zkSNARKs in the ROM with Unconditional UC-Security

TL;DR Micali and BCS are UC-secure in the GROM

Giacomo Fenzi @ EPFL

eprint.iacr.org/2024/724

Joint work with Alessandro Chiesa

EPFL

Motivation

zkSNARKs are deployed in the real world

zkSNARKs are deployed in the real world

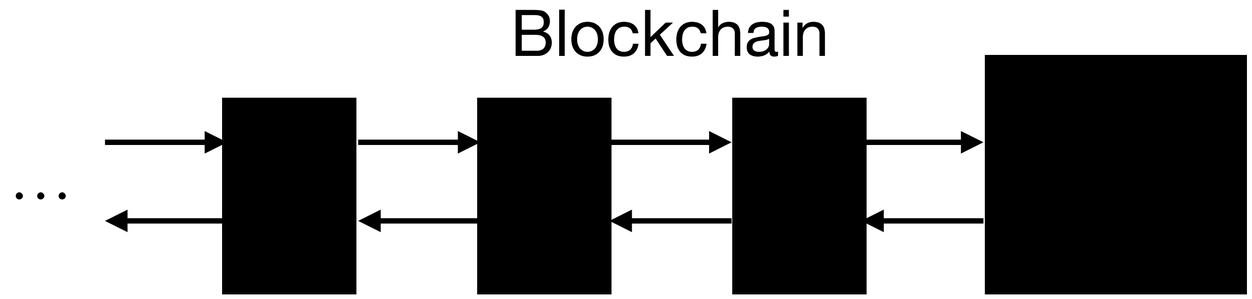
zkSNARKs are ZKPs where verification is **exponentially** faster than execution.

zkSNARKs are deployed in the real world

E.g.: proof based rollups
to improve scalability

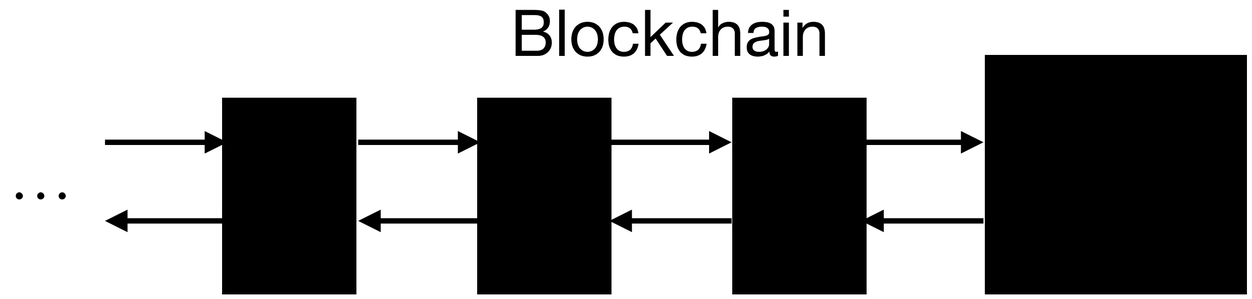
zkSNARKs are deployed in the real world

E.g.: proof based rollups
to improve scalability

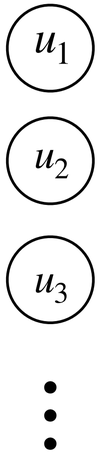


zkSNARKs are deployed in the real world

E.g.: proof based rollups to improve scalability

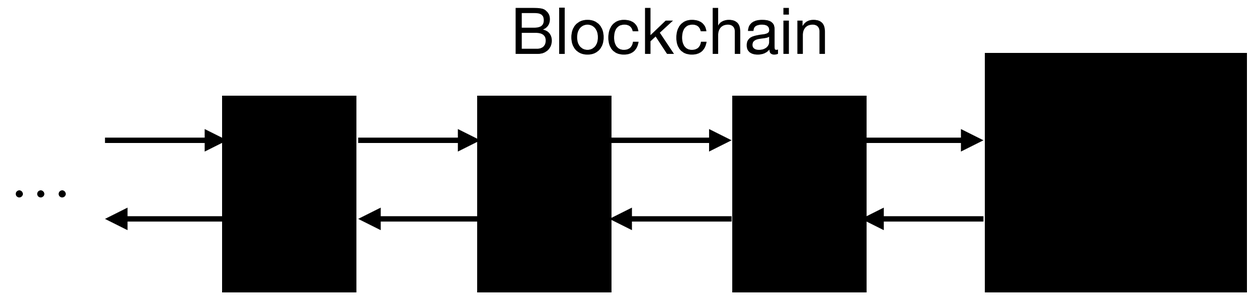


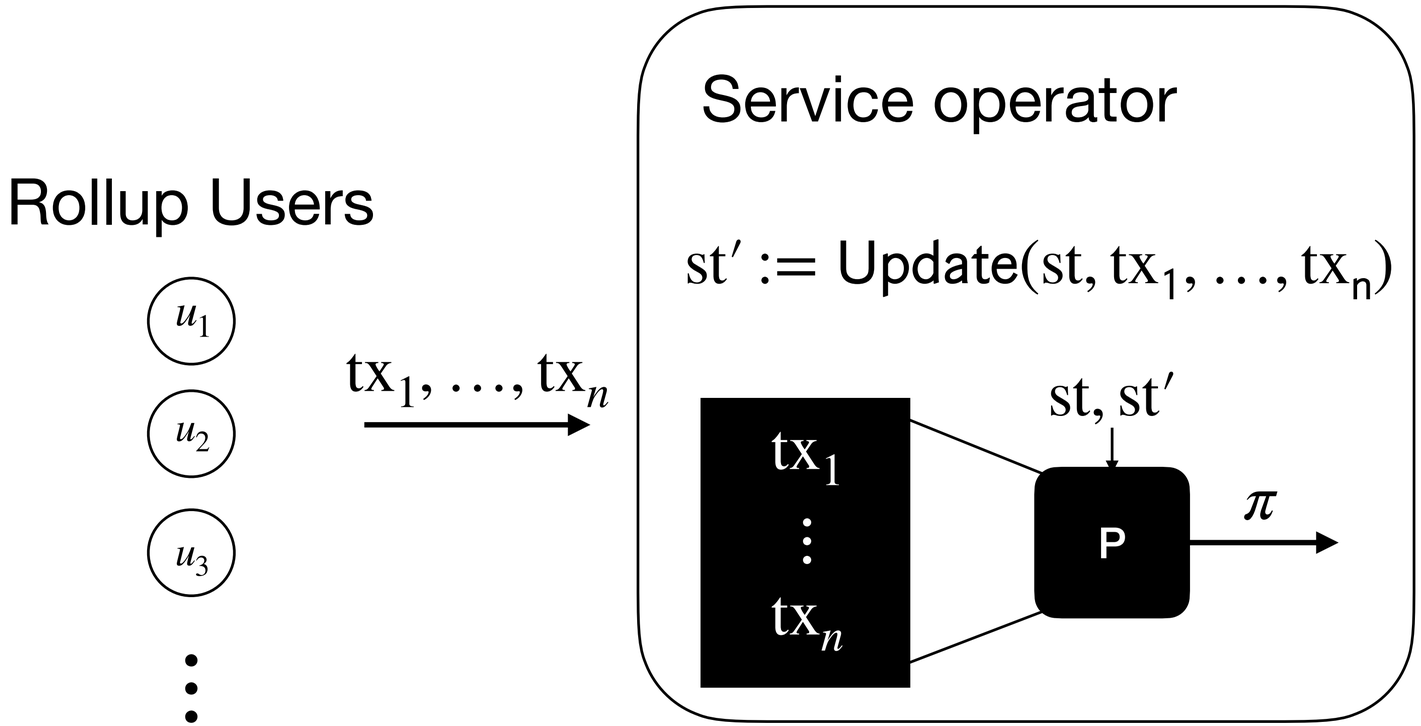
Rollup Users



zkSNARKs are deployed in the real world

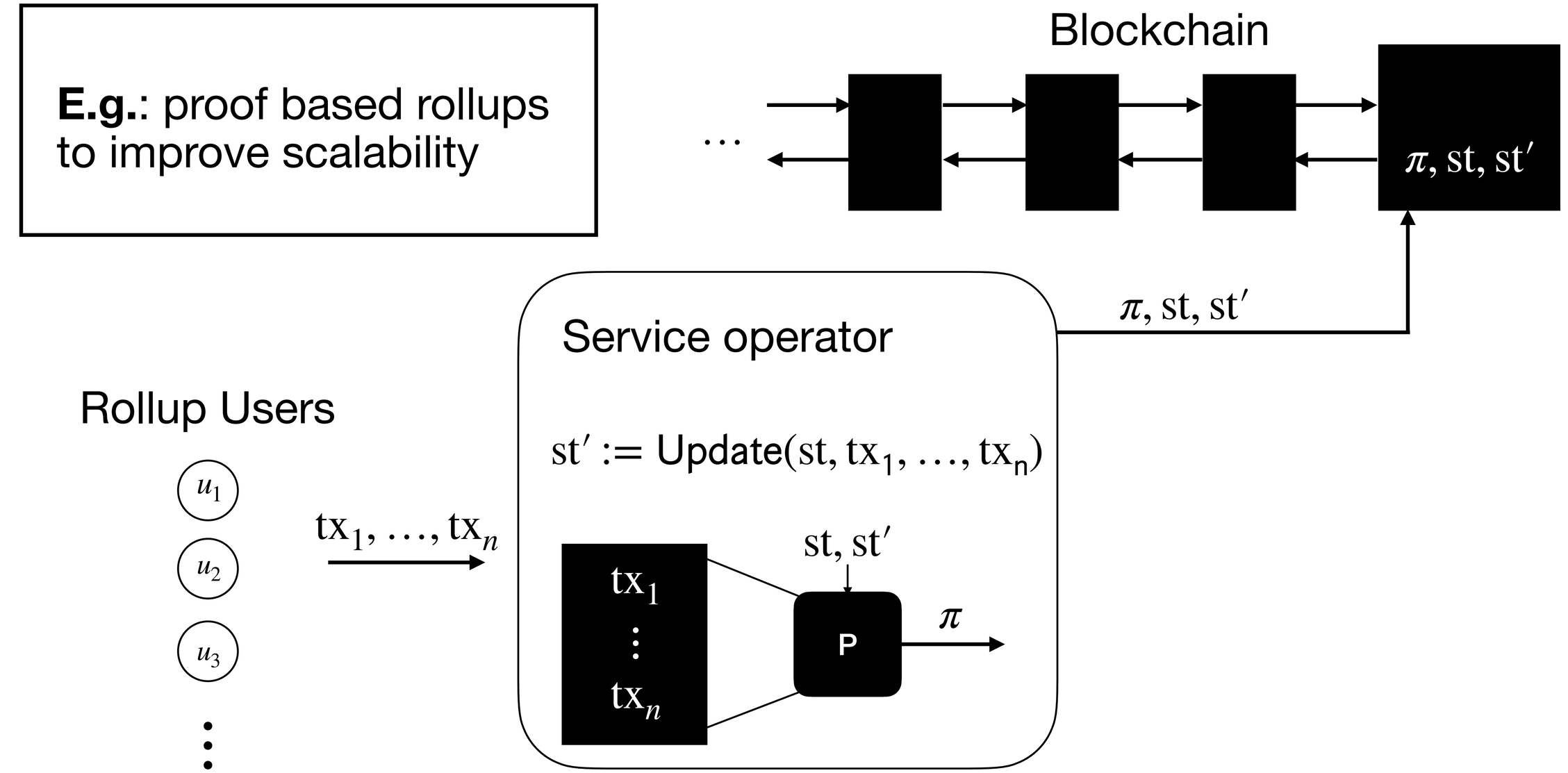
E.g.: proof based rollups to improve scalability





