On Instantiating Unleveled Fully-Homomorphic Signatures from Falsifiable Assumptions

Romain Gay IBM Research, Zurich

Bogdan Ursu Linea, Consensys

Fully Homomorphic Encryption

$\mathsf{ct}_1 = \mathsf{FHE.Enc}(\mathsf{pk}, m_1)$ $\mathsf{ct}_2 = \mathsf{FHE.Enc}(\mathsf{pk}, m_2)$

$\mathsf{Eval}_+(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)$

$\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, m_1 + m_2)$

Linea

Fully Homomorphic Encryption

$\mathsf{ct}_1 = \mathsf{FHE.Enc}(\mathsf{pk}, m_1)$

$\mathsf{ct}_2 = \mathsf{FHE.Enc}(\mathsf{pk}, m_2)$

$\mathsf{Eval}_+(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)$ → $\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, m_1 + m_2)$

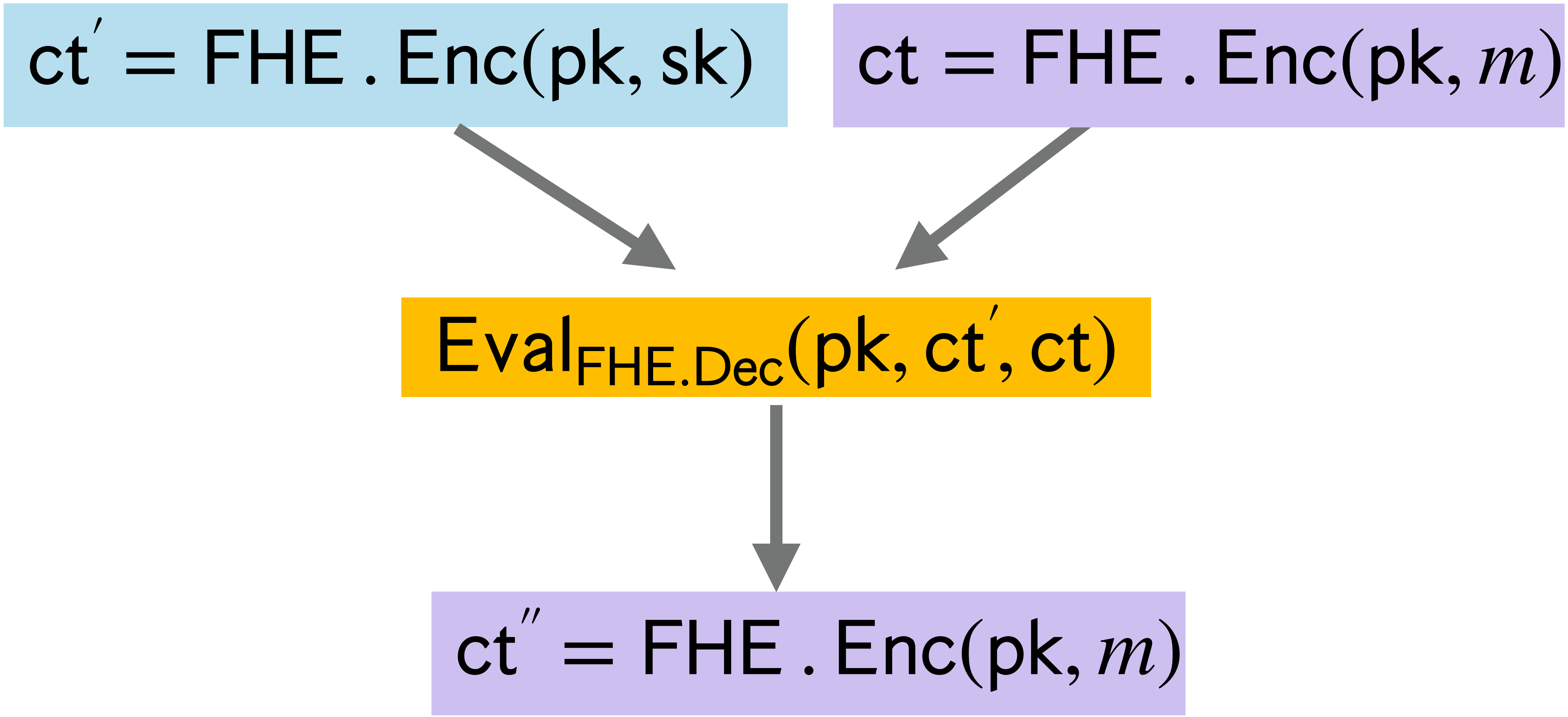
$\mathsf{Eval}_\times(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)$ → $\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, m_1 \cdot m_2)$

$\mathsf{Eval}_\times(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)$ → $\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, C(m_1, m_2))$

Correctness preserved up to a maximal noise bound B_{noise}.

Linea

Bootstrapping for FHE



Bootstrapping for FHE

$\mathsf{ct}' = \mathsf{FHE.Enc}(\mathsf{pk}, \mathsf{sk})$

$\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, m)$

$\mathsf{Eval}_{\mathsf{FHE.Dec}}(\mathsf{pk}, \mathsf{ct}', \mathsf{ct})$

$\mathsf{ct}'' = \mathsf{FHE.Enc}(\mathsf{pk}, m)$ refresh the noise in ct'' (due to the rounding step in Dec)

Fully Homomorphic Signatures (FHS)

$\sigma_1 = \mathsf{FHS.Sign}(\mathsf{sk}, m_1)$

$\sigma_2 = \mathsf{FHS.Sign}(\mathsf{sk}, m_2)$

$\mathsf{Eval}_+(\mathsf{vk}, \sigma_1, \sigma_2)$

$\sigma = \mathsf{FHS.Sign}(\mathsf{sk}, m_1 + m_2)$

Fully Homomorphic Signatures (FHS)

$\sigma_1 = \mathsf{FHS.Sign}(\mathsf{sk}, m_1)$

$\sigma_2 = \mathsf{FHS.Sign}(\mathsf{sk}, m_2)$

$\mathsf{Eval}_+(\mathsf{vk}, \sigma_1, \sigma_2)$ → $\sigma = \mathsf{FHS.Sign}(\mathsf{sk}, m_1 + m_2)$

$\mathsf{Eval}_\times(\mathsf{vk}, \sigma_1, \sigma_2)$ → $\sigma = \mathsf{FHS.Sign}(\mathsf{sk}, m_1 \cdot m_2)$

$\mathsf{Eval}_C(\mathsf{vk}, \sigma_1, \sigma_2)$ → $\sigma_C = \mathsf{FHS.Sign}(\mathsf{sk}, C(m_1, m_2))$

Linea

Fully Homomorphic Signatures (FHS)

$\sigma_1 = \mathsf{FHS.Sign}(\mathsf{sk}, m_1)$ $\sigma_2 = \mathsf{FHS.Sign}(\mathsf{sk}, m_2)$

$\mathsf{Eval}_+(\mathsf{vk}, \sigma_1, \sigma_2)$ $\mathsf{Eval}_\times(\mathsf{vk}, \sigma_1, \sigma_2)$ $\mathsf{Eval}_C(\mathsf{vk}, \sigma_1, \sigma_2)$

$\sigma = \mathsf{FHS.Sign}(\mathsf{sk}, m_1 + m_2)$ $\sigma = \mathsf{FHS.Sign}(\mathsf{sk}, m_1 \cdot m_2)$ $\sigma_C = \mathsf{FHS.Sign}(\mathsf{sk}, C(m_1, m_2))$

[GVW15] is an FHS based on lattices (SIS) which is levelled.

Correctness preserved up to a maximal noise bound B_{noise}.

Bootstrapping for FHS?

$\mathsf{ct}' = \mathsf{FHE.Enc}(\mathsf{pk}, \mathsf{sk})$

$\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, x)$

$\mathsf{Eval}_{\mathsf{FHE.Dec}}(\mathsf{pk}, \mathsf{ct}', \mathsf{ct})$

$\mathsf{ct}'' = \mathsf{FHE.Enc}(\mathsf{pk}, x)$

ct'' has its noise refreshed

Bootstrapping for FHS?

$\mathsf{ct}' = \mathsf{FHE.Enc}(\mathsf{pk}, \mathsf{sk})$

$\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, x)$

$\mathsf{Eval}_{\mathsf{FHE.Dec}}(\mathsf{pk}, \mathsf{ct}', \mathsf{ct})$

$\mathsf{ct}'' = \mathsf{FHE.Enc}(\mathsf{pk}, x)$

ct'' has its noise refreshed

$\sigma' = \mathsf{FHS.Sign}(\mathsf{sk}, \ ? \ )$

$\sigma = \mathsf{FHS.Sign}(\mathsf{sk}, m)$

$\mathsf{Eval}_{?}(\mathsf{vk}, \ ? \ , \sigma)$

No bootstrapping equivalent for FHS.

FHS—The Model

Key Generation: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

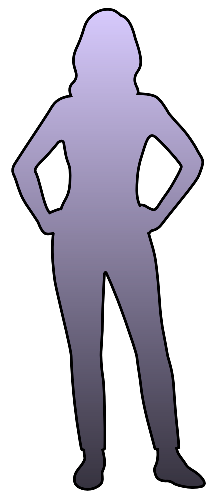
Precomputation of a specific vk_C for a circuit C as $\mathsf{vk}_C \leftarrow \mathsf{Process}_{\mathsf{vk}}(C)$.

