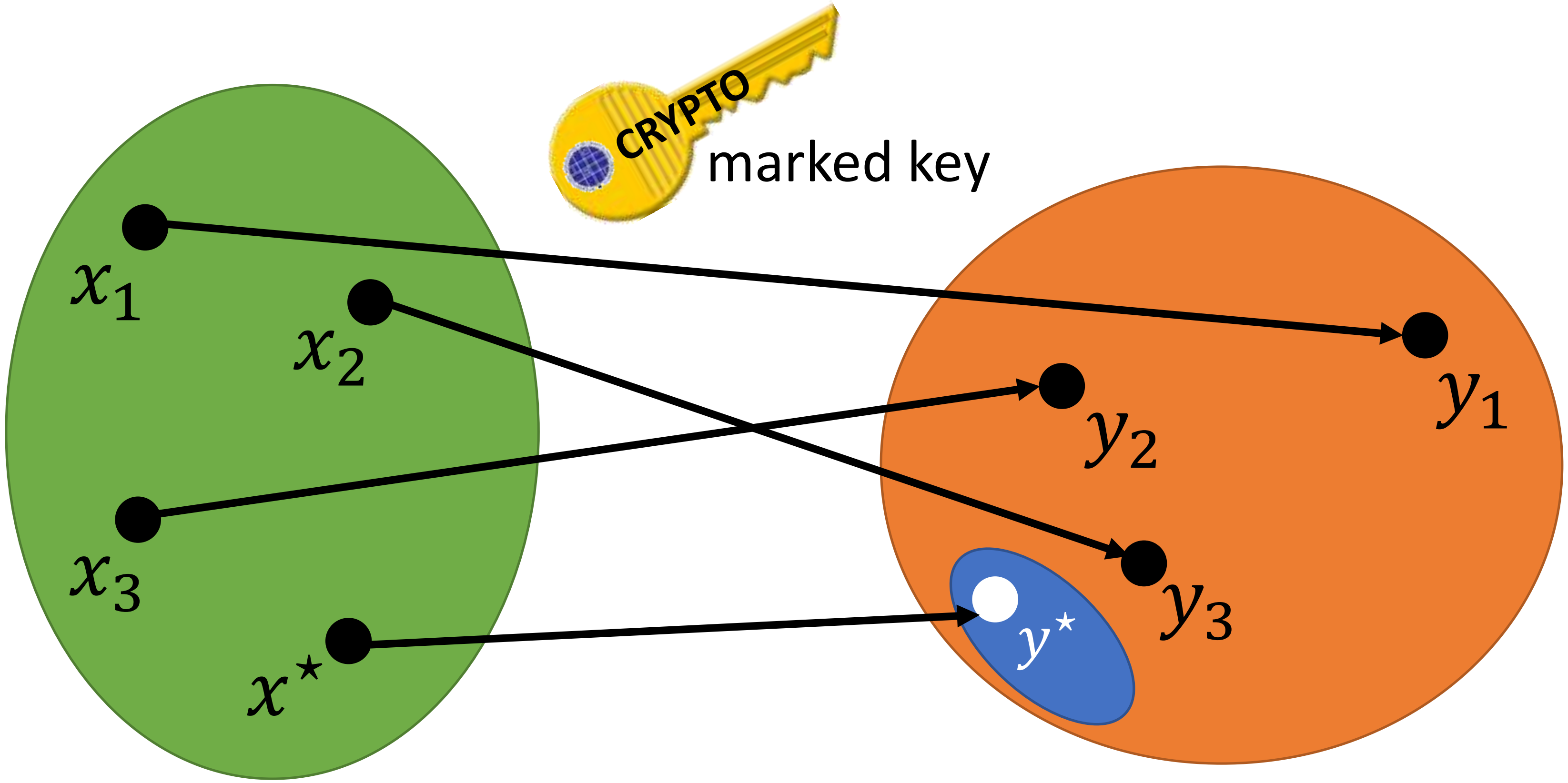
Sets satisfying such properties are called *translucent* [CDNO97]