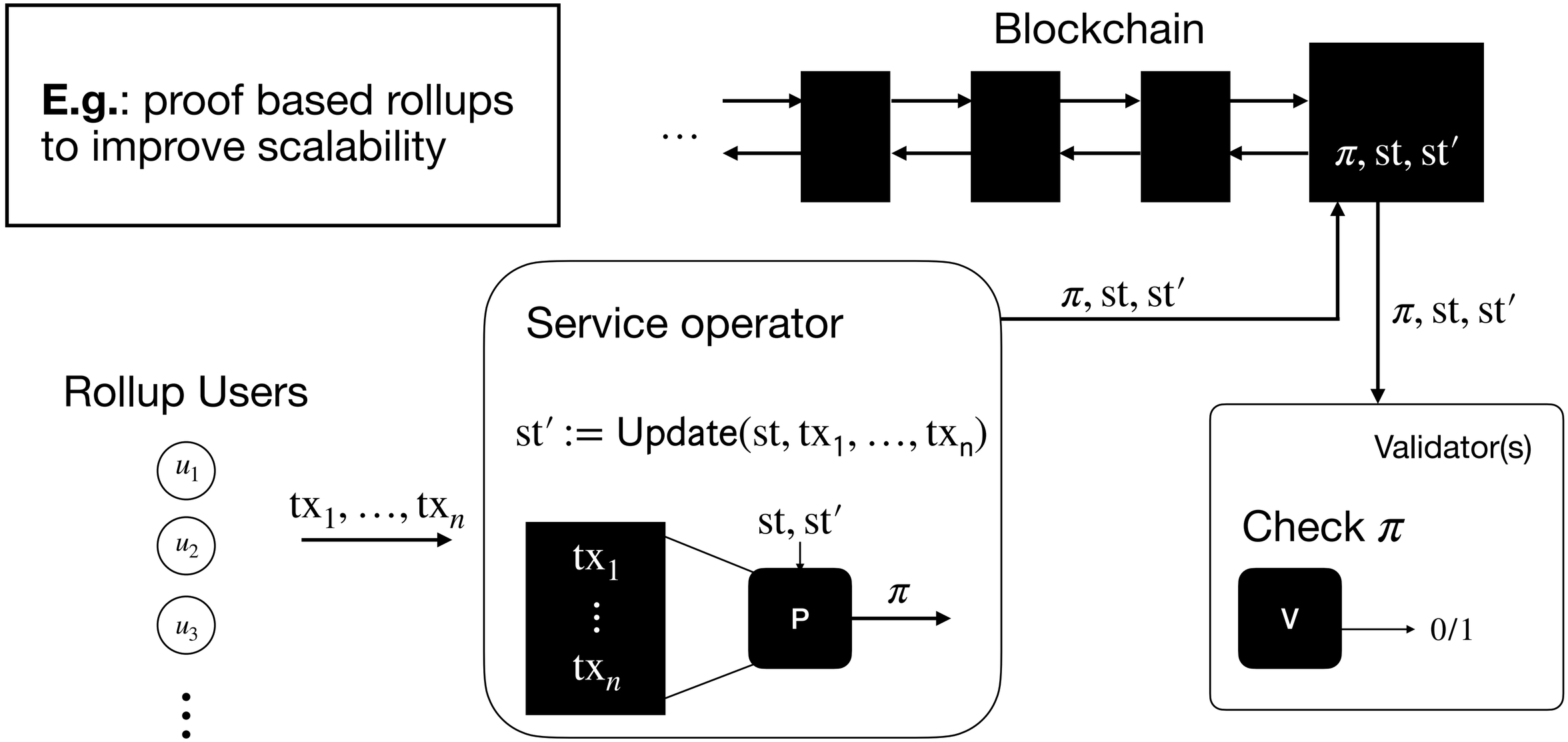
zkSNARKs are deployed in the real world

E.g.: proof based rollups to improve scalability



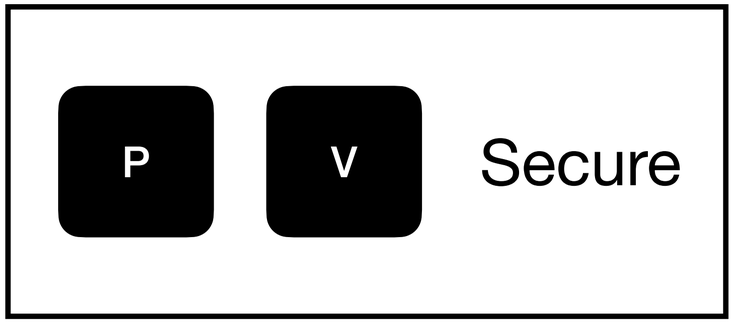
zkSNARKs are deployed in the real world

E.g.: proof based rollups to improve scalability

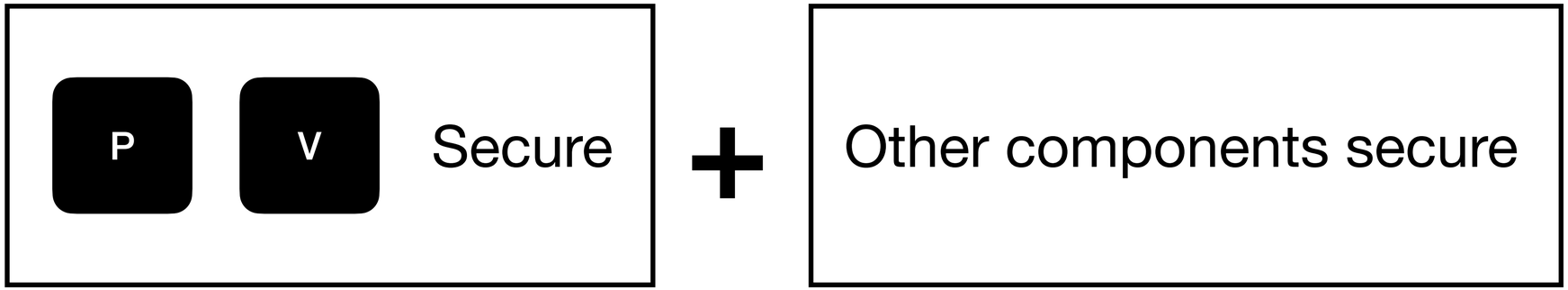


Goal: Modular Security Analysis

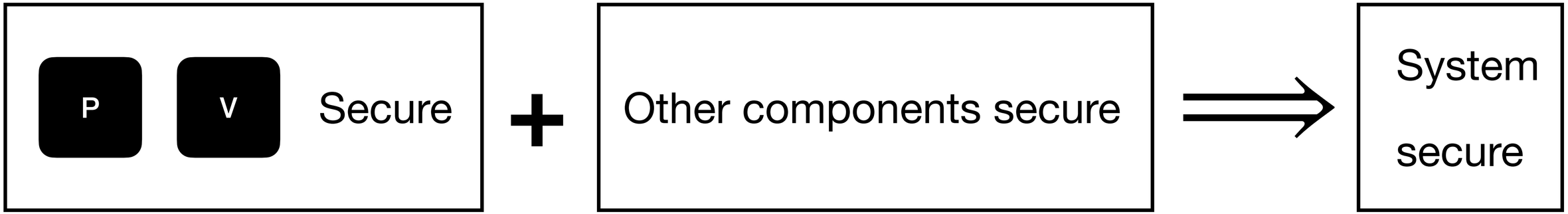
Goal: Modular Security Analysis



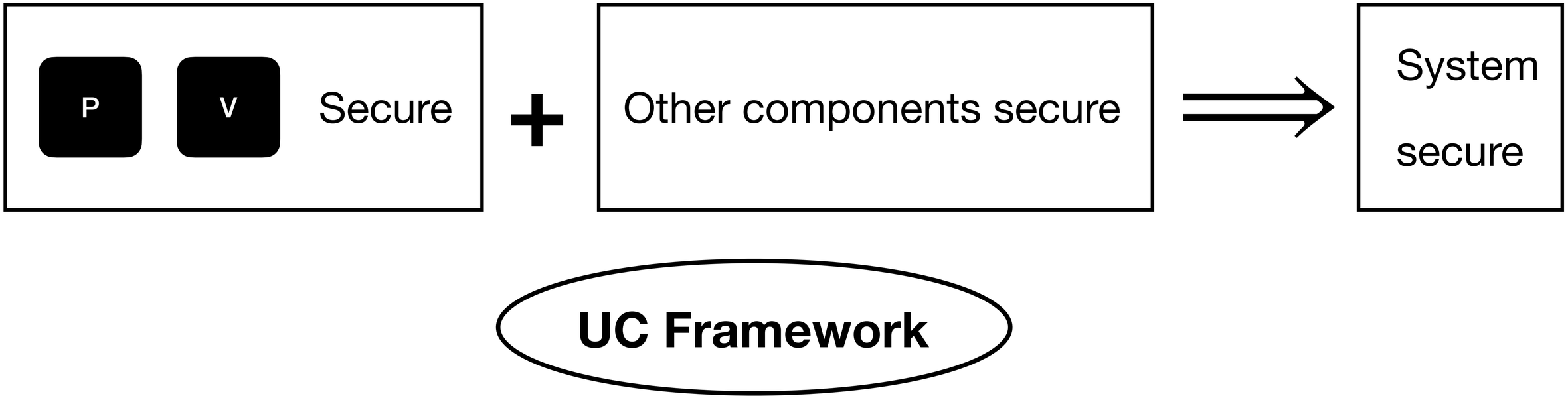
Goal: Modular Security Analysis



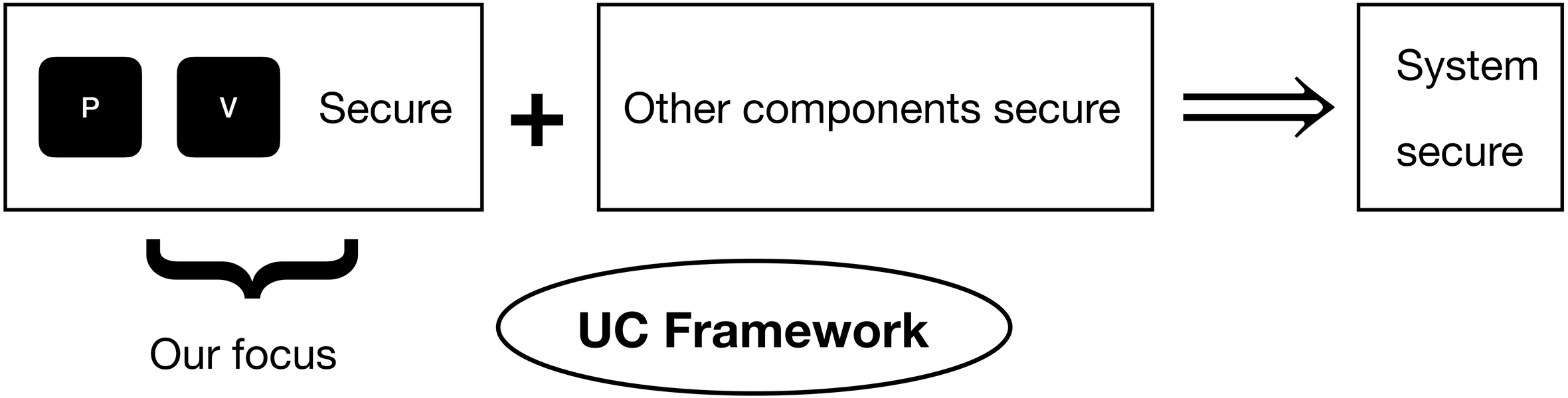
Goal: Modular Security Analysis



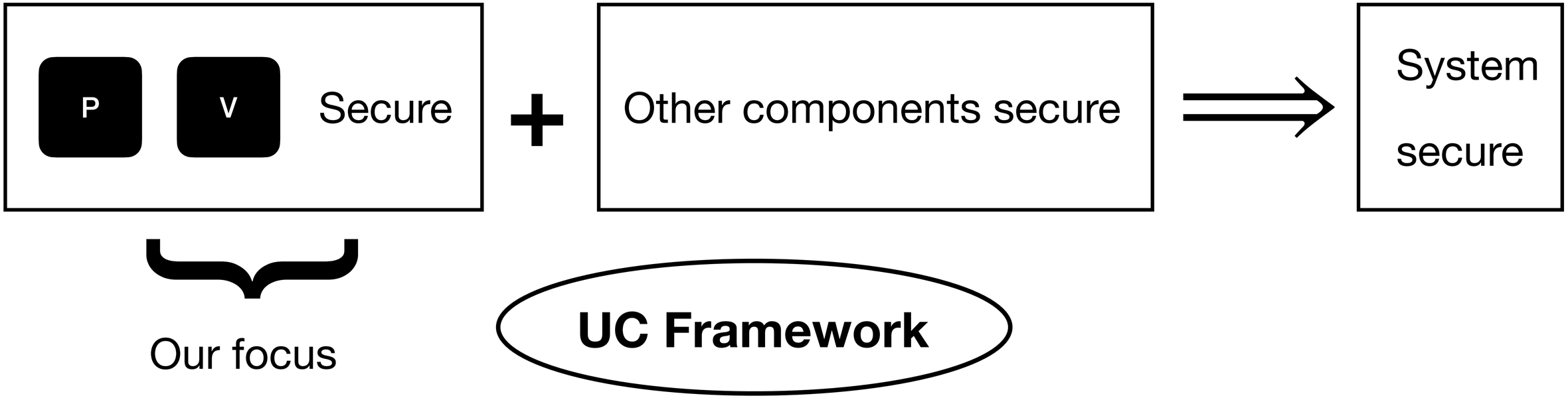
Goal: Modular Security Analysis



Goal: Modular Security Analysis



Goal: Modular Security Analysis



Which zkSNARKs are UC-secure?

Related works

Related works

C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[⋆] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[⋆] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]

Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically⋆

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]

Tiramisu: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat

Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Related works

Append an encryption of the witness to the proof.

- Cannot be succinct $|\pi| \geq |w|$

C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[⋆] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[⋆] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]

Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically⋆

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]

TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat

Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Related works

Append an encryption of the witness to the proof.

- Cannot be succinct $|\pi| \geq |w|$

C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[⋆] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[⋆] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]

Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically⋆

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]

TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat

Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Related works

Append an encryption of the witness to the proof.