FHS — The Model

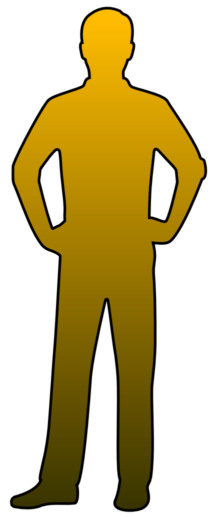
Key Generation: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Precomputation of a specific vk_C for a circuit C as $\mathsf{vk}_C \leftarrow \mathsf{Process}_{\mathsf{vk}}(C)$.

Alice



Bob

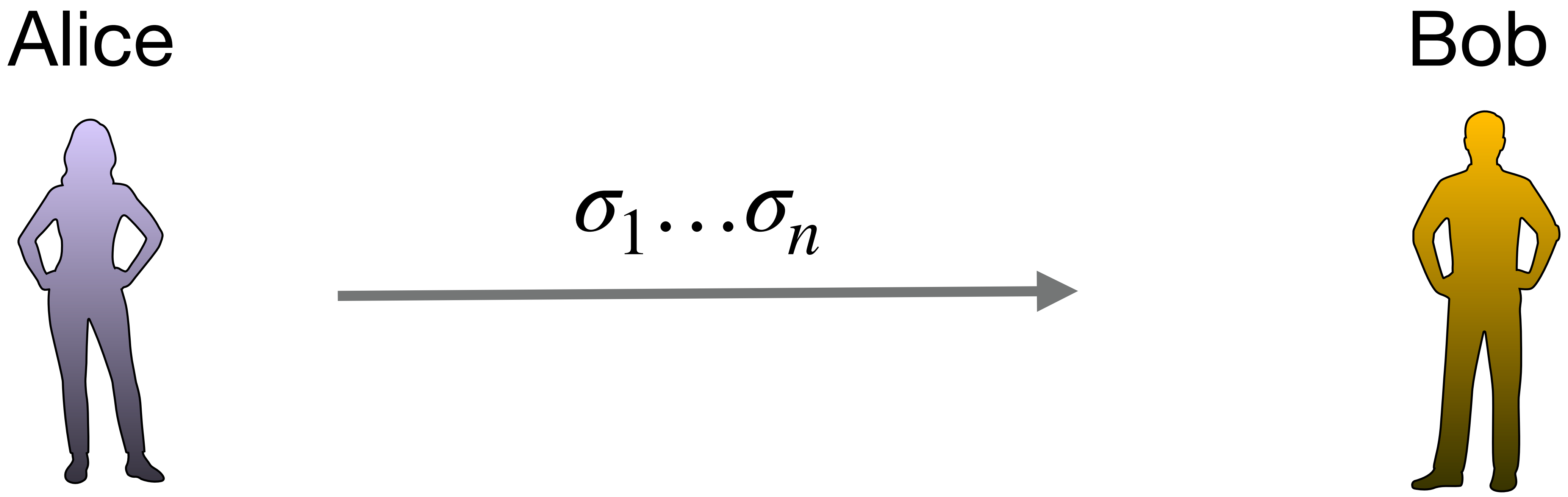


FHS—The Model

Key Generation: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Precomputation of a specific vk_C for a circuit C as $\mathsf{vk}_C \leftarrow \mathsf{Process}_{\mathsf{vk}}(C)$.

$(\sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_1 \ldots m_n)$



Linea

FHS—The Model

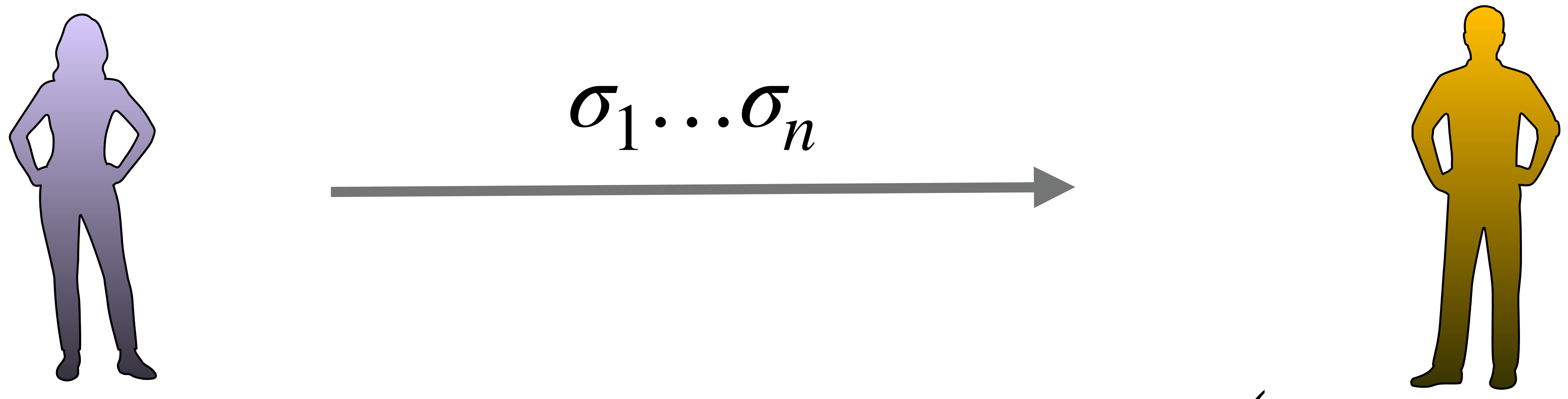
Key Generation: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Precomputation of a specific vk_C for a circuit C as $\mathsf{vk}_C \leftarrow \mathsf{Process}_{\mathsf{vk}}(C)$.

Alice

Bob

$(\sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_1 \ldots m_n)$



$\sigma_1 \ldots \sigma_n$

$$\sigma^* = \mathsf{Eval}_{\mathsf{vk}}\Big(C, (m_1, \sigma_1) \ldots (m_n, \sigma_n)\Big)$$

$$\mathsf{Verify}_{\mathsf{vk}}\Big(\mathsf{vk}_C, y, \sigma^*\Big) \stackrel{?}{=} 1$$

when $y = C(m_1 \ldots m_n)$

FHS—The Model

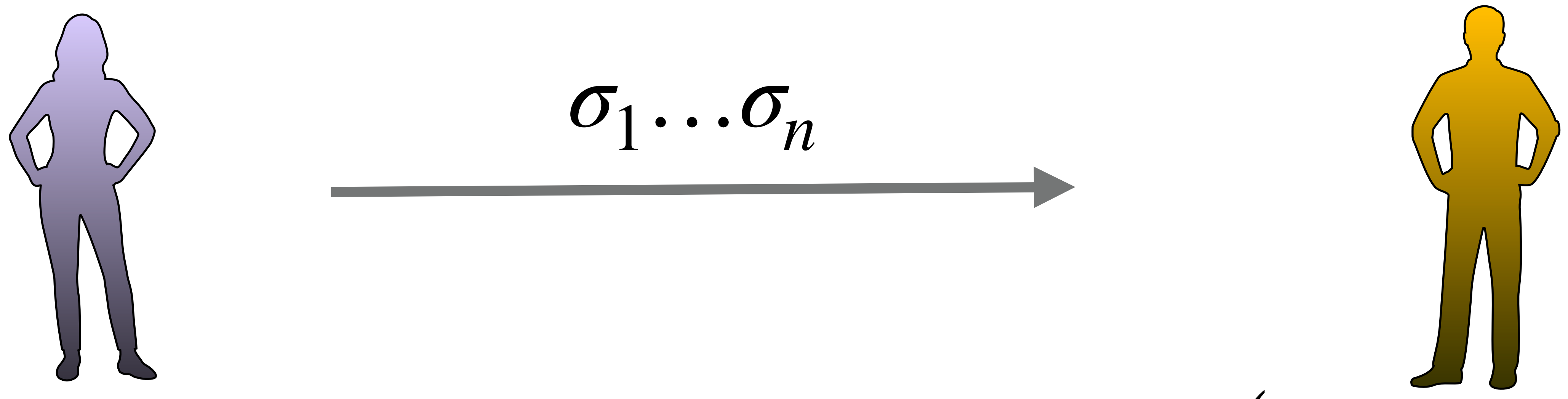
Key Generation: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Precomputation of a specific vk_C for a circuit C as $\mathsf{vk}_C \leftarrow \mathsf{Process}_{\mathsf{vk}}(C)$.

Alice

Bob

$(\sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_1 \ldots m_n)$



$\sigma_1 \ldots \sigma_n$

$$\sigma^* = \mathsf{Eval}_{\mathsf{vk}}\Big(C, (m_1, \sigma_1) \ldots (m_n, \sigma_n)\Big)$$

$$\mathsf{Verify}_{\mathsf{vk}}\Big(\mathsf{vk}_C, y, \sigma^*\Big) \stackrel{?}{=} 1$$

when $y = C(m_1 \ldots m_n)$

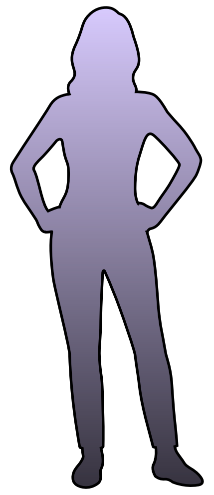
Succinctness: size of evaluated σ^* should not depend on n.