Watermarking from Private Translucent PRFs



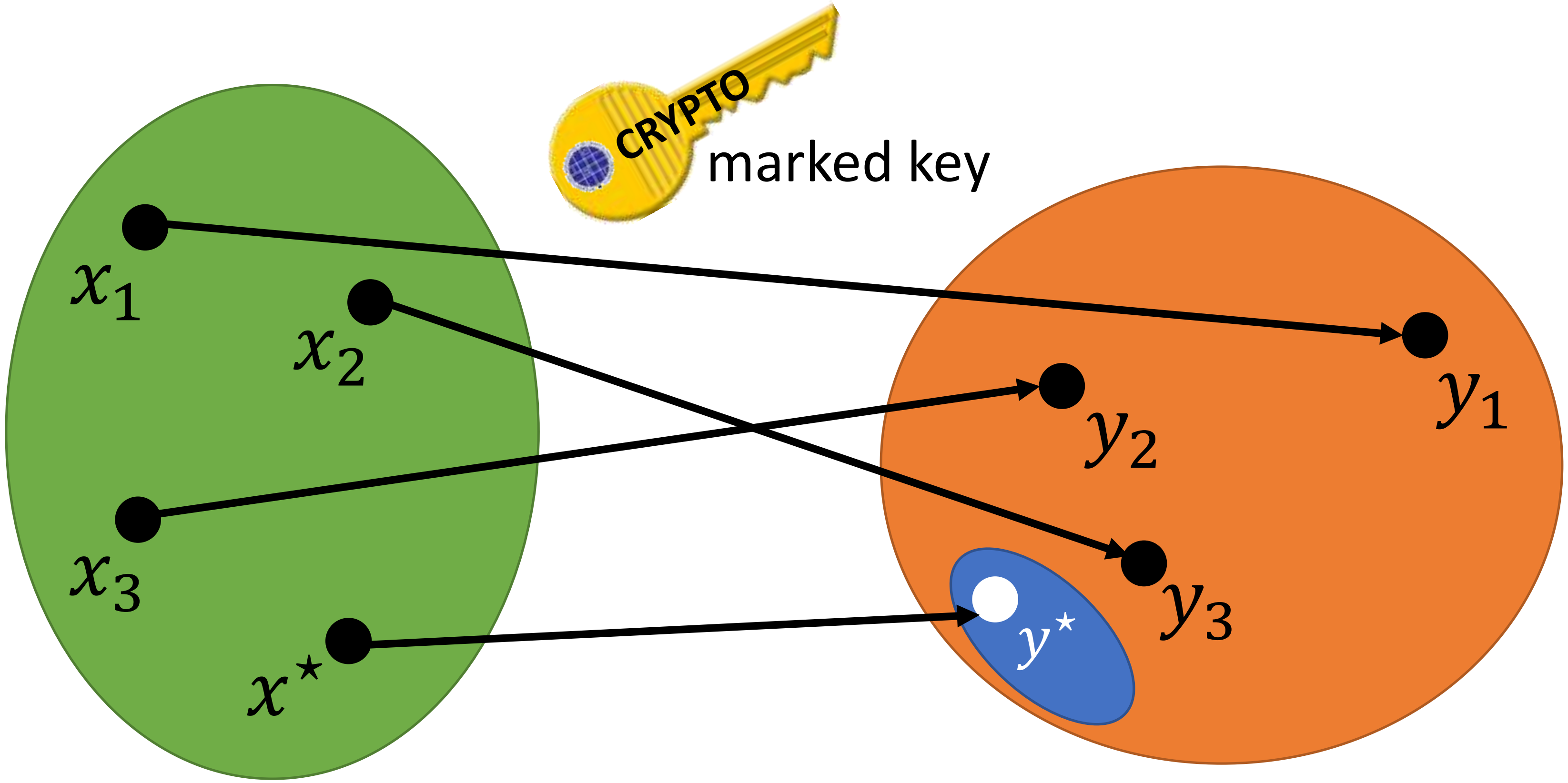
Watermarking secret key (wsk): test points $x_1, \ldots, x_d$ and testing key for private translucent PRF

Watermarking from Private Translucent PRFs



To mark a PRF key k, derive special point $x^\star$ and puncture k at $x^\star$; watermarked key is a program that evaluates using the punctured key

Watermarking from Private Translucent PRFs



To test whether a program C' is watermarked, derive test point $x^\star$ and check whether $C'(x^\star)$ is in the translucent set (using the testing key for the private translucent PRF)

Constructing Private Translucent PRFs

Learning with Errors (LWE) [Reg05]:

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

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

Homomorphic Matrix Encodings [BGGHNSVV14]

A way to encode $x \in \{0,1\}^\ell$ as a collection of LWE samples

take LWE matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell \in \mathbb{Z}_q^{n \times m}$ and a secret $\boldsymbol{s} \in \mathbb{Z}_q^n$:

Homomorphic Matrix Encodings [BGGHNSVV14]

LWE matrix associated with each input bit

$\boldsymbol{G}$ denotes a special "gadget" matrix

e … ction of LWE samples

tri … ^{m} and a secret $\boldsymbol{s} \in \mathbb{Z}_q^n$:

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1 \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$

encoding of x_1 with respect to $\boldsymbol{A}_1$

Homomorphic Matrix Encodings [BGGHNSVV14]

LWE matrix associated with each input bit

G denotes a special "gadget" matrix

e … ction of LWE samples

tri … ^{m} and a secret $\boldsymbol{s} \in \mathbb{Z}_q^n$:

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1 \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$

encoding of x_1 with respect to $\boldsymbol{A}_1$

$$\vdots$$

$$\boldsymbol{s}^T(\boldsymbol{A}_\ell + x_\ell \cdot \boldsymbol{G}) + \boldsymbol{e}_\ell$$

Homomorphic Matrix Encodings [BGGHNSVV14]

LWE matrix associated with each input bit

$\boldsymbol{G}$ denotes a special "gadget" matrix

e … ction of LWE samples

tri … m and a secret $\boldsymbol{s} \in \mathbb{Z}_q^n$:

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1 \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$

$$\vdots$$

$$\boldsymbol{s}^T(\boldsymbol{A}_\ell + x_\ell \cdot \boldsymbol{G}) + \boldsymbol{e}_\ell$$

Function of f and $\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell$

$$\boldsymbol{s}^T\left(\boldsymbol{A}_f + f(x) \cdot \boldsymbol{G}\right) + \text{noise}$$

Encodings support homomorphic operations

Puncturable PRFs from LWE [BV15]

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1 \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$
$$\vdots$$
$$\boldsymbol{s}^T(\boldsymbol{A}_\ell + x_\ell \cdot \boldsymbol{G}) + \boldsymbol{e}_\ell$$

$\Rightarrow$

$$\boldsymbol{s}^T\left(\boldsymbol{A}_f + f(x) \cdot \boldsymbol{G}\right) + \text{noise}$$

For any function f, two ways to (approximately) compute $\boldsymbol{s}^T\boldsymbol{A}_f$

- Directly given the LWE secret $\boldsymbol{s}$ and public matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell$
- Homomorphically given encodings $\boldsymbol{s}^T(\boldsymbol{A}_i + x_i \cdot \boldsymbol{G})$
  - Works as long as $f(x) = 0$

Puncturable PRFs from LWE [BV15]

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1 \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$

$$\vdots$$

$$\boldsymbol{s}^T(\boldsymbol{A}_\ell + x_\ell \cdot \boldsymbol{G}) + \boldsymbol{e}_\ell$$

$$\boldsymbol{s}^T\left(\boldsymbol{A}_f + f(x) \cdot \boldsymbol{G}\right) + \text{noise}$$

For puncturing at $x^\star$, let $f_x(x^\star) = \text{eq}(x, x^\star)$ be the equality function

- PRF evaluation at x with secret key $\boldsymbol{s}$: $\text{PRF}(\boldsymbol{s}, x) \coloneqq \left\lfloor \boldsymbol{s}^T \boldsymbol{A}_{f_x} \right\rceil_p$

Puncturable PRFs from LWE [BV15]

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1 \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$

$$\vdots$$

$$\boldsymbol{s}^T(\boldsymbol{A}_\ell + x_\ell \cdot \boldsymbol{G}) + \boldsymbol{e}_\ell$$

$$\boldsymbol{s}^T\left(\boldsymbol{A}_f + f(x) \cdot \boldsymbol{G}\right) + \text{noise}$$

To evaluate at x, homomorphically compute $\boldsymbol{A}_{f_x}$

For puncturing at $x^\star$, let $f_x(x^\star) = \text{eq}(x, x^\star)$ be the equality function

- PRF evaluation at x with secret key $\boldsymbol{s}$: $\text{PRF}(\boldsymbol{s}, x) := \left\lfloor \boldsymbol{s}^T \boldsymbol{A}_{f_x} \right\rceil_p$

Puncturable PRFs from LWE [BV15]

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1 \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$
$$\vdots$$
$$\boldsymbol{s}^T(\boldsymbol{A}_\ell + x_\ell \cdot \boldsymbol{G}) + \boldsymbol{e}_\ell$$

$$\Rightarrow \quad \boldsymbol{s}^T\left(\boldsymbol{A}_f + f(x) \cdot \boldsymbol{G}\right) + \text{noise}$$