- Cannot be succinct $|\pi| \geq |w|$

C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[⋆] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[⋆] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]

Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically⋆

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]

TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat

Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Compile Σ-protocol into NIZK

+ Techniques inspired this work

- Not succinct

- Expensive compilation (non-FS)

Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Succinct UC-secure zkSNARKs

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

Succinct UC-secure zkSNARKs

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Succinct UC-secure zkSNARKs

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Combines simulation-extractable zkSNARK with a PCS

Succinct UC-secure zkSNARKs

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Combines simulation-extractable zkSNARK with a PCS

Use Fischlin-like techniques to achieve straight-line extraction

Succinct UC-secure zkSNARKs

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

+ Achieves succinct proofs

Combines simulation-extractable zkSNARK with a PCS

Use Fischlin-like techniques to achieve straight-line extraction

Succinct UC-secure zkSNARKs

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Combines simulation-extractable zkSNARK with a PCS

Use Fischlin-like techniques to achieve straight-line extraction

+ Achieves succinct proofs

+ UC-Secure in the (non-programmable) observable GROM

Succinct UC-secure zkSNARKs

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Combines simulation-extractable zkSNARK with a PCS

Use Fischlin-like techniques to achieve straight-line extraction

\+ Achieves succinct proofs

\+ UC-Secure in the (non-programmable) observable GROM

\- Expensive non-standard construction

Succinct UC-secure zkSNARKs

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

First UC-secure **S**NARK

Combines simulation-extractable zkSNARK with a PCS

Use Fischlin-like techniques to achieve straight-line extraction

\+ Achieves succinct proofs

\+ UC-Secure in the (non-programmable) observable GROM

\- Expensive non-standard construction

\- Focuses on asymptotic security

This work

zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

This work

zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Show **existing** zkSNARKs are UC-secure
(including deployed ones)

This work

zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Show **existing** zkSNARKs are UC-secure
(including deployed ones)

Succinct

This work

zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Show **existing** zkSNARKs are UC-secure
(including deployed ones)

Succinct

ROM **only**: transparent, post-quantum,
unconditional security

This work

zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Show **existing** zkSNARKs are UC-secure (including deployed ones)

Succinct

ROM **only**: transparent, post-quantum, unconditional security

Concrete security bounds: useful for practitioners

Background

UC Security I

[Canetti 2001]

UC Security I

[Canetti 2001]

- Motivation: Modular security analysis of protocols

UC Security I

[Canetti 2001]

- Motivation: Modular security analysis of protocols
- Why UC? ‘Gold-standard’ + vast literature

UC Security I

[Canetti 2001]

- Motivation: Modular security analysis of protocols
- Why UC? ‘Gold-standard’ + vast literature

Composition Theorem

UC Security I

π : protocol

φ : ideal functionality

ρ : calling protocol

- Motivation: Modular security analysis of protocols
- Why UC? ‘Gold-standard’ + vast literature

Composition Theorem

UC Security I

π : protocol

φ : ideal functionality

ρ : calling protocol

- Motivation: Modular security analysis of protocols
- Why UC? ‘Gold-standard’ + vast literature

Composition Theorem

UC Security I

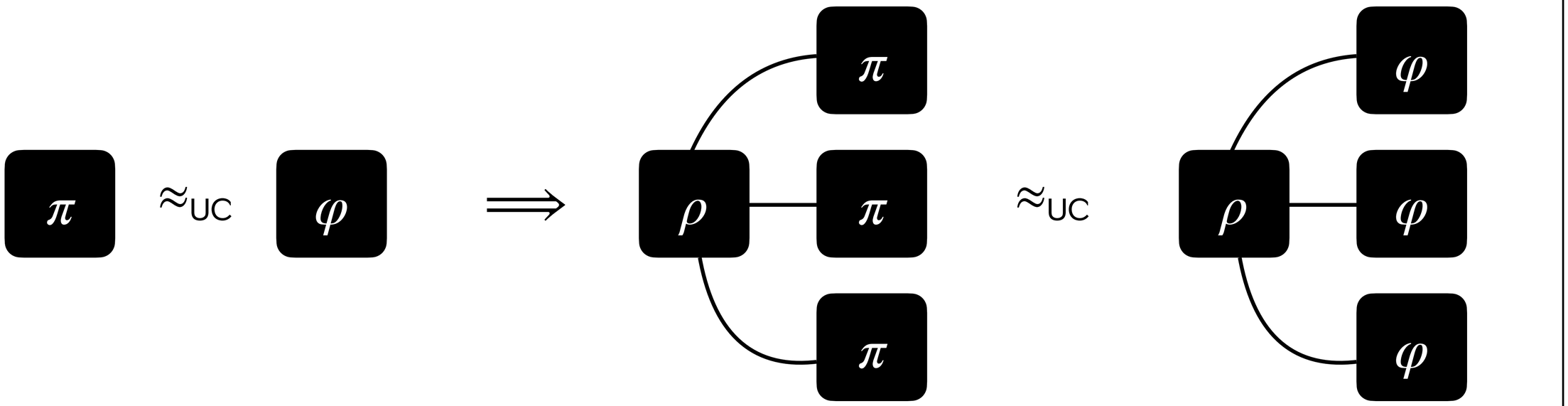
π : protocol

φ : ideal functionality

ρ : calling protocol

- Motivation: Modular security analysis of protocols
- Why UC? ‘Gold-standard’ + vast literature

Composition Theorem



UC Security II

UC Security II

π : protocol	$\mathscr{E}$: environment
$\mathscr{F}$: ideal functionality	$\mathscr{A}$: adversary
D : dummy party	$\mathscr{S}$: simulator

UC Security II

π : protocol

$\mathscr{E}$: environment

$\mathscr{F}$: ideal functionality

$\mathscr{A}$: adversary

D : dummy party

$\mathscr{S}$: simulator

$\pi \approx_{\mathrm{UC}} \mathscr{F}$

UC Security II

Goal: Cannot distinguish protocol from idealized version.

$\pi \approx_{\text{UC}} \mathcal{F}$

π : protocol	$\mathcal{E}$: environment
$\mathcal{F}$: ideal functionality	$\mathcal{A}$: adversary
D : dummy party	$\mathcal{S}$: simulator

UC Security II

Goal: Cannot distinguish protocol from idealized version.

$$\pi \approx_{\text{UC}} \mathcal{F}$$
$$\Longleftrightarrow$$
$$\forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{E}$$

π : protocol	$\mathcal{E}$: environment
$\mathcal{F}$: ideal functionality	$\mathcal{A}$: adversary
D : dummy party	$\mathcal{S}$: simulator

UC Security II

Goal: Cannot distinguish protocol from idealized version.

π : protocol

$\mathscr{F}$: ideal functionality

D : dummy party

$\mathscr{E}$: environment

$\mathscr{A}$: adversary

$\mathscr{S}$: simulator

Real

$$\pi \approx_{\text{UC}} \mathscr{F}$$
$$\iff$$
$$\forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

UC Security II

Goal: Cannot distinguish protocol from idealized version.

π : protocol	$\mathscr{E}$: environment
$\mathscr{F}$: ideal functionality	$\mathscr{A}$: adversary
D : dummy party	$\mathscr{S}$: simulator

$$\pi \approx_{\text{UC}} \mathscr{F}$$
$$\Longleftrightarrow$$
$$\forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

Real

Ideal

$$\approx$$

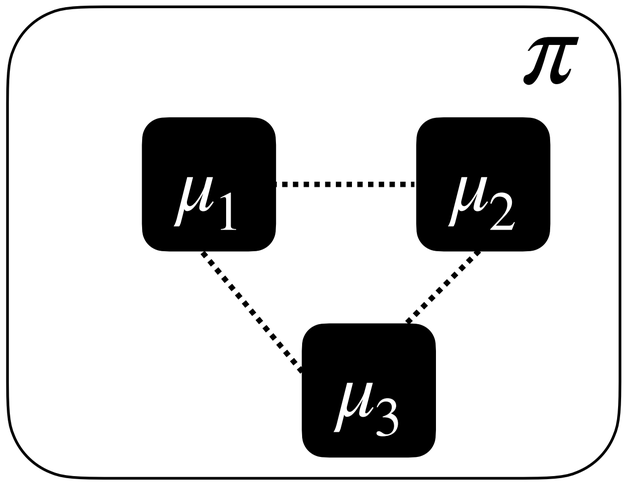
UC Security II

Goal: Cannot distinguish protocol from idealized version.

π : protocol
$\mathscr{E}$: environment
$\mathscr{F}$: ideal functionality
$\mathscr{A}$: adversary
D : dummy party
$\mathscr{S}$: simulator

$$\pi \approx_{\text{UC}} \mathscr{F} \iff \forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

Real



$\approx$

Ideal

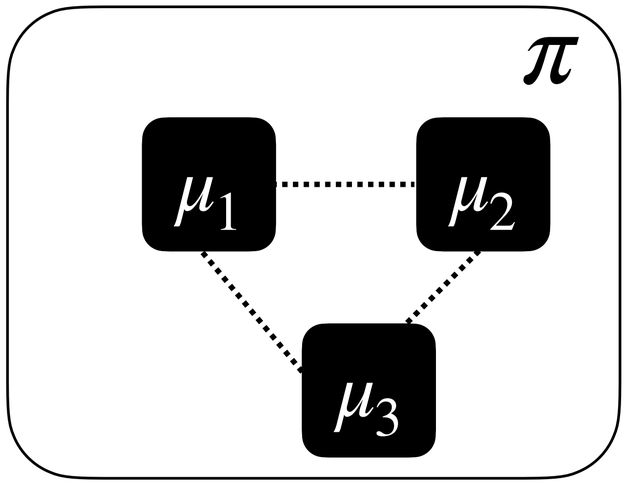
UC Security II

Goal: Cannot distinguish protocol from idealized version.

π : protocol
$\mathscr{E}$: environment
$\mathscr{F}$: ideal functionality
$\mathscr{A}$: adversary
D : dummy party
$\mathscr{S}$: simulator

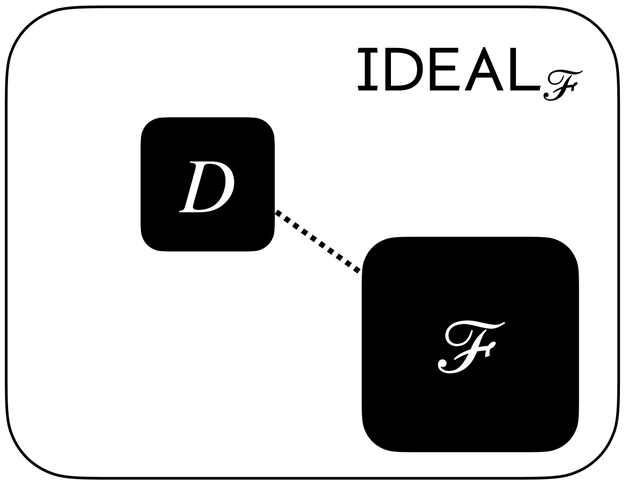
$$\pi \approx_{\text{UC}} \mathscr{F}$$
$$\Longleftrightarrow$$
$$\forall \mathscr{A}, \exists \mathscr{S}, \forall \mathscr{E}$$

Real



$\approx$

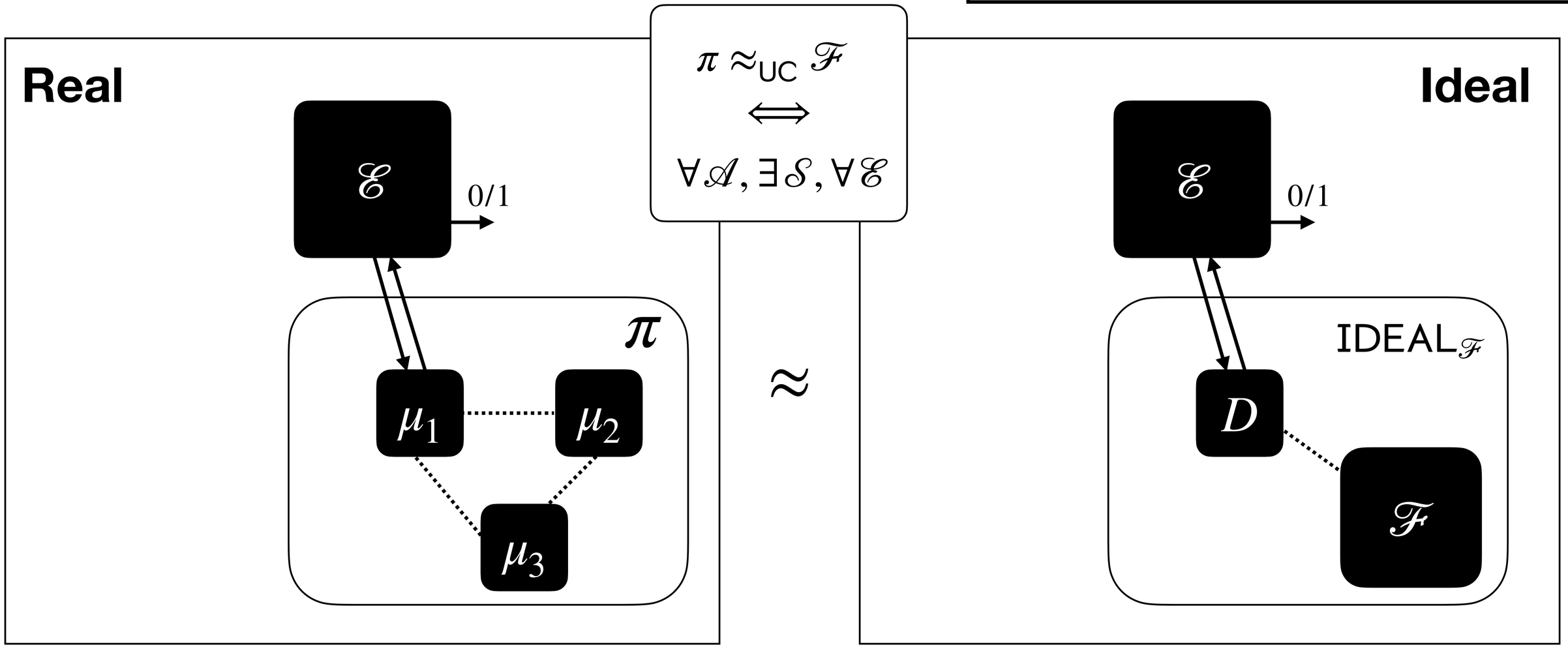
Ideal



UC Security II

Goal: Cannot distinguish protocol from idealized version.