Succinctness: size of circuit verification key vk_C should not depend on $|C|$.

Precomputation of vk_C can be done for the circuits of interest depending on the application.

FHS—Composability

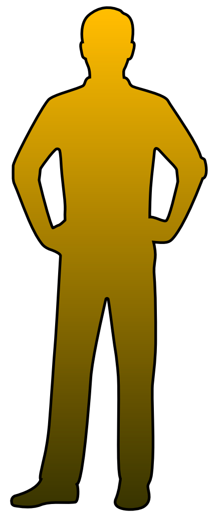
Alice



$(\sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_1 \ldots m_n)$

$\sigma_1 \ldots \sigma_n$ $\longrightarrow$

Bob



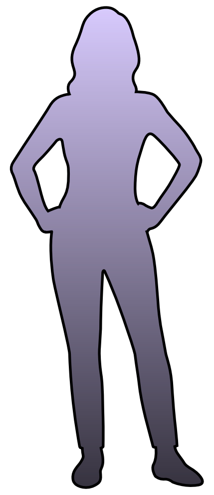
$\sigma^* = \mathsf{Eval}_{\mathsf{vk}}\Big(C, (m_1, \sigma_1) \ldots (m_n, \sigma_n)\Big)$

$\mathsf{Verify}_{\mathsf{vk}}\Big(\mathsf{vk}_C, y, \sigma\Big) \stackrel{?}{=} 1$

FHS—Composability

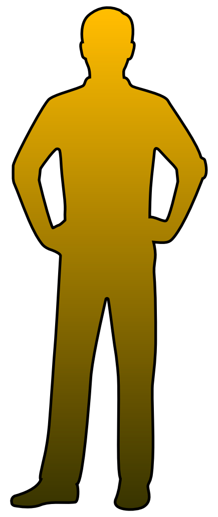
Alice

$(\sigma_1 \dots \sigma_n) \leftarrow \mathsf{Sign}_{sk}(m_1 \dots m_n)$



$\sigma_1 \dots \sigma_n$

Bob



$\sigma^* = \mathsf{Eval}_{vk}\Big(C, (m_1, \sigma_1) \dots (m_n, \sigma_n)\Big)$

$\mathsf{Verify}_{vk}\Big(\mathsf{vk}_C, y, \sigma\Big) \stackrel{?}{=} 1$

k circuits $C_1 \dots C_k$

$\sigma_{C_1} = \mathsf{FHS.Sign}(\mathsf{sk}, C_1(m_1 \dots m_n))$ $\quad \dots \quad$ $\sigma_{C_k} = \mathsf{FHS.Sign}(\mathsf{sk}, C_k(m_1 \dots m_n))$

$\mathsf{Eval}_F(\sigma_1 \dots \sigma_k)$

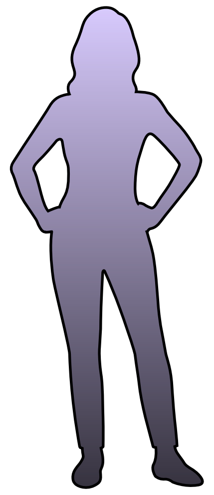
$\sigma_G = \mathsf{FHS.Sign}(\mathsf{sk}, G(m_1 \dots m_n))$

Linea

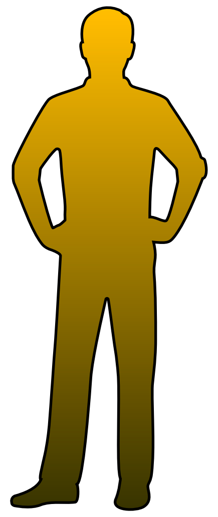
FHS — Composability

Alice

$(\sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_1 \ldots m_n)$



$\sigma_1 \ldots \sigma_n$



Bob

$\sigma^* = \mathsf{Eval}_{\mathsf{vk}}\Big(C, (m_1, \sigma_1) \ldots (m_n, \sigma_n)\Big)$

$\mathsf{Verify}_{\mathsf{vk}}\Big(\mathsf{vk}_C, y, \sigma\Big) \stackrel{?}{=} 1$

$\sigma_{C_1} = \mathsf{FHS.Sign}(\mathsf{sk}, C_1(m_1 \ldots m_n))$ ... $\sigma_{C_k} = \mathsf{FHS.Sign}(\mathsf{sk}, C_k(m_1 \ldots m_n))$

$\mathsf{Eval}_F(\sigma_1 \ldots \sigma_k)$

$\sigma_G = \mathsf{FHS.Sign}(\mathsf{sk}, G(m_1 \ldots m_n))$

$G(m_1 \ldots m_n) \stackrel{\text{def}}{=} F\Big(\underbrace{C_1(m_1 \ldots m_n) \ \ldots \ C_k(m_1 \ldots m_n)}_{k \text{ inputs to } F}\Big)$

Unbounded number of composable operations.

Relative to the original $m_1 \ldots m_n$.

Linea

FHS—Security

Adversary

Challenger

vk, sk ←

$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda).$

Linea

FHS — Security


Adversary

Challenger


vk, sk

$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

$m_1 \ldots m_n$

$(\sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_1 \ldots m_n)$

FHS — Security

Adversary

Challenger

vk, sk ← $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

$m_1 \ldots m_n$ → $(\sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_1 \ldots m_n)$

← $\sigma_1 \ldots \sigma_n$

(y^*, C^*, σ^*) →

FHS — Security

Adversary

Challenger

vk, sk ←

(vk, sk) ← KeyGen(1^λ).

$m_1 \ldots m_n$ →

$(\sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_1 \ldots m_n)$

$\sigma_1 \ldots \sigma_n$ ←

(y^*, C^*, σ^*) →

Adversary wins if $\mathsf{Verify}_{\mathsf{vk}}(C^*, y^*, \sigma^*) = 1$ but $y^* \neq C^*(m_1 \ldots m_n)$.

Unforgeability is with respect to the original signed messages $m_1 \ldots m_n$.

Linea

FHS—Security

Adversary

Challenger

vk, sk ←

$(\text{vk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda)$.

$m_1 \ldots m_n$ →

$(\sigma_1 \ldots \sigma_n) \leftarrow \text{Sign}_{\text{sk}}(m_1 \ldots m_n)$

$\sigma_1 \ldots \sigma_n$ ←

(y^*, C^*, σ^*) →

Is the bound n a limitation?

Adversary wins if $\text{Verify}_{\text{vk}}(C^*, y^*, \sigma^*) = 1$ but $y^* \neq C^*(m_1 \ldots m_n)$.

Unforgeability is with respect to the original signed messages $m_1 \ldots m_n$.

Linea

FHS—Labelled Model (Multi-Dataset)

Key Generation: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

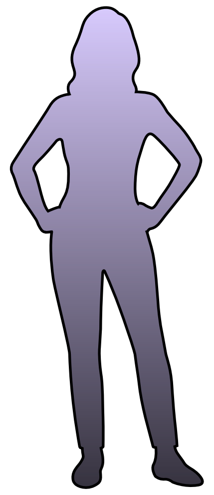
Precomputation of a specific vk_C for a circuit C as $\mathsf{vk}_C \leftarrow \mathsf{Process}_{\mathsf{vk}}(C)$.

FHS — Labelled Model (Multi-Dataset)

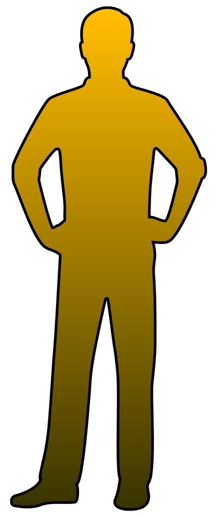
Key Generation: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Precomputation of a specific vk_C for a circuit C as $\mathsf{vk}_C \leftarrow \mathsf{Process}_{\mathsf{vk}}(C)$.

Alice



Bob



$$(\sigma_\tau, \sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}\Big((m_1 \ldots m_n), \tau\Big)$$

$\sigma_\tau, \sigma_1 \ldots \sigma_n$ →

$$\sigma^* = \mathsf{Eval}_{\mathsf{vk}}\Big(C, \sigma_\tau, (m_1, \sigma_1) \ldots (m_n, \sigma_n)\Big)$$

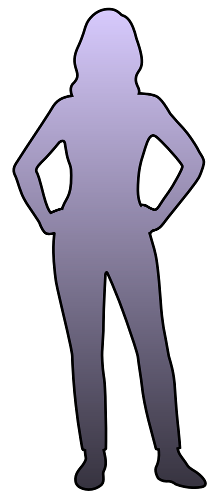
$$\mathsf{Verify}_{\mathsf{vk}}\Big(\mathsf{vk}_C, y, \tau, \sigma_\tau, \sigma^*\Big) \stackrel{?}{=} 1$$

FHS—Labelled Model (Multi-Dataset)

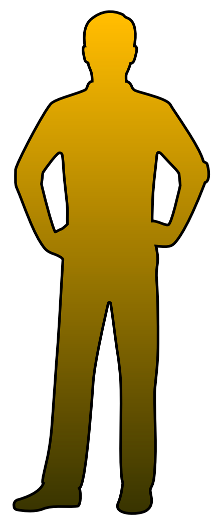
Key Generation: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Precomputation of a specific vk_C for a circuit C as $\mathsf{vk}_C \leftarrow \mathsf{Process}_{\mathsf{vk}}(C)$.