For puncturing at $x^\star$, let $f_x(x^\star) = \text{eq}(x, x^\star)$ be the equality function

- PRF evaluation at x with secret key $\boldsymbol{s}$: $\text{PRF}(\boldsymbol{s}, x) \coloneqq \lfloor \boldsymbol{s}^T \boldsymbol{A}_{f_x} \rceil_p$
- Punctured key consists of encodings of bits of $x^\star$
  - Allows computing $\boldsymbol{s}^T\left(\boldsymbol{A}_{f_x} + \text{eq}(x, x^\star) \cdot \boldsymbol{G}\right) + \text{noise}$ for all x

Puncturable PRFs from LWE [BV15]

$$\mathbf{s}^T(\mathbf{A}_1 + x_1 \cdot \mathbf{G}) + \mathbf{e}_1$$
$$\vdots$$
$$\mathbf{s}^T(\mathbf{A}_\ell + x_\ell \cdot \mathbf{G}) + \mathbf{e}_\ell$$

$\Rightarrow$

$$\mathbf{s}^T\left(\mathbf{A}_f + f(x) \cdot \mathbf{G}\right) + \text{noise}$$

For puncturing at $x^\star$, let $f_x(x^\star) = \text{eq}(x, x^\star)$ be the equality function

- PRF evaluation at x with sec … $\mathbf{A}_{f_x}\rfloor_p$
- Punctured key consists of en … bits of x
  - Allows computing $\mathbf{s}^T\left(\mathbf{A}_{f_x} + \text{eq}(x, x^\star) \cdot \mathbf{G}\right) + \text{noise}$ for all x

Enables evaluation at all x except when $x = x^\star$

Privately Puncturable PRFs [BKM17]

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

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1^\star \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$
$$\vdots$$
$$\boldsymbol{s}^T(\boldsymbol{A}_\ell + x_\ell^\star \cdot \boldsymbol{G}) + \boldsymbol{e}_\ell$$

Evaluating PRF using punctured key requires knowledge of $x^\star$

Key idea in [BKM17]: encrypt the punctured point using an FHE scheme and homomorphically evaluate the equality function

Evaluating using punctured key outputs

$$\boldsymbol{s}^T \boldsymbol{A}_{f_x} + \boldsymbol{s}^T \left(\frac{q}{2} \cdot \mathrm{eq}(x, x^\star) + e \right) \cdot \boldsymbol{G} + \boldsymbol{e}'$$

Privately Puncturable PRFs [BKM17]

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

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1^\star \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$
$$\vdots$$
$$\boldsymbol{s}^T(\boldsymbol{A}_\ell + x_\ell^\star \cdot \boldsymbol{G}) + \boldsymbol{e}_\ell$$

Key idea in [BKM17]: encrypt the punctured point using an FHE scheme and homomorphically evaluate the equality function

Evaluating PRF using punctured key requir

Value after FHE evaluation and decryption

Evaluating usin

$$\boldsymbol{s}^T \boldsymbol{A}_{f_x} + \boldsymbol{s}^T \left(\frac{q}{2} \cdot \mathrm{eq}(x, x^\star) + e\right) \cdot \boldsymbol{G} + \boldsymbol{e}'$$

Privately Puncturable PRFs [BKM17]

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

$$\boldsymbol{s}^T(\boldsymbol{A}_1 + x_1^\star \cdot \boldsymbol{G}) + \boldsymbol{e}_1$$

$$\vdots$$

$$\boldsymbol{s}^T(\boldsymbol{A}_\ell + x_\ell^\star \cdot \boldsymbol{G}) + \boldsymbol{e}_\ell$$

Evaluating PRF using punctured key requires knowledge of $x^\star$

Key idea in [BKM17]: encrypt the punctured point using an FHE scheme and homomorphically evaluate the equality function

Evaluating using punctured key outputs

$$\boldsymbol{s}^T \boldsymbol{A}_{f_x} + \boxed{\boldsymbol{s}^T \left(\frac{q}{2} \cdot \text{eq}(x, x^\star) + e\right) \cdot \boldsymbol{G}} + \boldsymbol{e}'$$

Rounding away the FHE error: small if $\boldsymbol{s}$ is short and we restrict to low-norm columns of $\boldsymbol{G}$