π : protocol
$\mathscr{E}$: environment
$\mathscr{F}$: ideal functionality
$\mathscr{A}$: adversary
D : dummy party
$\mathscr{S}$: simulator

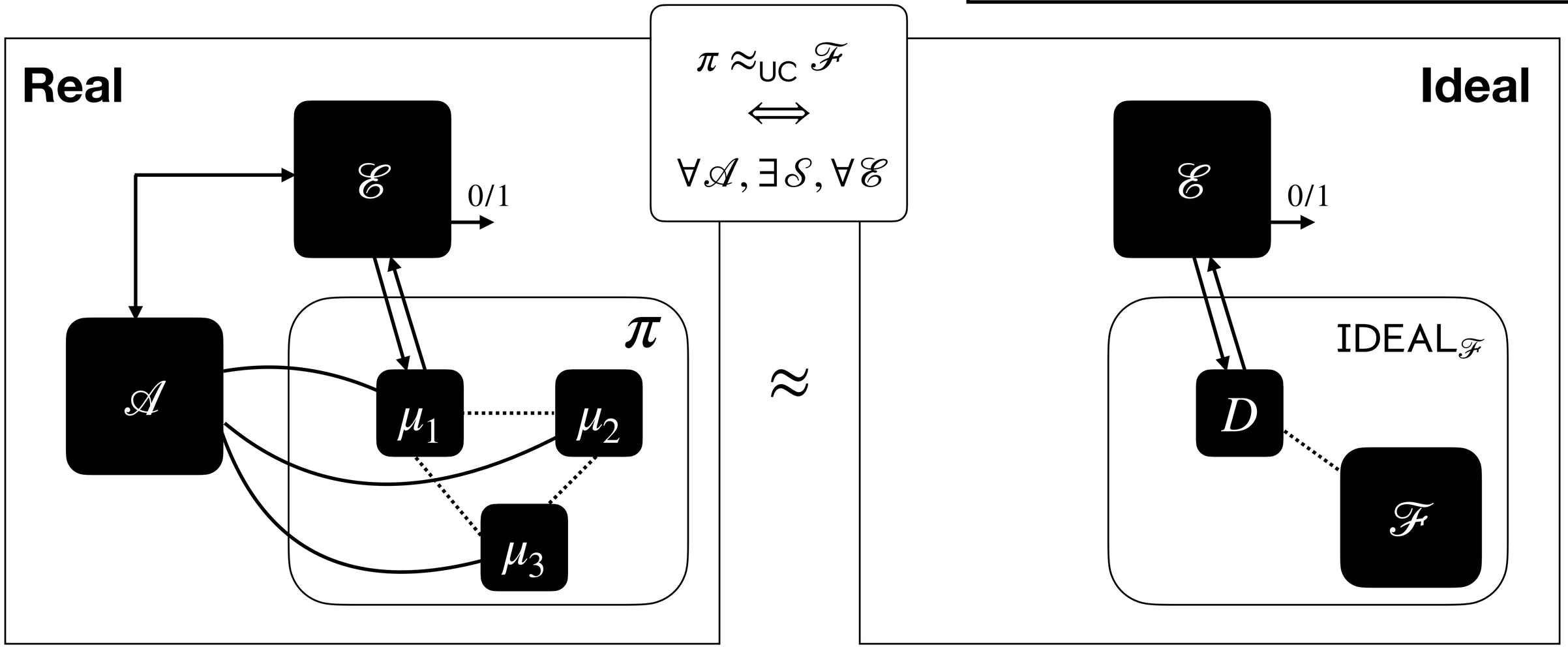


UC Security II

Goal: Cannot distinguish protocol from idealized version.

π : protocol
$\mathcal{E}$: environment
$\mathcal{F}$: ideal functionality
$\mathcal{A}$: adversary
D : dummy party
$\mathcal{S}$: simulator

$$\pi \approx_{\text{UC}} \mathcal{F} \iff \forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{E}$$

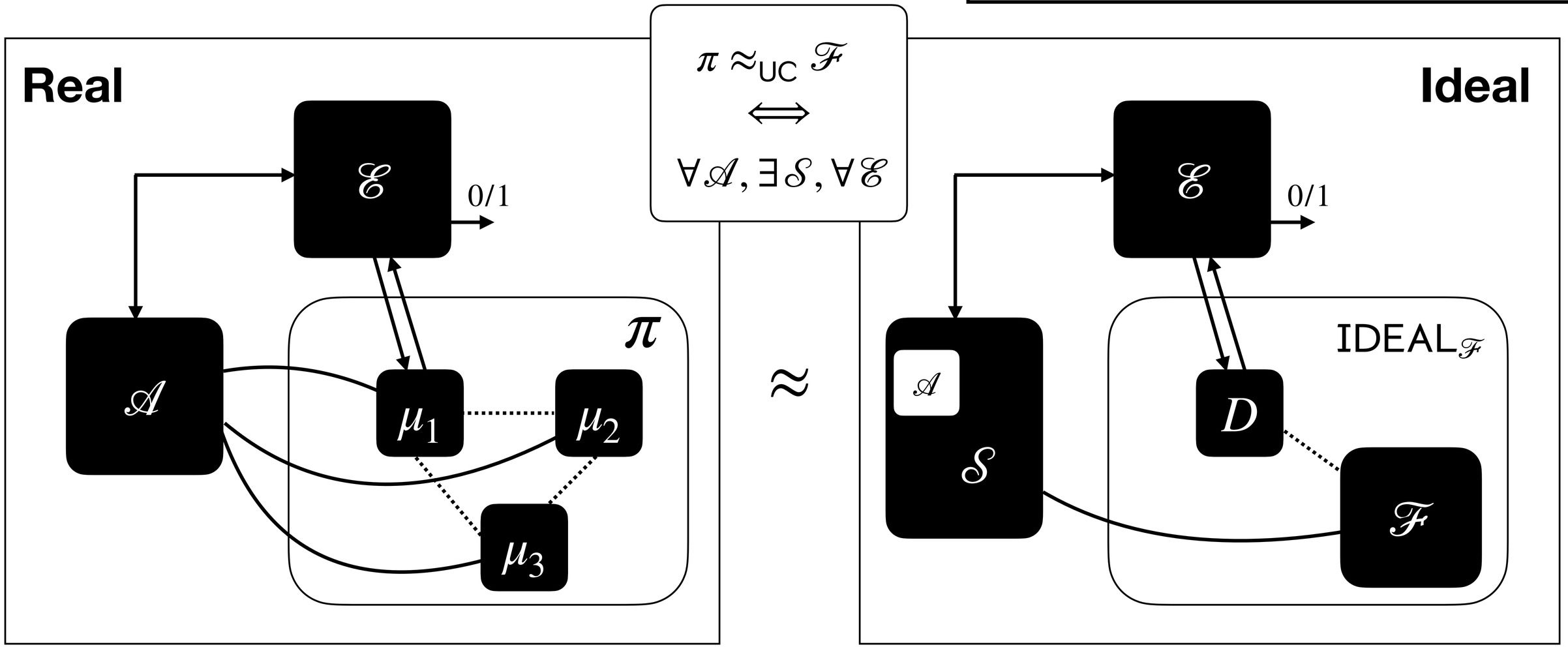


UC Security II

Goal: Cannot distinguish protocol from idealized version.

π : protocol
$\mathcal{E}$: environment
$\mathcal{F}$: ideal functionality
$\mathcal{A}$: adversary
D : dummy party
$\mathcal{S}$: simulator

$$\pi \approx_{\text{UC}} \mathcal{F} \iff \forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{E}$$



GROM

[CDGLN18]

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

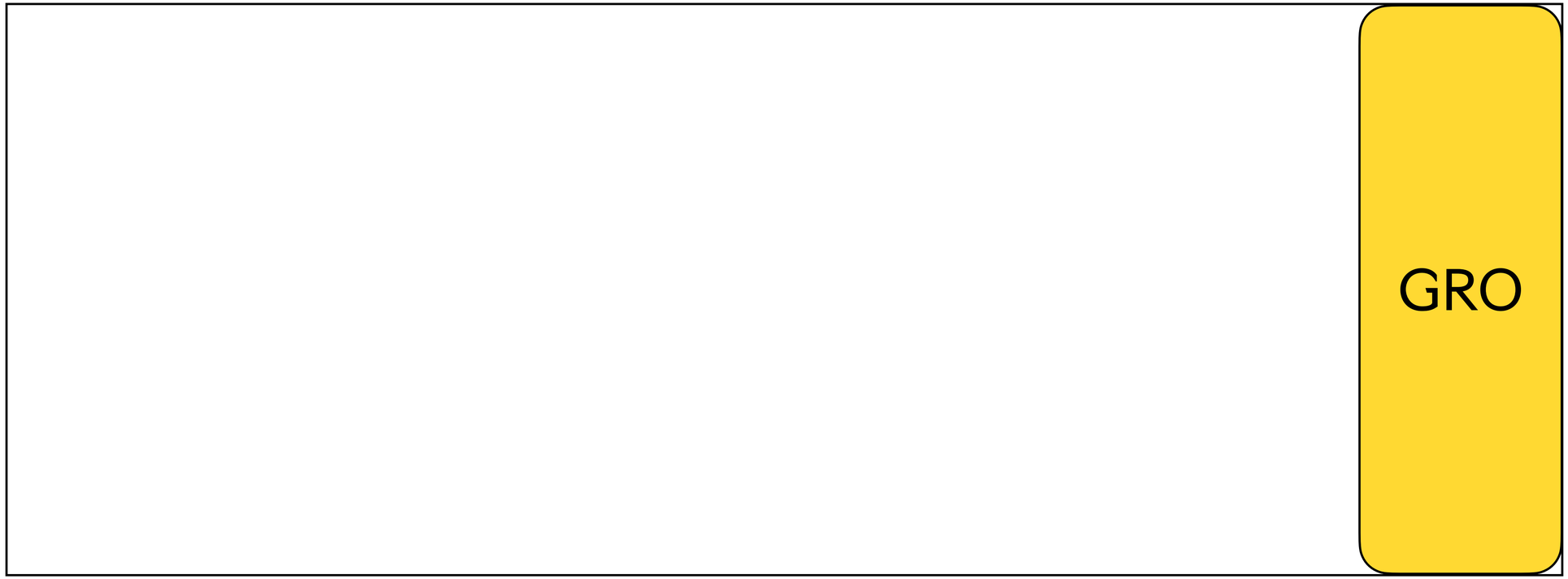
Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle



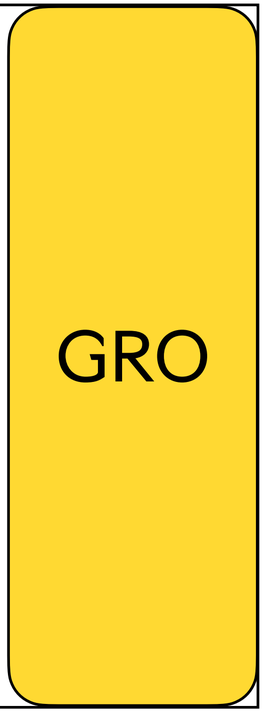
GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

- Query(x): as in ROM



GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

- Query(x): as in ROM
- Observe(s): get all queries with prefix s from adversary or from parties with sid $\neq s$

GRO

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

- Query(x): as in ROM
- Observe(s): get all queries with prefix s from adversary or from parties with sid $\neq s$
- Program(x, y): Program the GRO (maintaining consistency)

GRO

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

- Query(x): as in ROM
- Observe(s): get all queries with prefix s from adversary or from parties with sid $\neq s$
- Program(x, y): Program the GRO (maintaining consistency)
- IsProgrammed(x): allows parties in session sid to check if a $x = \text{sid} \circ x'$ has been programmed

GRO

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

- Query(x): as in ROM
- Observe(s): get all queries with prefix s from adversary or from parties with sid $\neq s$
- Program(x, y): Program the GRO (maintaining consistency)
- IsProgrammed(x): allows parties in session sid to check if a $x = \text{sid} \circ x'$ has been programmed

GRO

GROM

[CDGLN18]

Goal: ROM-like interface shared by **all** parties in the security experiment

Flavor: **r**estricted **p**rogrammable and **o**bservable **g**lobal **r**andom **o**racle

- Query(x): as in ROM
- Observe(s): get all queries with prefix s from adversary or from parties with sid $\neq s$
- Program(x, y): Program the GRO (maintaining consistency)
- IsProgrammed(x): allows parties in session sid to check if a $x = \text{sid} \circ x'$ has been programmed

GRO

Crucial: Simulator can program points without being detected!

Argument functionality [LR22]

Argument functionality [LR22]

- Model a zkSNARK as an ideal functionality.

Argument functionality [LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).

Argument functionality [LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.