Alice



Bob



$(\sigma_\tau, \sigma_1 \ldots \sigma_n) \leftarrow \mathsf{Sign}_{\mathsf{sk}}\Big((m_1 \ldots m_n), \tau\Big)$

$\sigma_\tau, \sigma_1 \ldots \sigma_n$ →

$\sigma^* = \mathsf{Eval}_{\mathsf{vk}}\Big(C, \sigma_\tau, (m_1, \sigma_1) \ldots (m_n, \sigma_n)\Big)$

$\mathsf{Verify}_{\mathsf{vk}}\Big(\mathsf{vk}_C, y, \tau, \sigma_\tau, \sigma^*\Big) \stackrel{?}{=} 1$

Arbitrary labels $\tau \in \{0,1\}^*$.

Single-data to multi-data can be achieved using a generic transformation due to [GVW15].

Bound n can also be removed using ROM techniques due to [GVW15].

Linea

Previous Work

Reference	Setting	Limitations	Depth	Assumptions
[GW13]	MACs	security up to O(log(n)) verification queries.	unbounded	FHE scheme with ω(log(n)) random bits.
[GVW15]	signatures	levelled scheme, bounded number of messages	bounded	SIS
[GVW15]	signatures	levelled scheme	bounded	SIS + ROM
[BCFL23]	signatures	bounded circuit width	unbounded	pairings or new lattice assumptions (extension of SIS)
Snark-based	signatures		unbounded	knowledge assumptions or ROM
This work	signatures	bounded number of messages	unbounded	IO + one-way functions + FHE + NIZK
This work	signatures	unbounded number of messages	unbounded	IO + one-way functions + FHE + NIZK + ROM

Linea

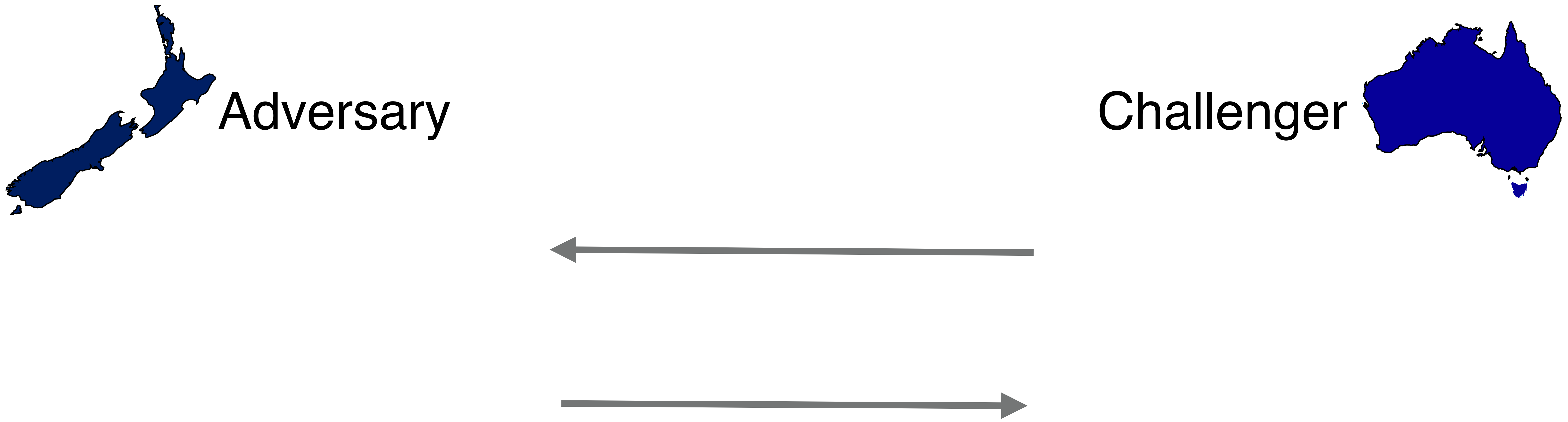
Previous Work

Reference	Setting	Limitations	Depth	Assumptions
[GW13]	MACs	security up to O(log(n)) verification queries.	unbounded	FHE scheme with ω(log(n)) random bits.
[GVW15]	signatures	levelled scheme, bounded number of messages	bounded	SIS
[GVW15]	signatures	levelled scheme	bounded	SIS + ROM
[BCFL23]	signatures	bounded circuit width	unbounded	pairings or new lattice assumptions (extension of SIS)
Snark-based	signatures		unbounded	knowledge assumptions or ROM
This work	signatures	bounded number of messages	unbounded	IO + one-way functions + FHE + NIZK
This work	signatures	unbounded number of messages	unbounded	IO + one-way functions + FHE + NIZK + ROM

Our goal: fully homomorphic signatures for unbounded-depth circuits in the standard model.

Falsifiable assumptions.

Falsifiable Assumptions



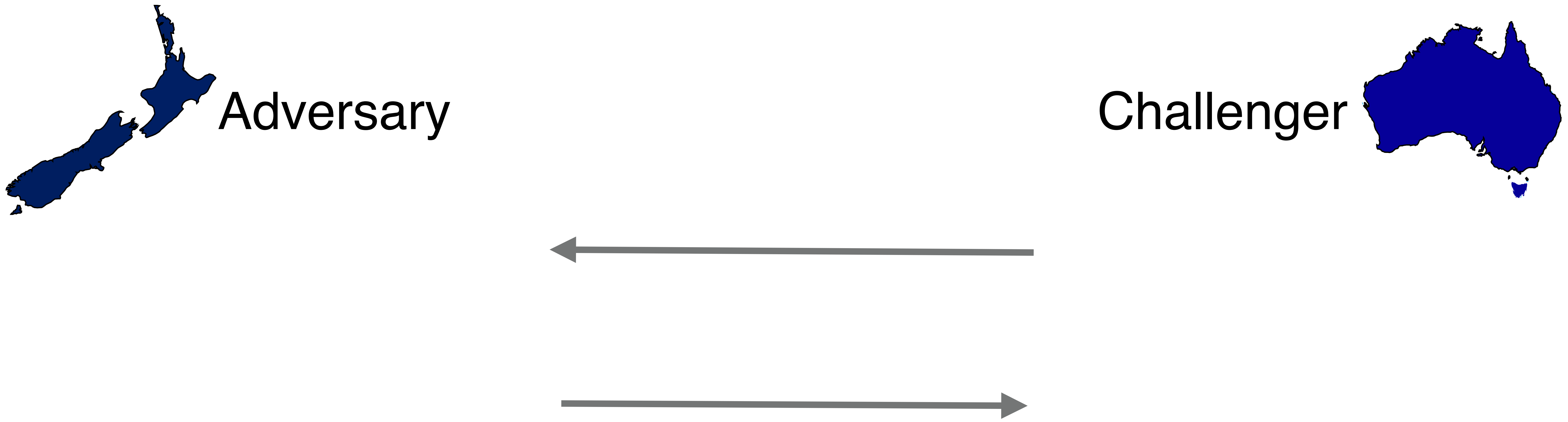
Condition 1

Assumption can be modelled as an interactive game between the adversary and the challenger.

Condition 2

Challenger can decide efficiently whether the adversary has won the game.

Falsifiable Assumptions



Condition 1

Assumption can be modelled as an interactive game between the adversary and the challenger.

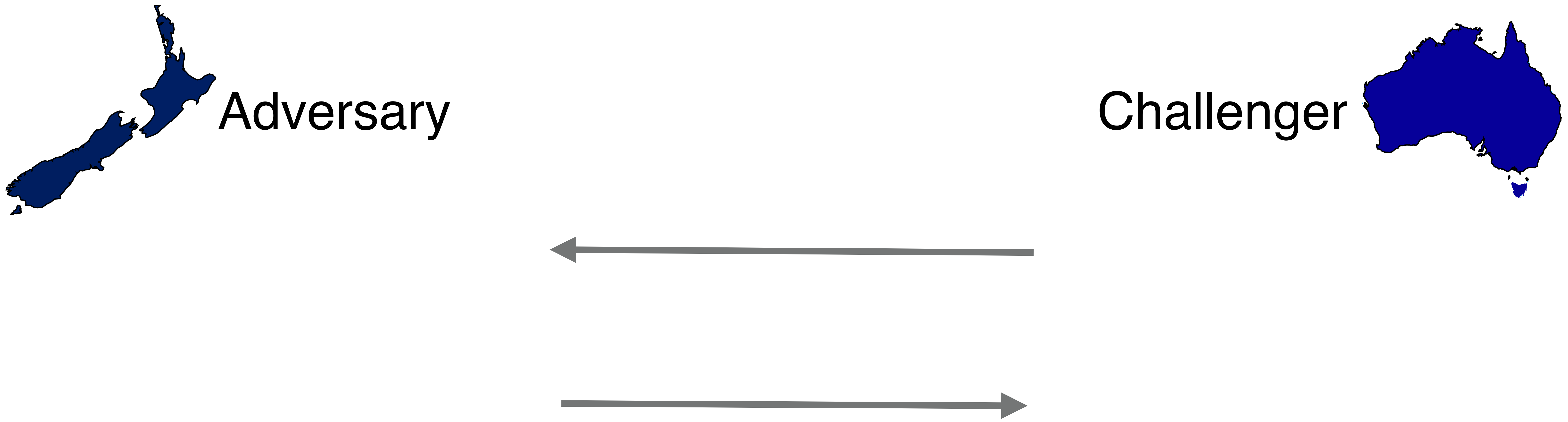
Condition 2

Challenger can decide efficiently whether the adversary has won the game.