Private Translucent PRFs

Change real PRF evaluation: $\mathrm{PRF}(\boldsymbol{s}, x) := \left\lfloor \boldsymbol{s}^T \boldsymbol{A}_{f_x} \boldsymbol{G}^{-1}(\boldsymbol{D}) \right\rceil_p$

Evaluating using a punctured key yields

$$\boldsymbol{s}^T\left(\boldsymbol{A}_{f_x} + (w \cdot \mathrm{eq}(x, x^\star) + e) \cdot \boldsymbol{G}\right)\boldsymbol{G}^{-1}(\boldsymbol{D}) + \text{noise}$$

Use a different scaling factor in FHE scheme (instead of $q/2$)

Private Translucent PRFs

Change real PRF evaluation: $\mathrm{PRF}(\boldsymbol{s}, x) \coloneqq \lfloor \boldsymbol{s}^T \boldsymbol{A}_{f_x} \boldsymbol{G}^{-1}(\boldsymbol{D}) \rceil_p$

Evaluating using a punctured key yields

$$\boldsymbol{s}^T \left(\boldsymbol{A}_{f_x} \boldsymbol{G}^{-1}(\boldsymbol{D}) + (w \cdot \mathrm{eq}(x, x^\star) + e) \cdot \boldsymbol{D} \right) + \mathrm{noise}$$

When $x = x^\star$, this becomes $\left\lfloor \boldsymbol{s}^T \left(\boldsymbol{A}_{f_x} \boldsymbol{G}^{-1}(\boldsymbol{D}) + w\boldsymbol{D} \right) \right\rceil_p$

Can choose w (part of the punctured key) to tweak the value at punctured point

Private Translucent PRFs

If $x = x^\star$, evaluating using the punctured key yields $\left\lfloor \boldsymbol{s}^T\left(\boldsymbol{A}_{f_x}\boldsymbol{G}^{-1}(\boldsymbol{D}) + w\boldsymbol{D}\right)\right\rceil_p$

Key idea: sum up multiple evaluations with different multiples w_i and $\boldsymbol{D}_i$ yields

$$\left\lfloor \sum_i \boldsymbol{s}^T\left(\boldsymbol{A}_{f_x}\boldsymbol{G}^{-1}(\boldsymbol{D}) + w_i\boldsymbol{D}_i\right)\right\rceil_p = \left\lfloor \boldsymbol{s}^T\boldsymbol{W}\right\rceil_p$$

Output at punctured point is an LWE sample with respect to $\boldsymbol{W}$ (fixed public matrix) – critical for implementing a translucent set

Private Translucent PRFs

If $x = x^\star$, evaluating using the punctured key yields $\left\lfloor \boldsymbol{s}^T \left(\boldsymbol{A}_{f_x} \boldsymbol{G}^{-1}(\boldsymbol{D}) + w\boldsymbol{D} \right) \right\rceil_p$

Key idea: sum up multiple evaluations with different multiples w_i and $\boldsymbol{D}_i$ yields

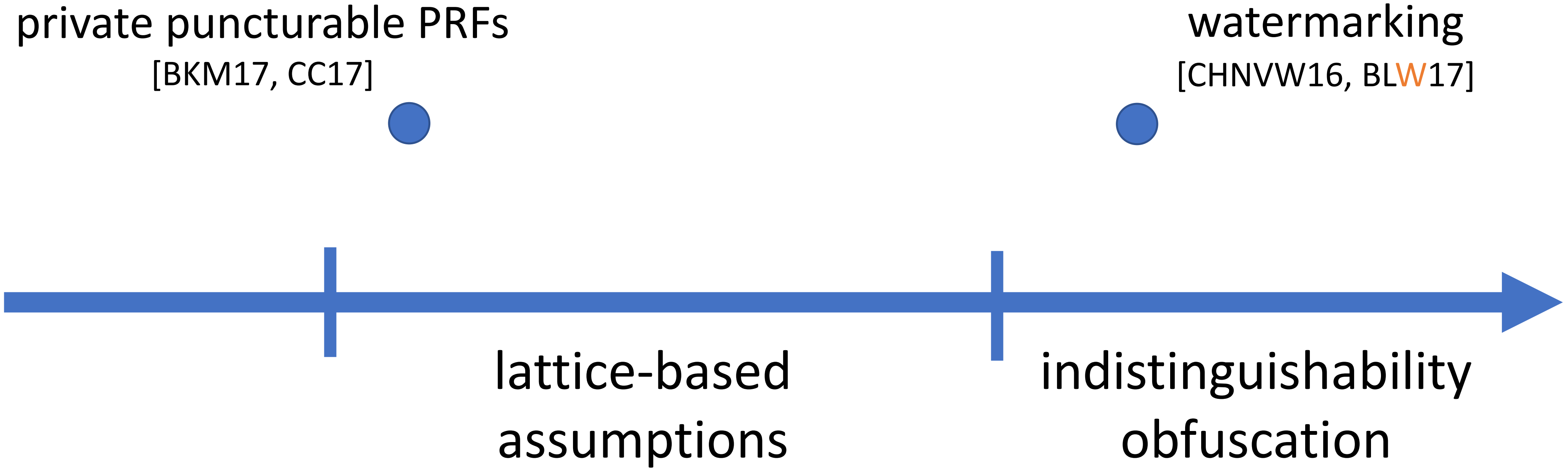
$$\left\lfloor \sum_i \boldsymbol{s}^T \left(\boldsymbol{A}_{f_x} \boldsymbol{G}^{-1}(\boldsymbol{D}) + w_i \boldsymbol{D}_i \right) \right\rceil_p = \left\lfloor \boldsymbol{s}^T \boldsymbol{W} \right\rceil_p$$