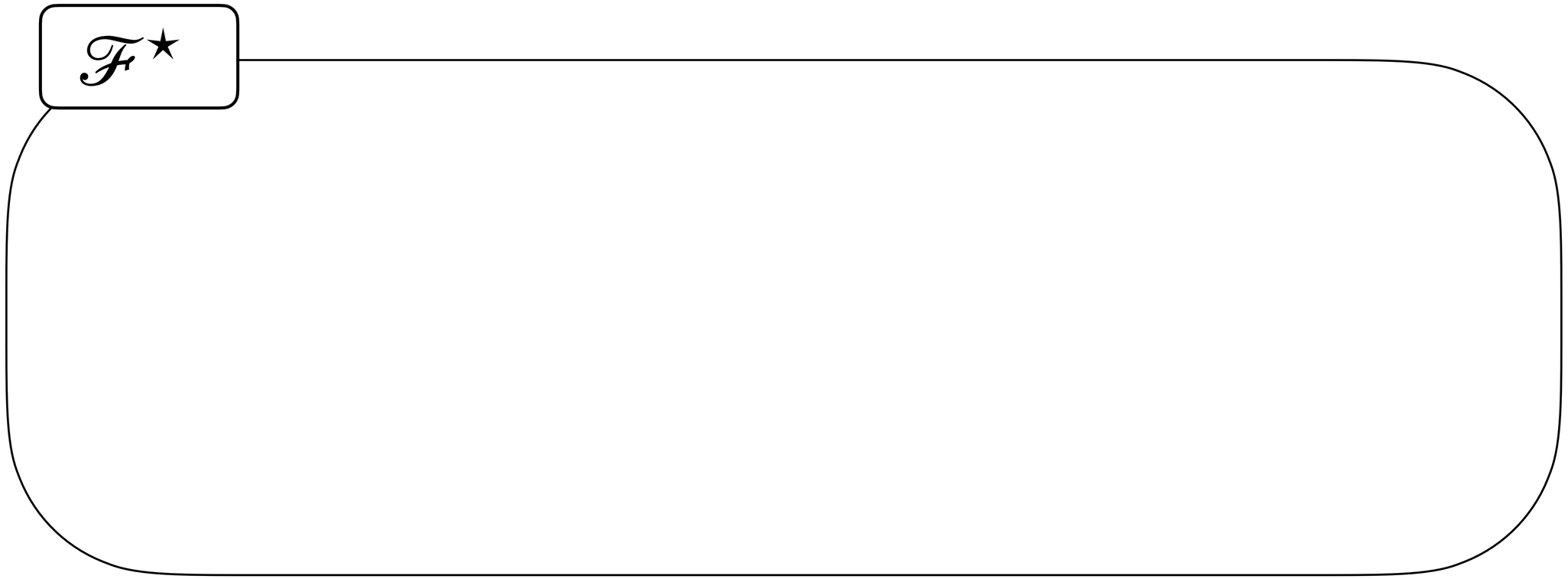
Argument functionality [LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.

Argument functionality

[LR22]

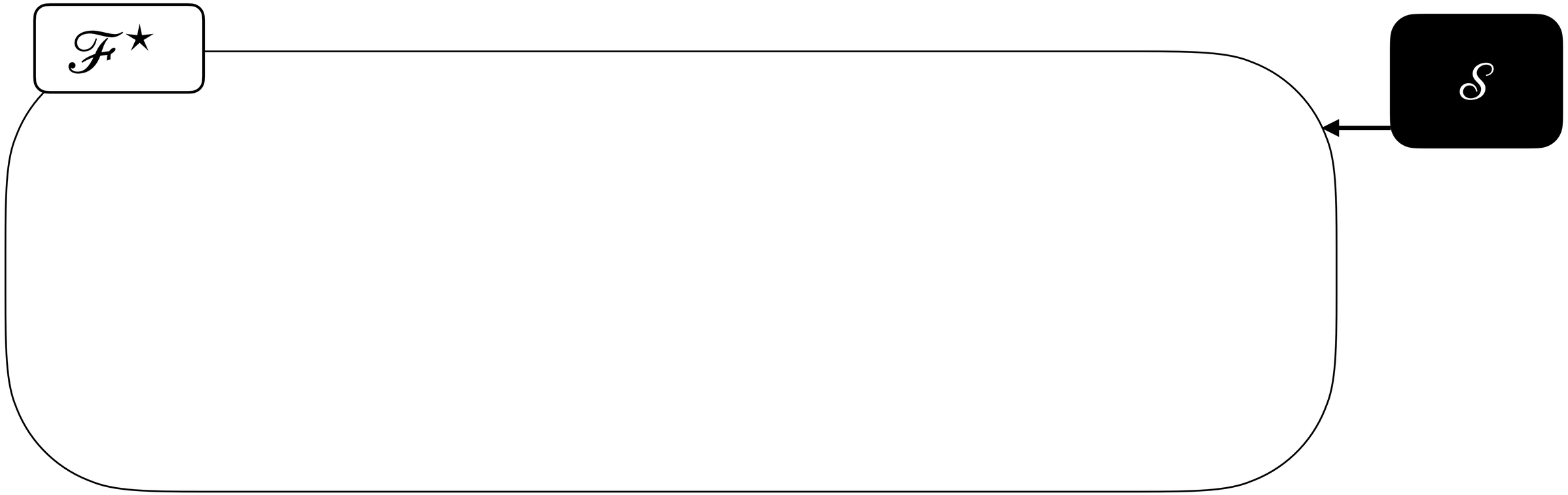
- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



Argument functionality

[LR22]

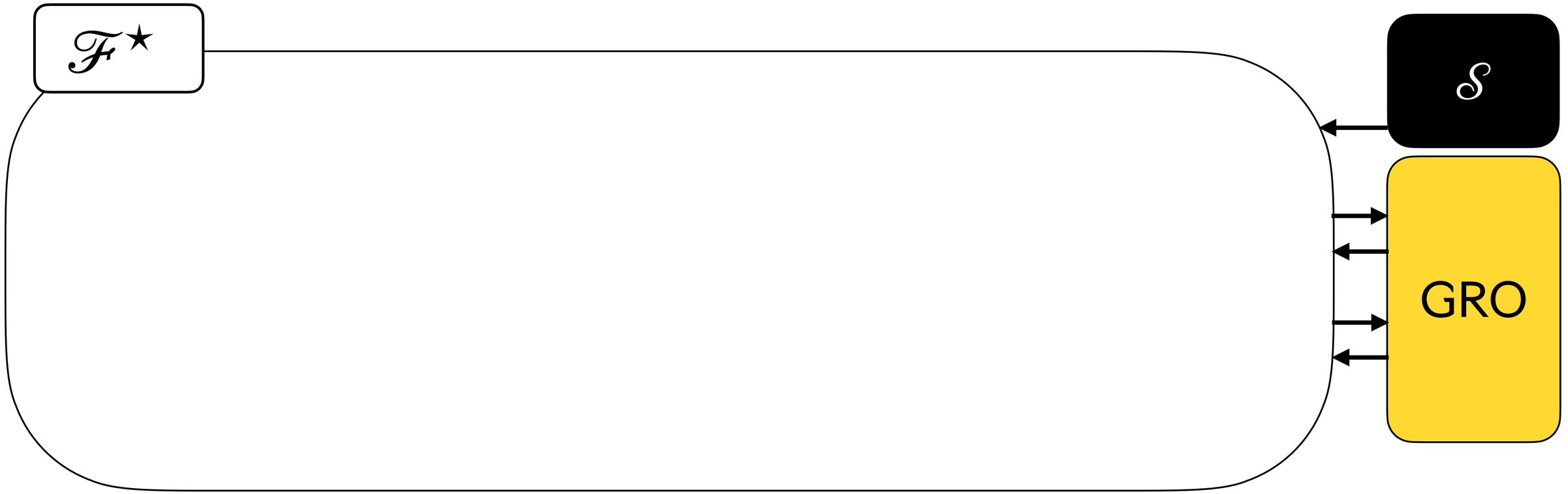
- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



Argument functionality

[LR22]

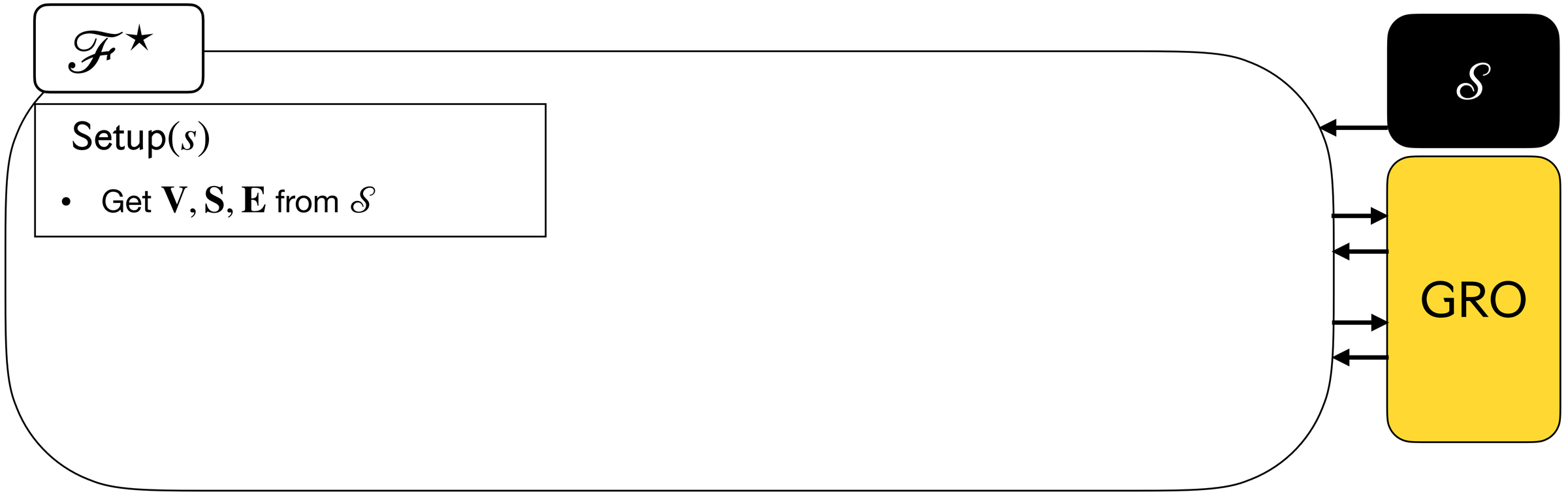
- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



Argument functionality

[LR22]

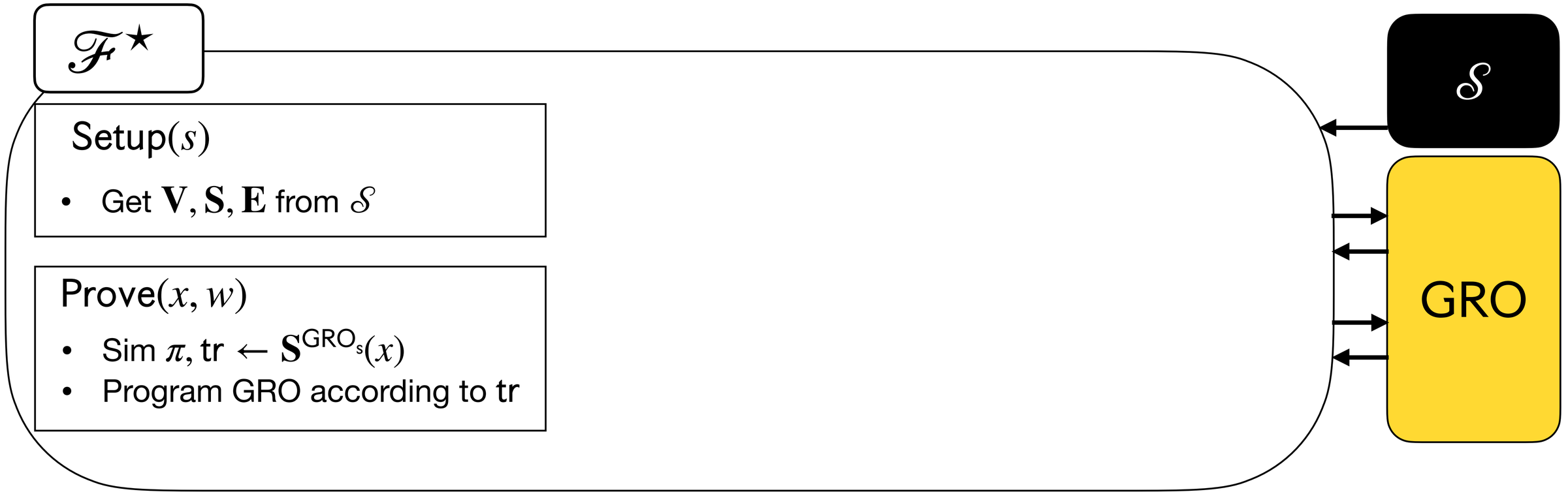
- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



Argument functionality

[LR22]

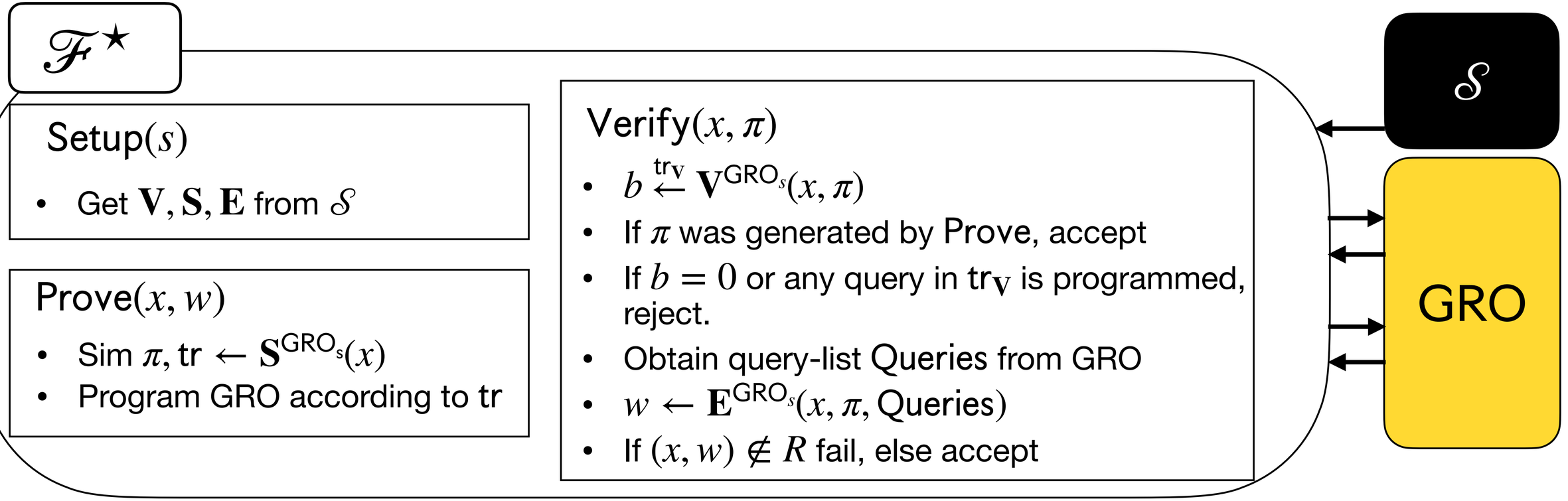
- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



Argument functionality

[LR22]

- Model a zkSNARK as an ideal functionality.
- Prover generates simulated proofs (without using the witness).
- Verifier aims to extract a witness from each accepting proof.
- Proofs generated in Prove are always accepted by Verify.