The adversary winning probability cannot be smaller than $1/\text{poly}$.

Linea

Falsifiable Assumptions



Condition 1

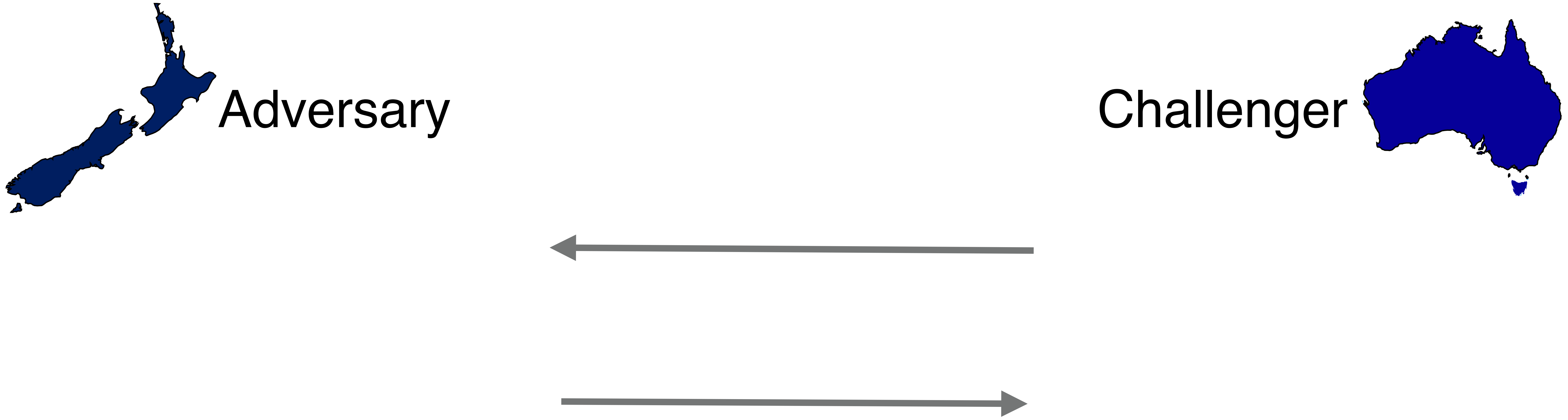
Assumption can be modelled as an interactive game between the adversary and the challenger.

Condition 2

Challenger can decide efficiently whether the adversary has won the game.

Falsifiable assumptions can be tested by exhibiting an attack against the assumption.

Falsifiable Assumptions



Condition 1

Assumption can be modelled as an interactive game between the adversary and the challenger.

Condition 2

Challenger can decide efficiently whether the adversary has won the game.

Example of unfalsifiable assumptions: knowledge-assumptions (knowledge of discrete log)

Building Block: Indistinguishability Obfuscation (iO)

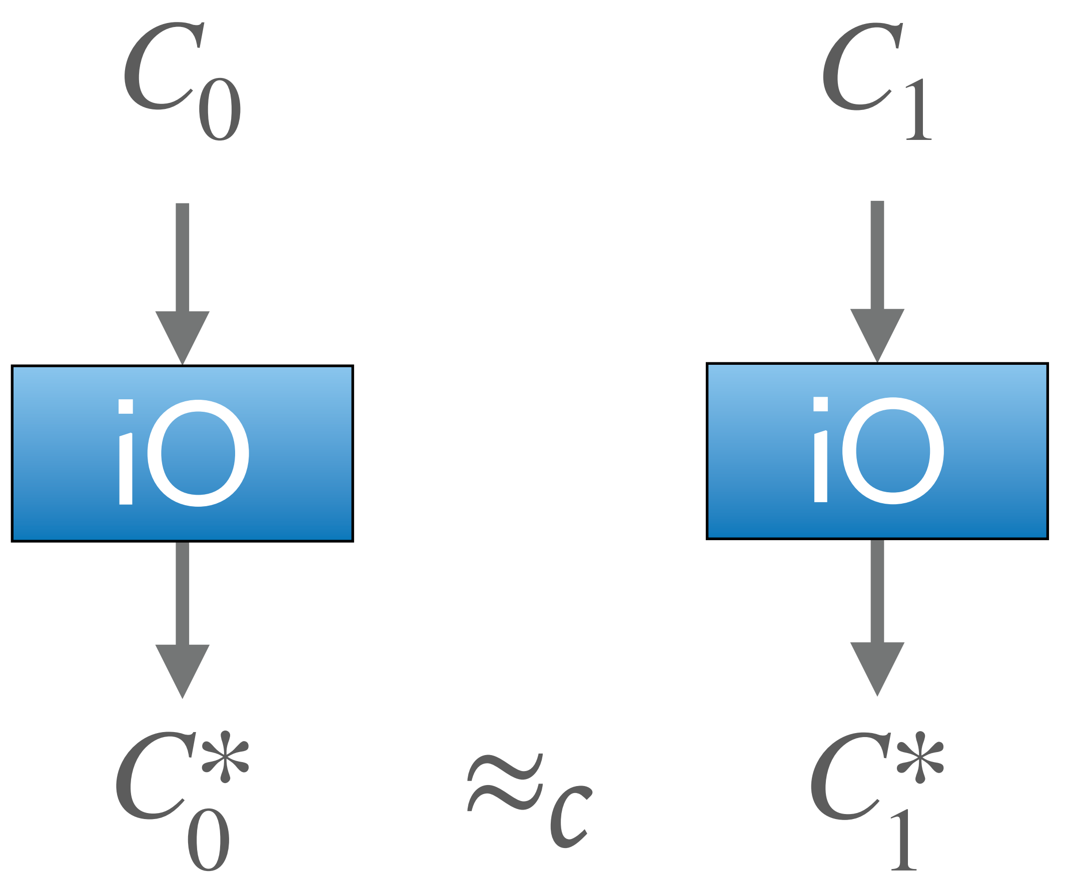
Obfuscation: functionality of a program is hidden.

but it can still be executed.

Indistinguishability Obfuscation (iO)

iO — an obfuscator of circuits.

Works on any circuits C_0 and C_1, such that $C_0(x) = C_1(x)$ for all inputs x.



The obfuscations C_0^* and C_1^* are computationally indistinguishable

Indistinguishability Obfuscation (iO)

iO — an obfuscator of circuits.

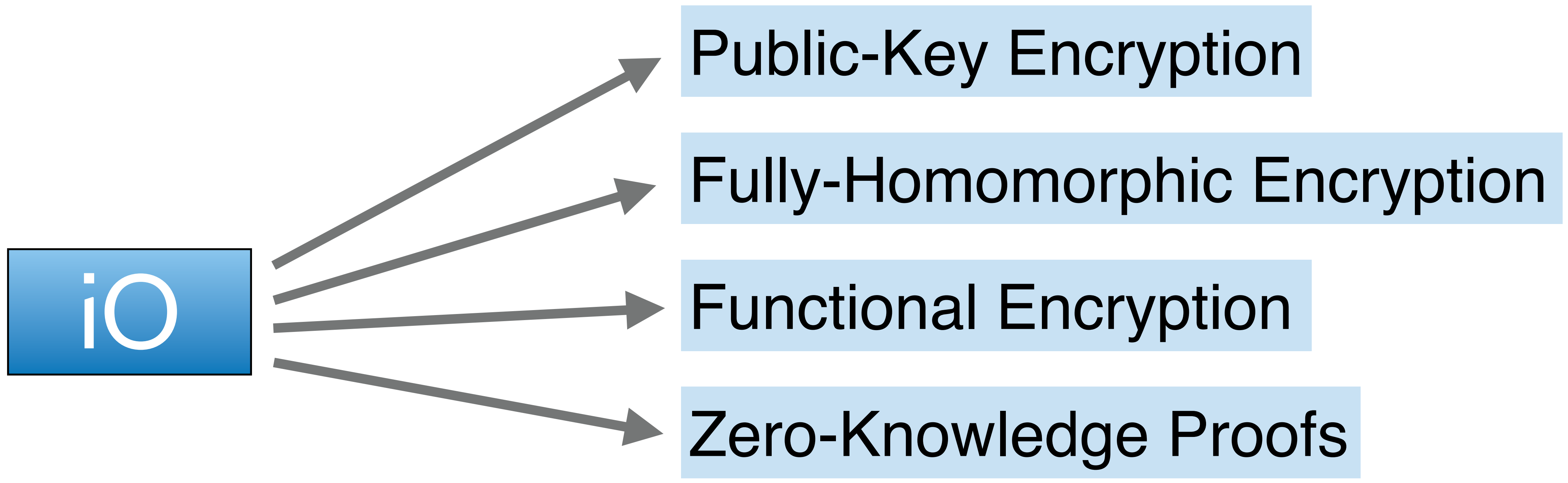
Very powerful object, implies a myriad of cryptographic primitives:

Linea

Indistinguishability Obfuscation (iO)

iO — an obfuscator of circuits.