Testing key is a short vector $\boldsymbol{z}$ where $\boldsymbol{W}\boldsymbol{z} = 0$:

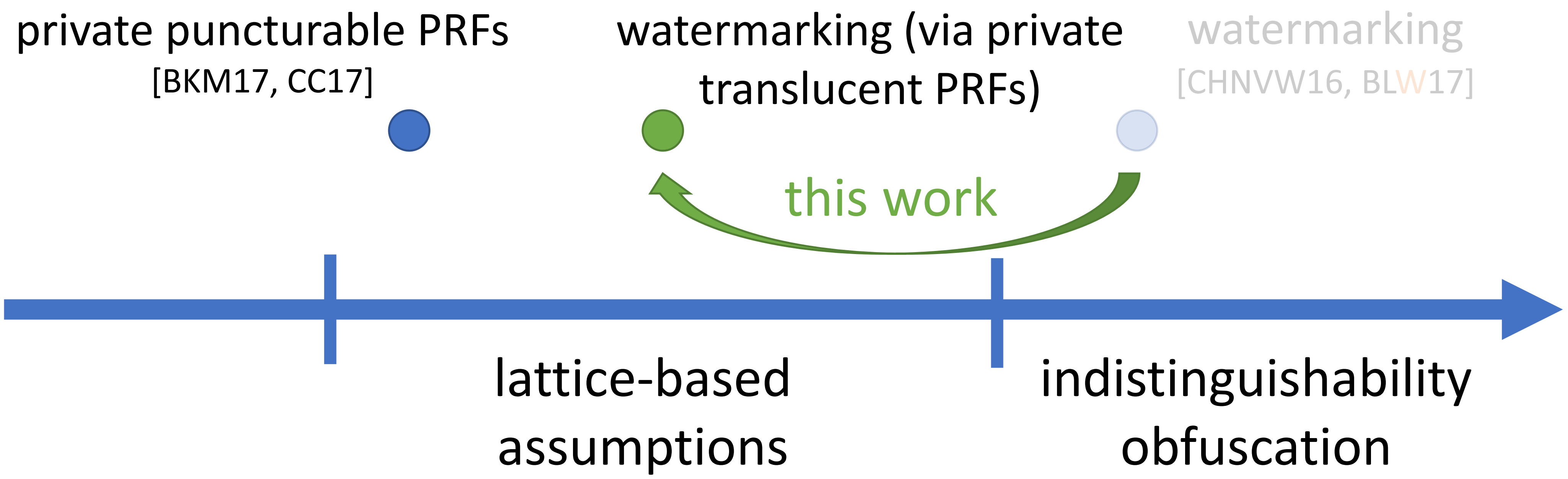
$$\left\langle \left\lfloor \boldsymbol{s}^T \boldsymbol{W} \right\rceil_p, \boldsymbol{z} \right\rangle \approx \left\lfloor \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{z} \right\rceil_p = 0$$

[See paper for details]

Conclusions



Conclusions



Open Problems

Publicly-verifiable watermarking without obfuscation?

- Current best construction relies on iO [CHNVW16]

Additional applications of private translucent PRFs?

Thank you!

http://eprint.iacr.org/2017/380