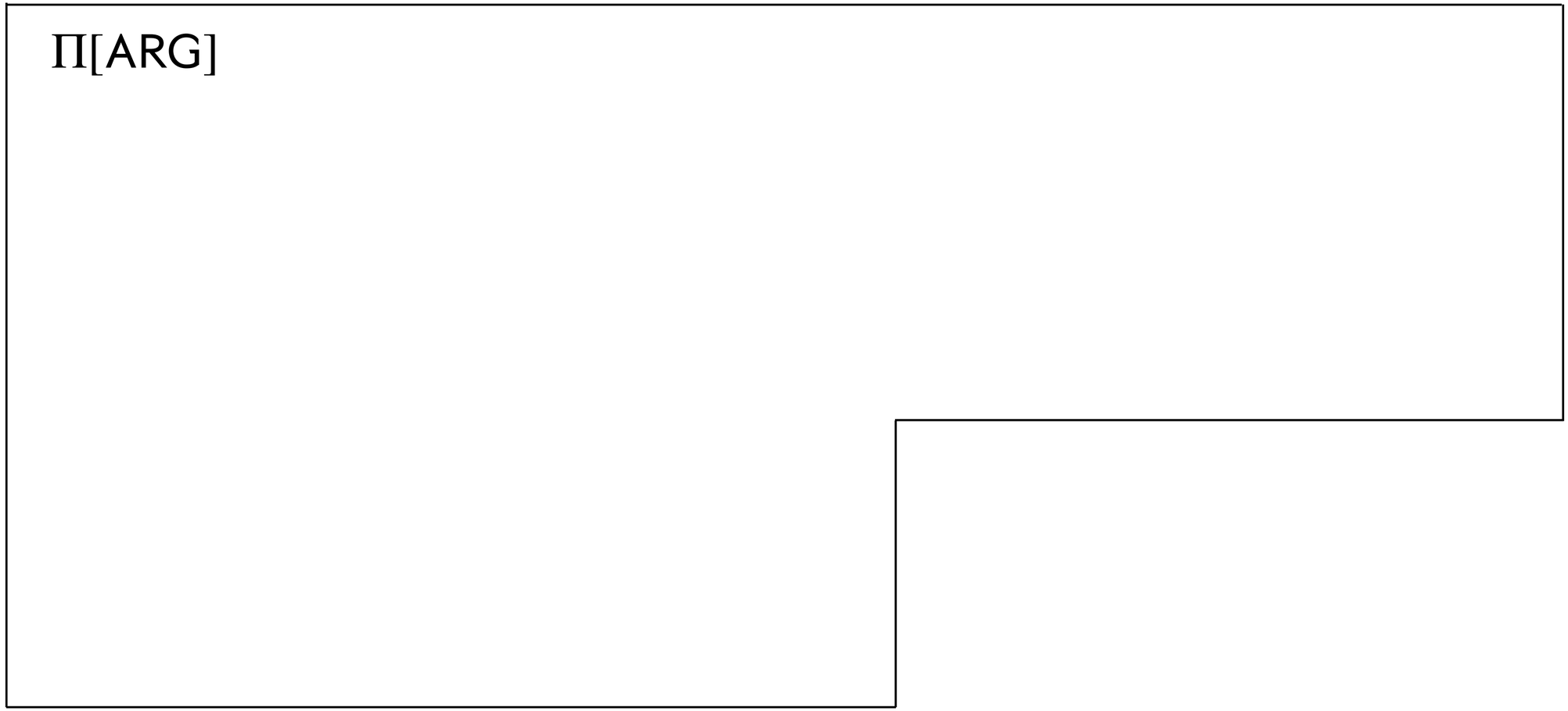
Wrapper protocol

Wrapper protocol

Converts an argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM

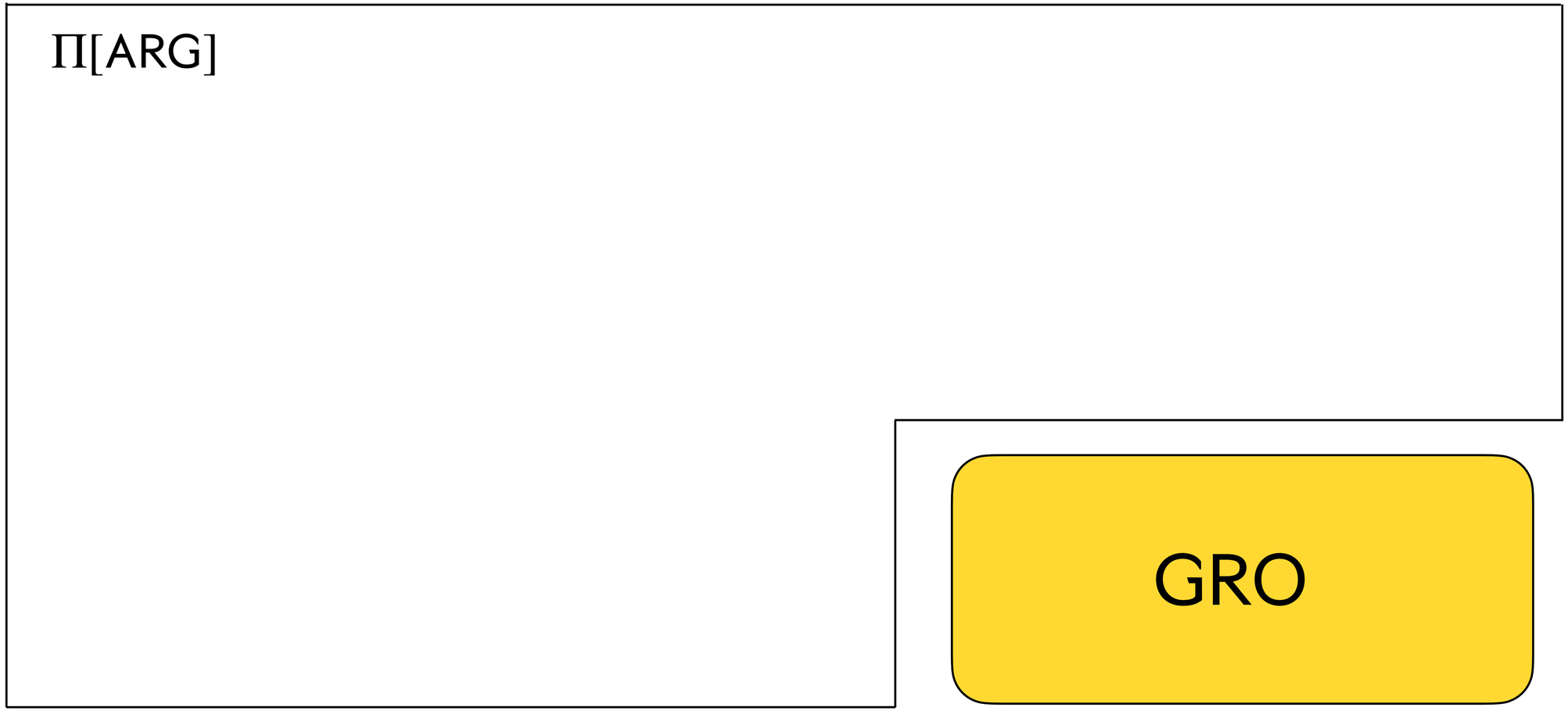
Wrapper protocol

Converts an argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM



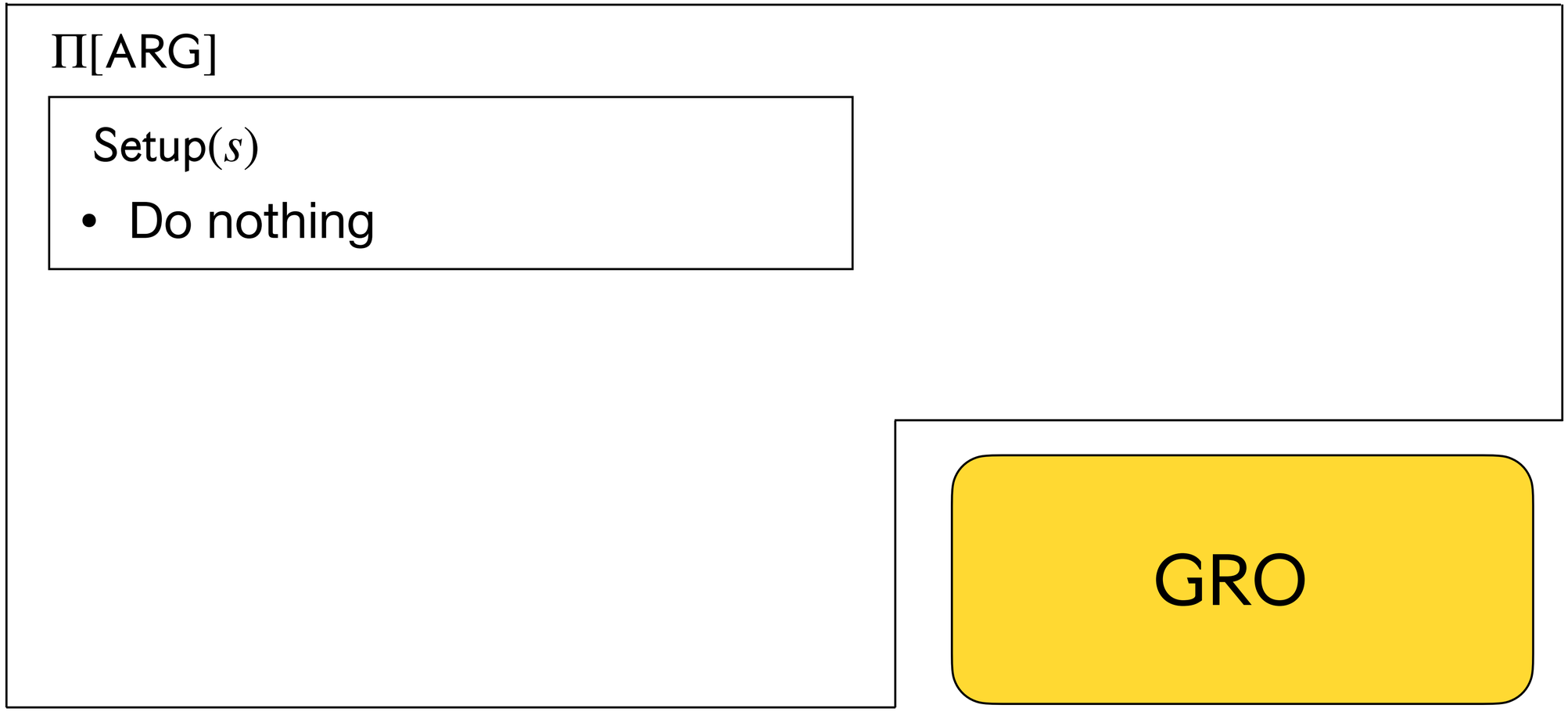
Wrapper protocol

Converts an argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM



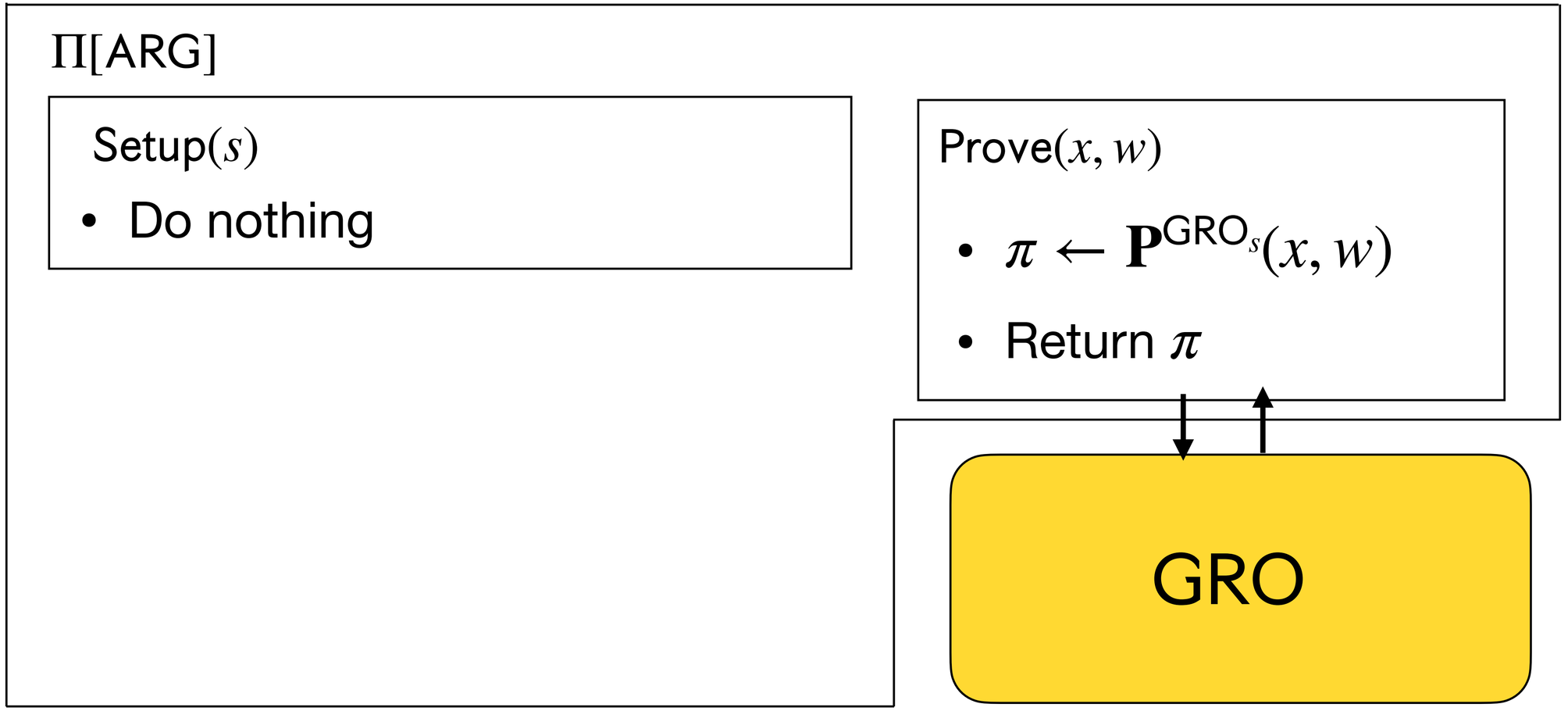
Wrapper protocol

Converts an argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM



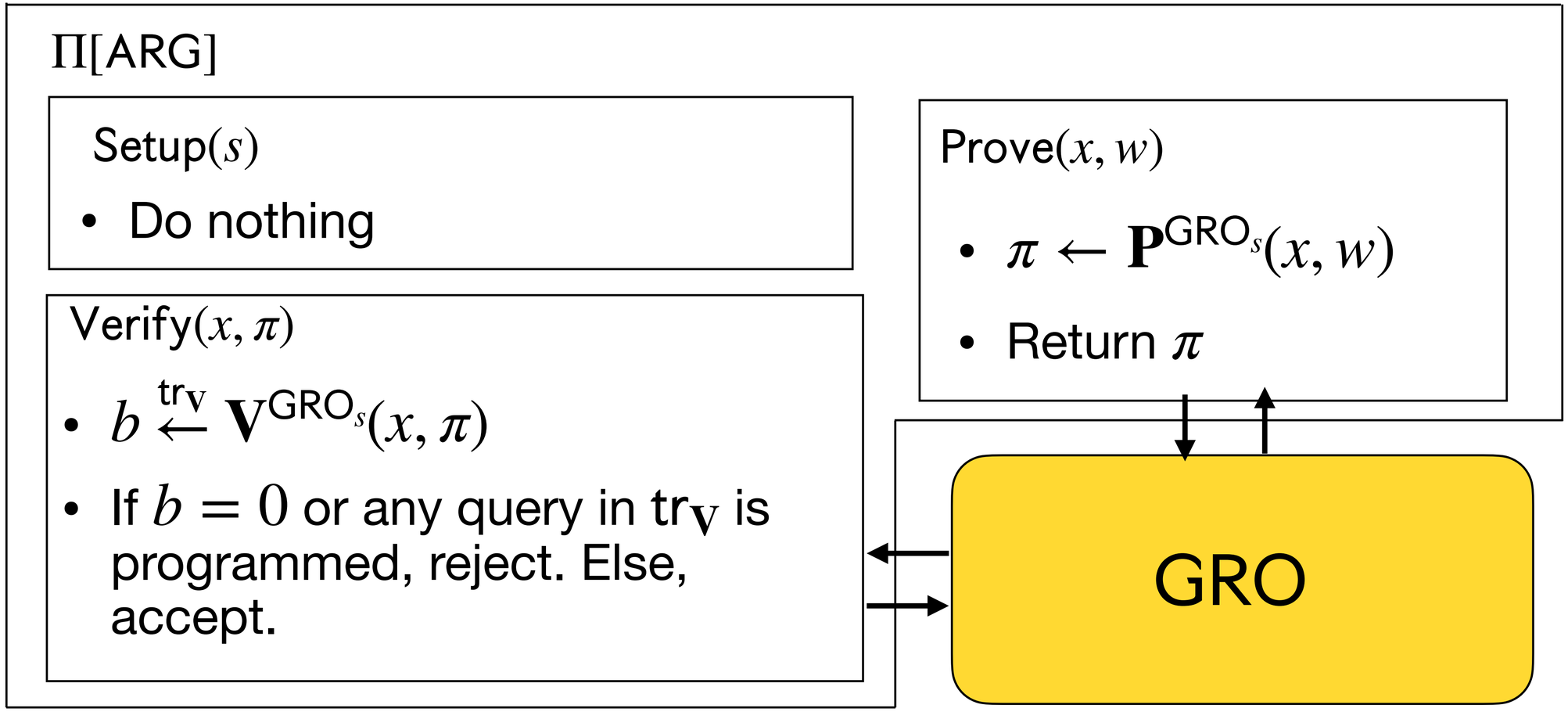
Wrapper protocol

Converts an argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM



Wrapper protocol

Converts an argument $\mathsf{ARG} = (\mathbf{P}, \mathbf{V})$ in the ROM into a protocol in the GROM



Recap and Goal

Recap and Goal

Find an ARG in the ROM such that

Recap and Goal

Find an ARG in the ROM such that

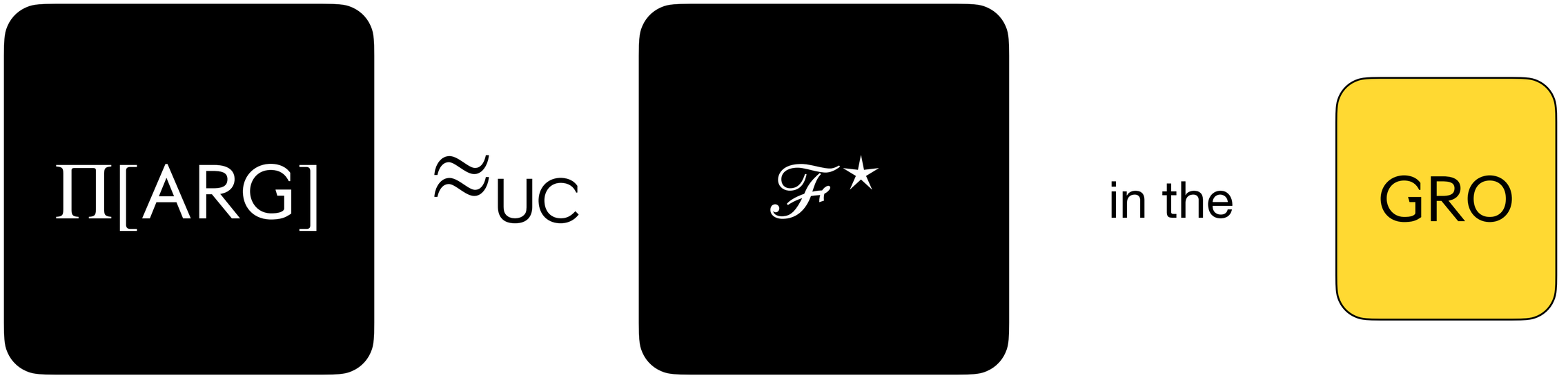
Recap and Goal

Find an ARG in the ROM such that

in the

Recap and Goal

Find an ARG in the ROM such that



Our results

Main Thm.

There exists a zkSNARK that is unconditionally UC-secure in the GROM

Lemma

Let ARG *be a "UC-friendly" argument in the ROM.*

Then, Π[ARG] *is UC-secure in the GROM*

Theorem