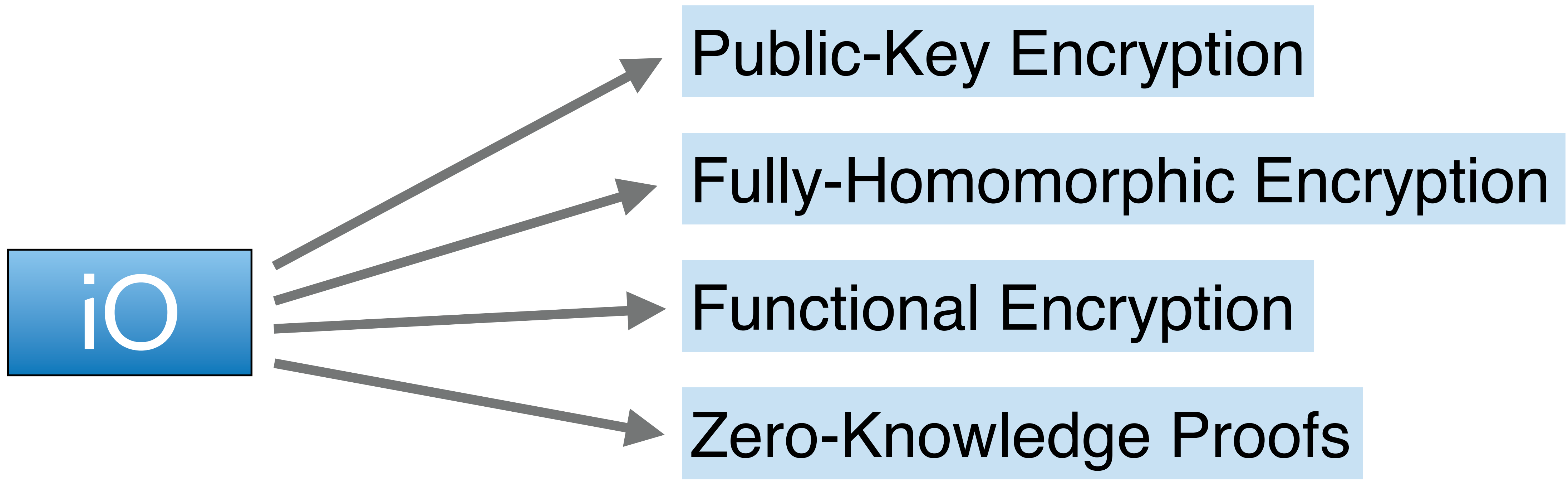
Very powerful object, implies a myriad of cryptographic primitives:



Indistinguishability Obfuscation (iO)

iO — an obfuscator of circuits.

Very powerful object, implies a myriad of cryptographic primitives:



A recent line of works builds IO from falsifiable assumptions.

Main Tool—Punctured PRFs [SW14]

Program(x)

Obfuscation

Hardcoded key K

Randomness $r = \mathsf{PRF}(K, x)$

Use r as coins for some other primitive.

Main Tool—Punctured PRFs [SW14]

Obfuscation

Program(x)

Hardcoded key K

Randomness $r = \text{PRF}(K, x)$

Use r as coins for some other primitive.

Obfuscation

Program(x)

Hardcoded key $K\{x^*\}$, $a = \text{PRF}(K, x^*)$

If $x = x^*$ then $r = a$

Otherwise $r = \text{PRF}(K\{x^*\}, x)$

Use r as coins for some other primitive.

iO security

Linea

Main Tool—Punctured PRFs [SW14]

Program(x) **Obfuscation**

Hardcoded key $K\{x^*\}$, $a = \text{PRF}(K, x^*)$

If $x = x^*$ then $r = a$

Otherwise $r = \text{PRF}(K\{x^*\}, x)$

Use r as coins for some other primitive.

Main Tool—Punctured PRFs [SW14]

Obfuscation

Program(x)

Hardcoded key $K\{x^*\}$, $a = \text{PRF}(K, x^*)$

If $x = x^*$ then $r = a$

Otherwise $r = \text{PRF}(K\{x^*\}, x)$

Use r as coins for some other primitive.

Obfuscation

Program(x)

Hardcoded key $K\{x^*\}$, random a

If $x = x^*$ then $r = a$

Otherwise $r = \text{PRF}(K\{x^*\}, x)$

Use r as coins for some other primitive.

Pseudorandomness of PRF

Linea

Main Tool—Punctured PRFs [SW14]

Program(x) Obfuscation

Hardcoded key $K\{x^*\}$, $a = \text{PRF}(K, x^*)$

If $x = x^*$ then $r = a$

Otherwise $r = \text{PRF}(K\{x^*\}, x)$

Use r as coins for some other primitive.

Program(x) Obfuscation

Hardcoded key $K\{x^*\}$, random a

If $x = x^*$ then $r = a$

Otherwise $r = \text{PRF}(K\{x^*\}, x)$

Use r as coins for some other primitive.

Use security of the primitive.

Pseudorandomness of PRF

FHS Construction: Idea 1

Let DS be a non-homomorphic regular signature.

$\mathsf{FHS}\,.\,\mathsf{Sign}(m_i) = \mathsf{DS}\,.\,\mathsf{Sign}_{\mathsf{sk}}(m_i)$

FHS Construction: Idea 1

Let DS be a non-homomorphic regular signature.

$\mathsf{FHS.Sign}(m_i) = \mathsf{DS.Sign}_{\mathsf{sk}}(m_i)$

$\mathsf{EvalNand}((m_0, \sigma_0), (m_1, \sigma_1))$ Obfuscation

Hardcoded key K, let $x = m_0$ NAND m_1

Randomness $r = \mathsf{PRF}(K, x)$

Use r as the sk' of DS

if $\mathsf{Verify}_{vk_b}(C_b, m_b, \sigma_b) = 1$ then

Output $\mathsf{DS.Sign}_{\mathsf{sk}'}(x)$

Linea

FHS Idea 1: Issues

Let DS be a non-homomorphic regular signature.

$\text{FHS}.\text{Sign}(m_i) = \text{DS}.\text{Sign}_{\text{sk}}(m_i)$

No way to predict at which x the attacker will forge—in order to puncture at x,

$\text{EvalNand}((m_0, \sigma_0), (m_1, \sigma_1))$ Obfuscation

Hardcoded key K let $x = m_0$ NAND m_1

Randomness $r = \text{PRF}(K, x)$

Use r as the sk' of DS

if $\text{Verify}_{vk_b}(C_b, m_b, \sigma_b) = 1$ then

Output $\text{DS}.\text{Sign}_{\text{sk}'}(m_0 \text{ NAND } m_1)$

FHS Idea 1: Issues

Let DS be a non-homomorphic regular signature.

$\text{FHS.Sign}(m_i) = \text{DS.Sign}_{\text{sk}}(m_i)$

No way to predict at which x the attacker will forge—in order to puncture at x,

EvalNand does not have circuits C_b.

EvalNand$((m_0, \sigma_0), (m_1, \sigma_1))$ Obfuscation

Hardcoded key K let $x = m_0$ NAND m_1

Randomness $r = \text{PRF}(K, x)$

Use r as the sk' of DS

if $\text{Verify}_{vk_b}(C_b, m_b, \sigma_b) = 1$ then

Output $\text{DS.Sign}_{\text{sk}'}(m_0 \text{ NAND } m_1)$

FHS Idea 1: Issues

Let DS be a non-homomorphic regular signature.

$\text{FHS . Sign}(m_i) = \text{DS . Sign}_{\text{sk}}(m_i)$

No way to predict at which x the attacker will forge—in order to puncture at x,

EvalNand does not have circuits C_b.

Even if we somehow manage to puncture by guessing—sk simply becomes fully random—we still must remove it from the program to use DS unforgettability.