The Micali construction is "UC-friendly" in the ROM, provided that the underlying PCP is honest-verifier zero knowledge and knowledge sound.

Corollary

The Micali construction is UC-secure in the GROM, when instantiated as above.

Theorem

*The **BCS** construction is "UC-friendly" in the ROM, provided that the underlying **IOP** is honest-verifier zero knowledge and **(state-restoration)** knowledge sound.*

Corollary

*The **BCS** construction is UC-secure in the GROM, when instantiated as above.*

Techniques

Modelling shared functionalities

[BCHTZ22]

Modelling shared functionalities

[BCHTZ22]

Plain UC security **not enough** for **shared** setups

Modelling shared functionalities

[BCHTZ22]

Plain UC security **not enough** for **shared** setups

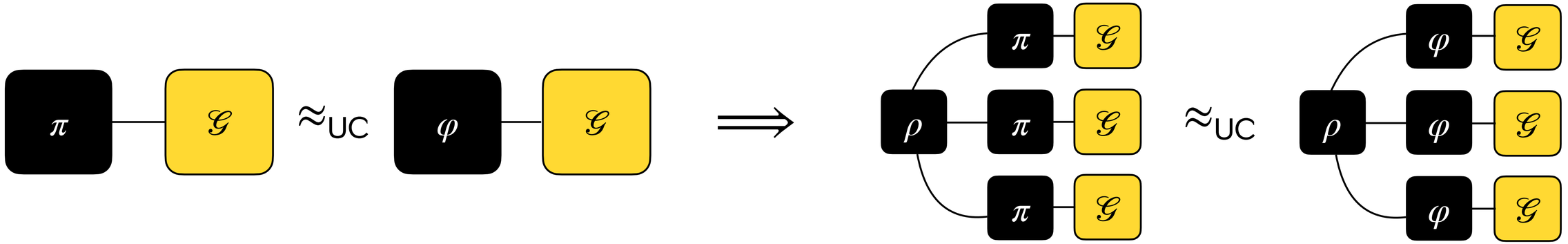
Plain UC:

Modelling shared functionalities

[BCHTZ22]

Plain UC security **not enough** for **shared** setups

Plain UC:

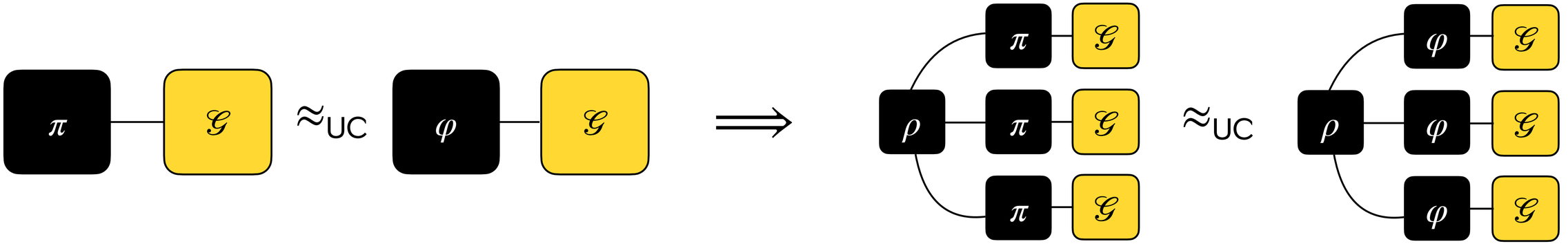


Modelling shared functionalities

[BCHTZ22]

Plain UC security **not enough** for **shared** setups

Plain UC:



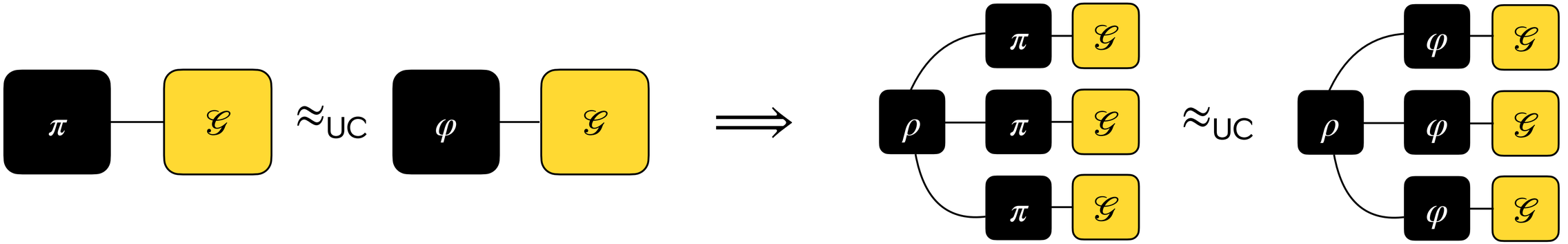
Solution: UC with Global Subroutines!

Modelling shared functionalities

[BCHTZ22]

Plain UC security **not enough** for **shared** setups

Plain UC:



Solution: UC with Global Subroutines!