$\text{EvalNand}((m_0, \sigma_0), (m_1, \sigma_1))$ Obfuscation

Hardcoded key K let $x = m_0$ NAND m_1

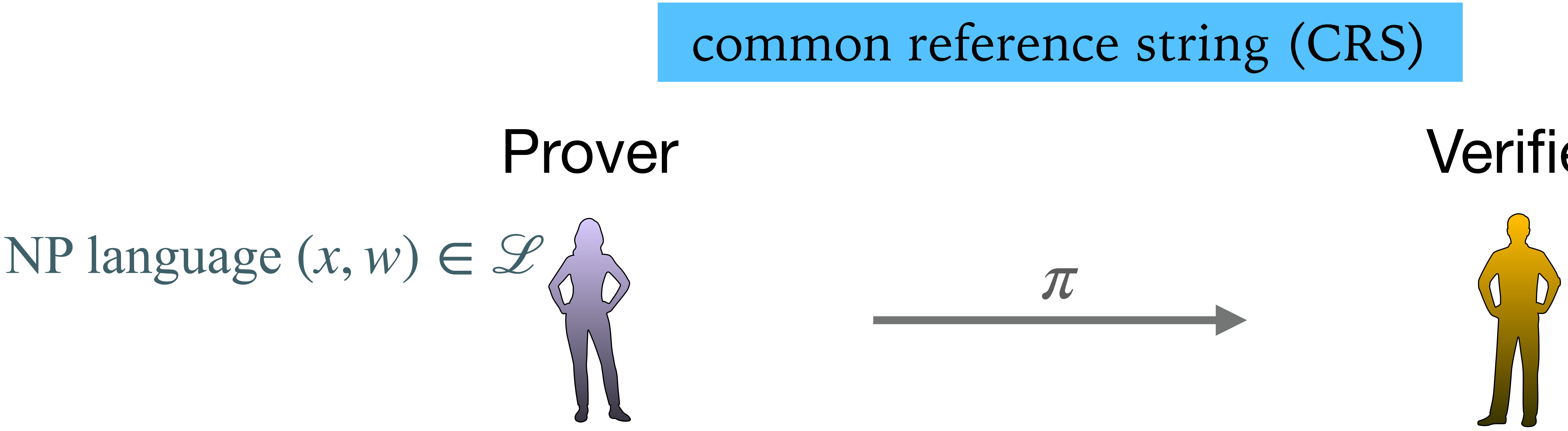
Randomness $r = \text{PRF}(K, x)$

Use r as the sk' of DS

if $\text{Verify}_{vk_b}(C_b, m_b, \sigma_b) = 1$ then

Output $\text{DS . Sign}_{\text{sk}'}(m_0 \text{ NAND } m_1)$

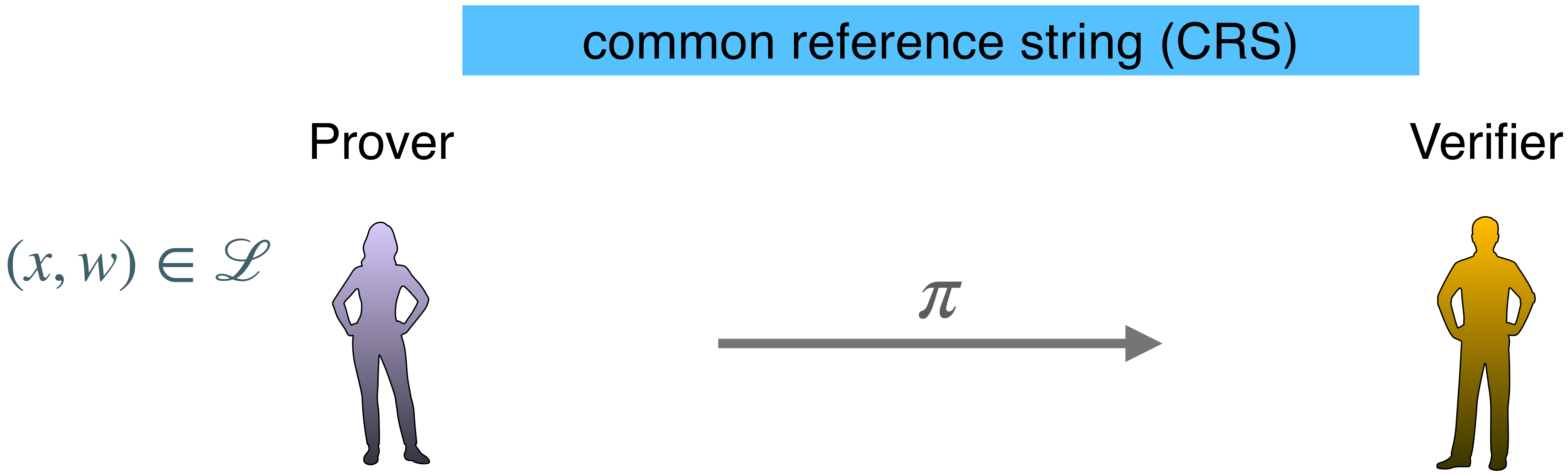
Non-Interactive Zero-Knowledge Proofs (NIZKs)



Interaction consists of only one message.

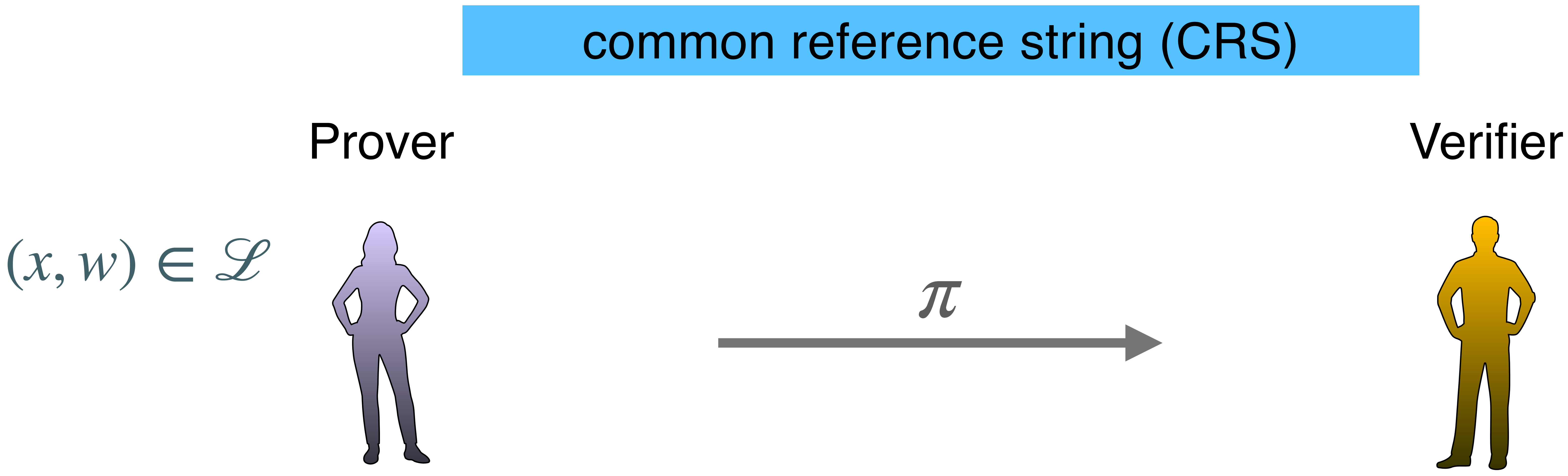
Both parties have access to the CRS, generated in an honest setup phase.

Building Block: NIZKs



CRS is in two modes:

Building Block: NIZKs



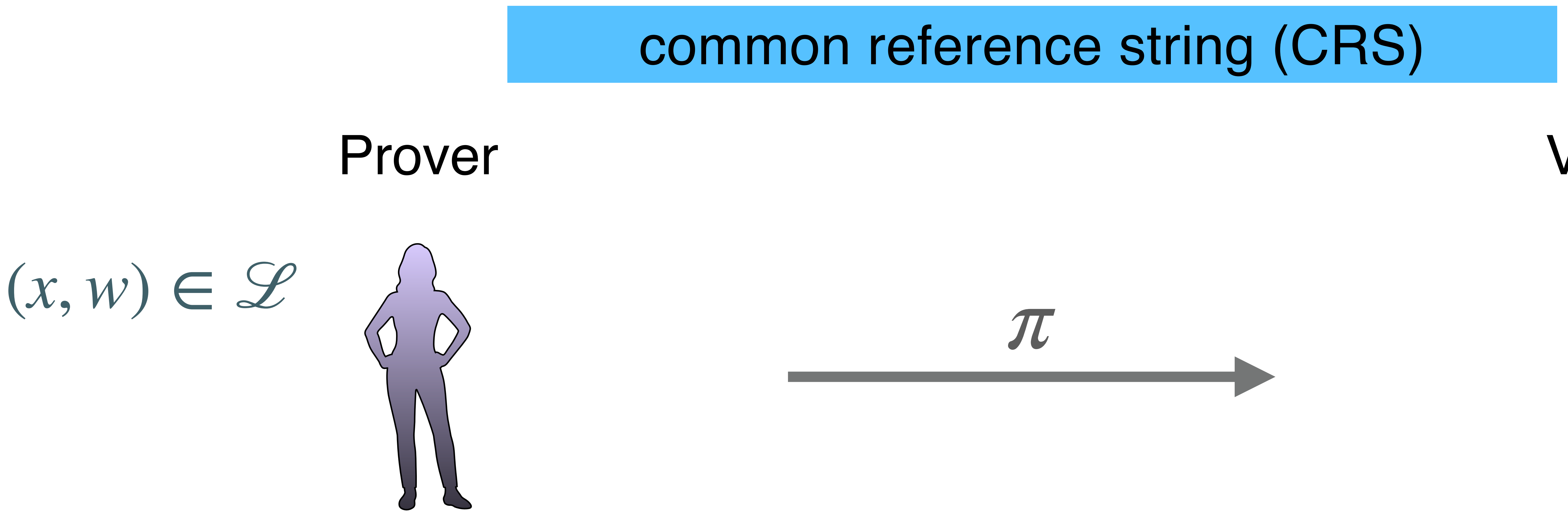
CRS is in two modes:

Binding CRS soundness, crs is generated along with an extraction trapdoor td_{ext}.

td_{ext} can be used to retrieve the witness.

Linea

Building Block: NIZKs



CRS is in two modes:

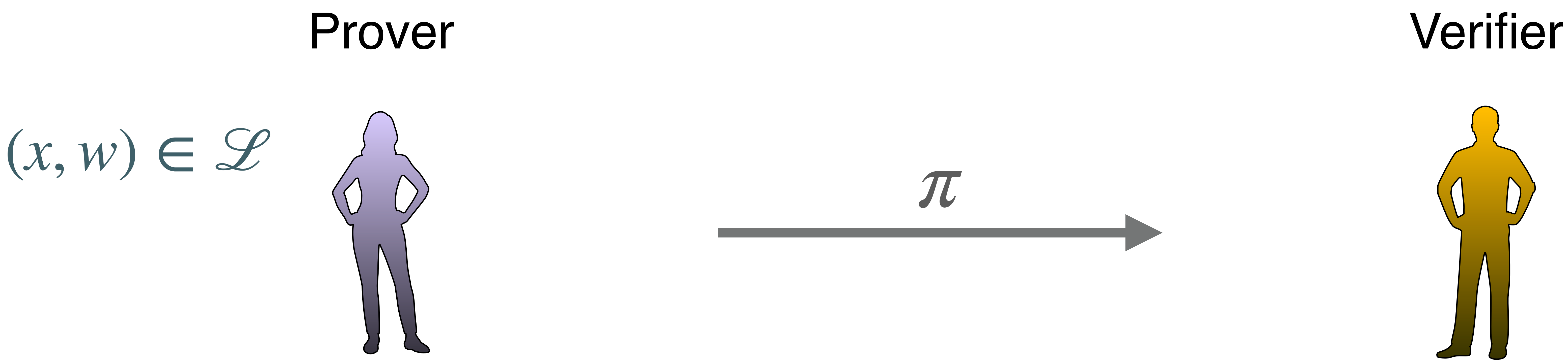
Binding CRS soundness, crs is generated along with an extraction trapdoor td_{ext}.
td_{ext} can be used to retrieve the witness.

Hiding CRS zero-knowledge, crs is generated along with a simulation trapdoor td_{sim}.
td_{sim} can be used to simulate proofs without a witness (fake proofs).

Linea

Building Block: NIZKs

common reference string (CRS)



CRS is in two modes:

Binding CRS soundness, crs is generated along with an extraction trapdoor $\mathsf{td}_{\mathsf{ext}}$.
$\mathsf{td}_{\mathsf{ext}}$ can be used to retrieve the witness.

Hiding CRS zero-knowledge, crs is generated along with a simulation trapdoor $\mathsf{td}_{\mathsf{sim}}$.
$\mathsf{td}_{\mathsf{sim}}$ can be used to simulate proofs without a witness (fake proofs).

The two modes are computationally indistinguishable.

FHS Idea 2

$\mathsf{FHS}.\mathsf{Sign}(m_i) = \sigma_i = (\mathsf{ct}_i, \pi_i)$, where π is a NIZK proof that $\mathsf{ct}_i = \mathsf{FHE}.\mathsf{Enc}_{\mathsf{pk}}(m_i)$

The ct_i component of the signature is homomorphic by default.

FHS Idea 2

$\mathsf{FHS.Sign}(m_i) = \sigma_i = (\mathsf{ct}_i, \pi_i)$, where π is a NIZK proof that $\mathsf{ct}_i = \mathsf{FHE.Enc}_{\mathsf{pk}}(m_i)$

The ct_i component of the signature is homomorphic by default.

$\mathsf{EvalNand}((m_0, \sigma_0), (m_1, \sigma_1))$ Obfuscation

Hardcoded key K let $x = m_0$ NAND m_1

Randomness $r = \mathsf{PRF}(K, x)$

Use r as the generating randomness of the NIZK crs (common reference string)

Compute $\mathsf{ct}' = \mathsf{FHE.Eval}_{\mathsf{pk,NAND}}(\mathsf{ct}_0, \mathsf{ct}_1)$

if $\mathsf{NIZK.Verify}(\mathsf{crs}, \pi_b) = 1$ then

Output proof π' for ct'.

How to compute π'

Witness is the randomness r' of ct$'$.

extract r_b from π_b using td_{ext} →

compute r' as $r' = \mathsf{FHE.EvalRand}_{pk,sk}(r_0, r_1)$

EvalNand$((m_0, \sigma_0), (m_1, \sigma_1))$ Obfuscation

Hardcoded key K let $x = m_0$ NAND m_1

Randomness $r = \mathsf{PRF}(K, x)$

Use r as the generating randomness of the NIZK crs (common reference string)

Compute
$\mathsf{ct}' = \mathsf{FHE.Eval}_{pk,NAND}(\mathsf{ct}_0, \mathsf{ct}_1)$

if $\mathsf{NIZK.Verify}(\mathsf{crs}, \pi_b) = 1$ then

Output proof π' for ct$'$ **(using r')**.

Linea

But Now Anyone Can Sign

$\mathsf{FHS.Sign}(m_i) = \sigma_i = (\mathsf{ct}_i, \pi_i)$, where π is a NIZK proof that $\mathsf{ct}_i = \mathsf{FHE.Enc}_{\mathsf{pk}}(m_i)$.

Simply compute an FHE ciphertext by generating r and $\mathsf{ct} = \mathsf{FHE.Enc}(m; r)$.

Then use r to compute the corresponding NIZK proof.

Linea

But Now Anyone Can Sign—Fix

$\mathsf{FHS.Sign}(m_i) = \sigma_i = (\mathsf{ct}_i, \pi_i)$, where π is a NIZK proof that $\mathsf{ct}_i = \mathsf{FHE.Enc}_{\mathsf{pk}}(m_i)$.

Generate n FHE ciphertexts for the initial messages $m_1 \ldots m_n$, i.e. $\mathsf{ct}_i = \mathsf{FHE.Enc}(m_i; r_i)$.

Include the initial $\mathsf{ct}_1 \ldots \mathsf{ct}_n$ inside the verification key.

Keep initial randomness r_i private to compute the proofs π_i that certify that ct_i encrypts m_i.

But Now Anyone Can Sign—Fix

$\mathsf{FHS}.\mathsf{Sign}(m_i) = \sigma_i = (\mathsf{ct}_i, \pi_i)$, where π is a NIZK proof that $\mathsf{ct}_i = \mathsf{FHE}.\mathsf{Enc}_{\mathsf{pk}}(m_i)$.

Generate n FHE ciphertexts for the initial messages $m_1 \ldots m_n$, i.e. $\mathsf{ct}_i = \mathsf{FHE}.\mathsf{Enc}(m_i; r_i)$.

Include the initial $\mathsf{ct}_1 \ldots \mathsf{ct}_n$ inside the verification key.

Keep initial randomness r_i private to compute the proofs π_i that certify that ct_i encrypts m_i.

But Now Anyone Can Sign—New Issue

$\mathsf{FHS}\,.\,\mathsf{Sign}(m_i) = \sigma_i = (\mathsf{ct}_i, \pi_i)$, where π is a NIZK proof that $\mathsf{ct}_i = \mathsf{FHE}\,.\,\mathsf{Enc}_{\mathsf{pk}}(m_i)$.

Generate n FHE ciphertexts for the initial messages $m_1 \ldots m_n$, i.e. $\mathsf{ct}_i = \mathsf{FHE}\,.\,\mathsf{Enc}(m_i; r_i)$.

Include the initial $\mathsf{ct}_1 \ldots \mathsf{ct}_n$ inside the verification key.

Keep initial randomness r_i private to compute the proofs π_i that certify that ct_i encrypts m_i.

When computing the vk we do not know what messages will be signed!

Linea

But Now Anyone Can Sign—New Issue

$\text{FHS.Sign}(m_i) = \sigma_i = (\text{ct}_i, \pi_i)$, where π is a NIZK proof that $\text{ct}_i = \text{FHE.Enc}_{\text{pk}}(m_i)$.

Generate n FHE ciphertexts for the initial messages $m_1 \ldots m_n$, i.e. $\text{ct}_i = \text{FHE.Enc}(m_i; r_i)$.

Include the initial $\text{ct}_1 \ldots \text{ct}_n$ inside the verification key.

Keep initial randomness r_i private to compute the proofs π_i that certify that ct_i encrypts m_i.

When computing the vk we do not know what messages will be signed!

Hybrid argument where ct_i are encryptions of 0 and π_i are simulated proofs using td_{sim}.

Linea

But Now Anyone Can Sign—New Issue

Many more technical problems to overcome, for details check out the paper.

Conclusion

We build a fully-homomorphic signature (FHS) scheme based on iO, FHE and NIZKs.

Our scheme is the first that achieves the following properties simultaneously:

Conclusion

We build a fully-homomorphic signature (FHS) scheme based on iO, FHE and NIZKs.

Our scheme is the first that achieves the following properties simultaneously:

- supports an unbounded number of levels.
- is arbitrarily composable.
- based on falsifiable assumptions.

Linea

Conclusion

We build a fully-homomorphic signature (FHS) scheme based on iO, FHE and NIZKs.

Our scheme is the first that achieves the following properties simultaneously:

- supports an unbounded number of levels.
- is arbitrarily composable.
- based on falsifiable assumptions.

Pairings + iO (subexp)

or

LWE + iO (subexp)

Thank you for your attention!

Questions?

Linea

Roadmap

- Fully-Homomorphic Encryption, Signatures and Bootstrapping.
- Defining Fully-Homomorphic Signatures.
- State of the Art.
- Preliminaries: iO and NIZKs.
- How to use iO.
- Technical Ideas towards an FHS construction.
- Conclusion.

Fully Homomorphic Encryption

$\mathsf{ct}_1 = \mathsf{FHE.Enc}(\mathsf{pk}, m_1)$ $\mathsf{ct}_2 = \mathsf{FHE.Enc}(\mathsf{pk}, m_2)$

$\mathsf{Eval}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)$

$\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, m_1 + m_2)$ $\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, m_1 \cdot m_2)$ $\mathsf{ct} = \mathsf{FHE.Enc}(\mathsf{pk}, C(m_1, m_2))$

Operations happen modulo a bound parameter B.

LWE-based schemes: every homomorphic operation increases the noise level.

Correctness preserved up to a maximal noise bound B_{noise}.

Linea