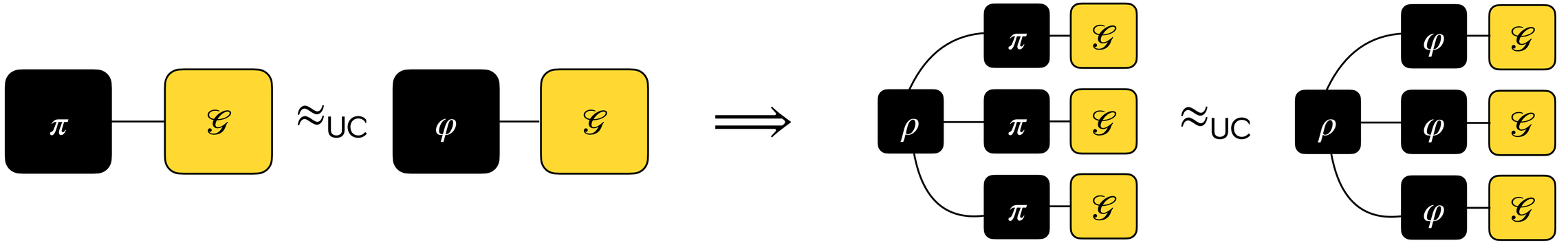
UCGS:

Modelling shared functionalities

[BCHTZ22]

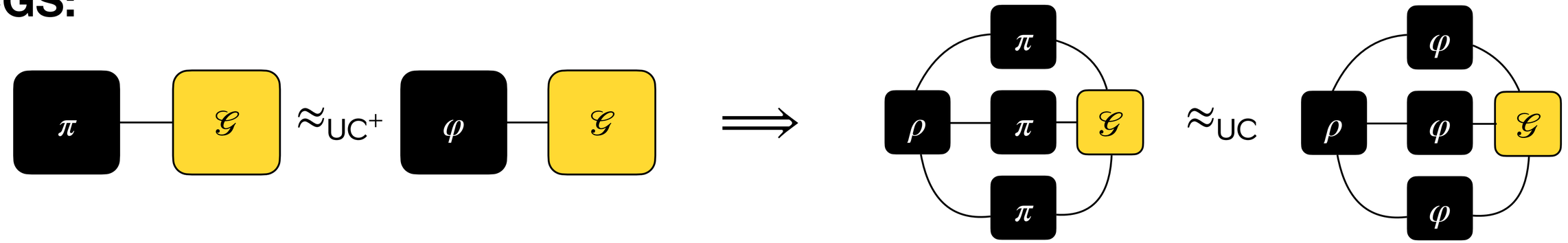
Plain UC security **not enough** for **shared** setups

Plain UC:



Solution: UC with Global Subroutines!

UCGS:

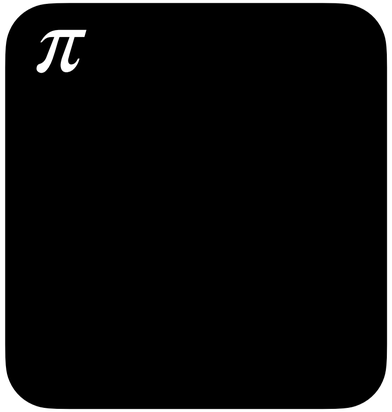


UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import

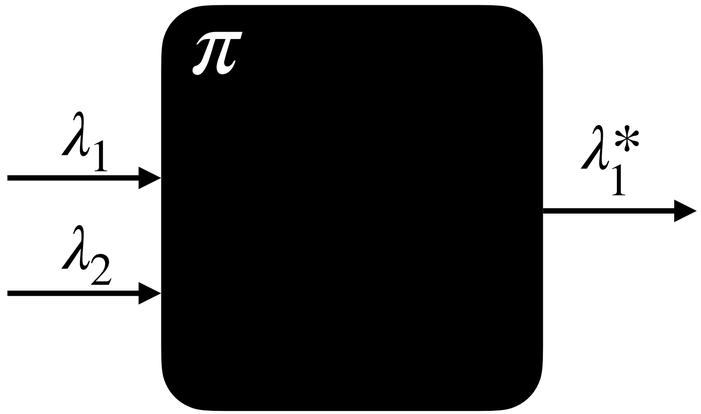
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import



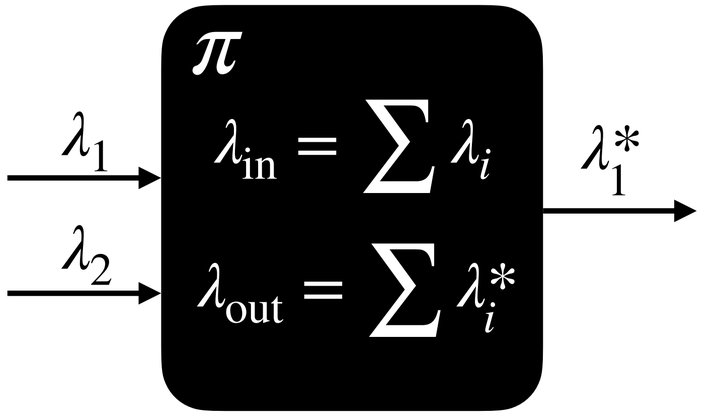
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using <u>import</u>



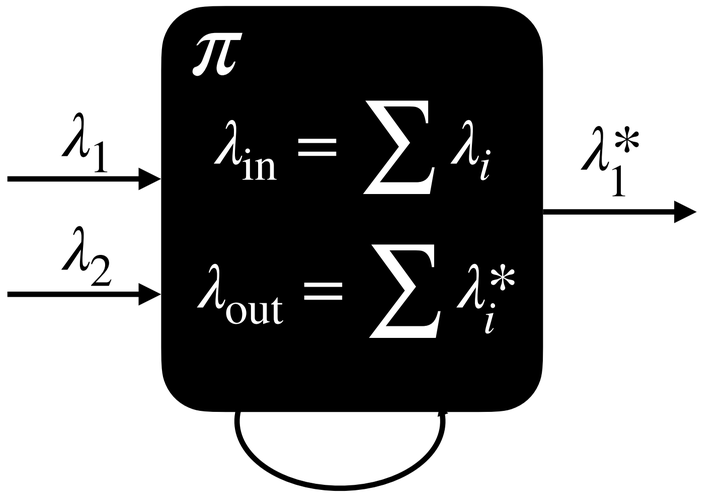
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import



UC with Budgets

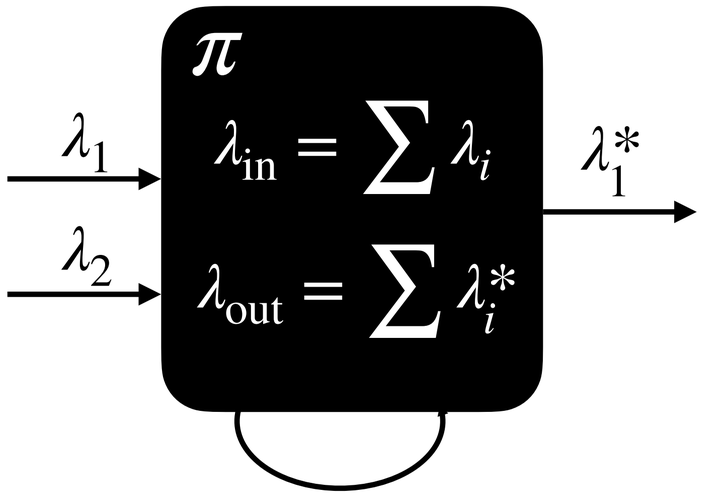
Plain UC only models adversaries that are **computationally** bounded using import



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import

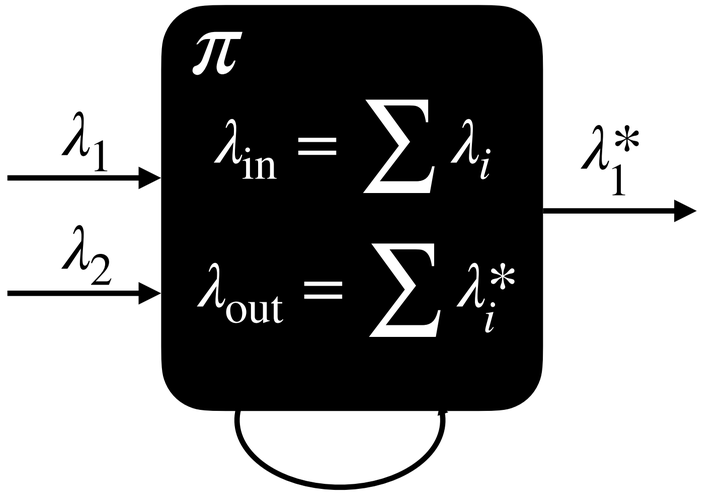


$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing budgets

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import

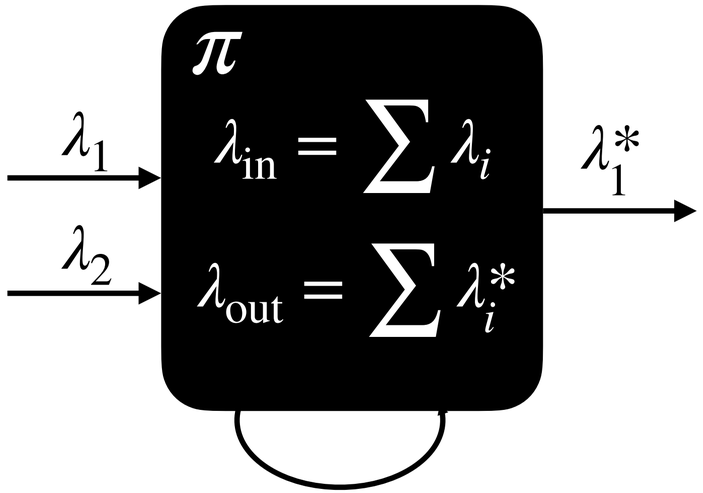


$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing budgets

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using <u>import</u>

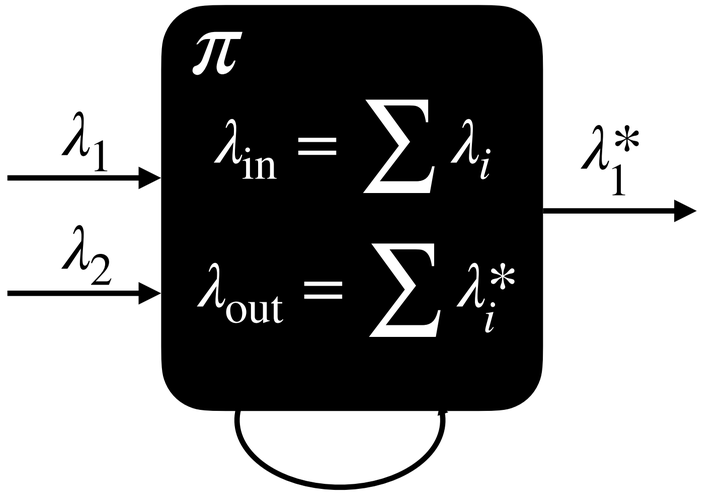


$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing <u>budgets</u>

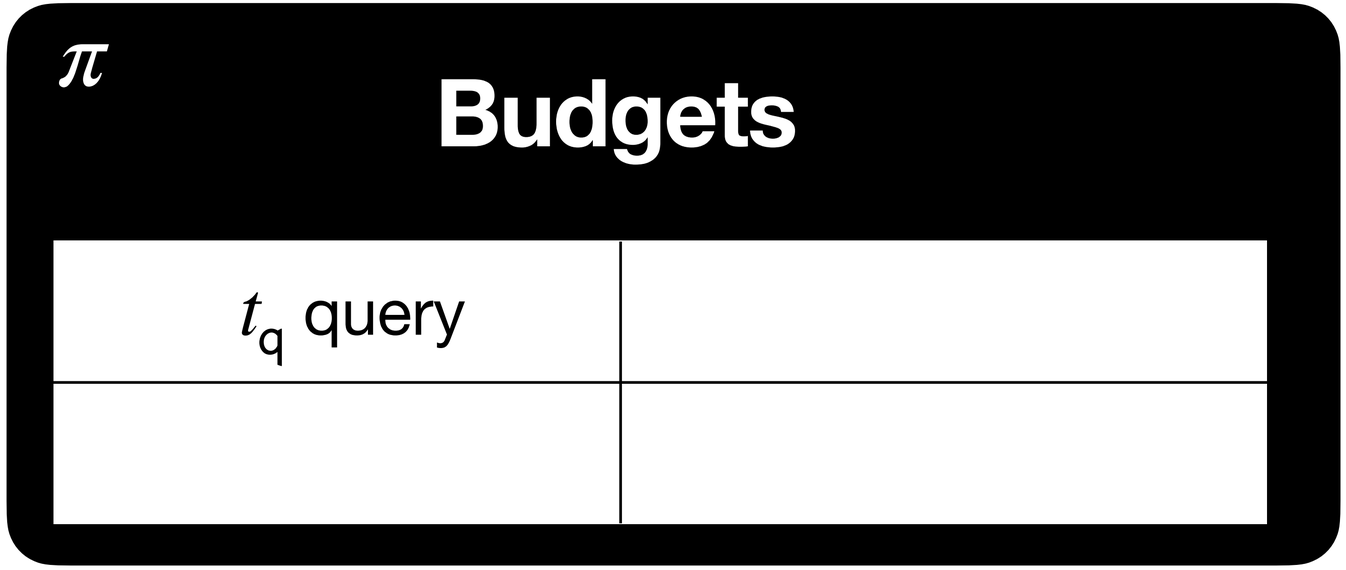
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import



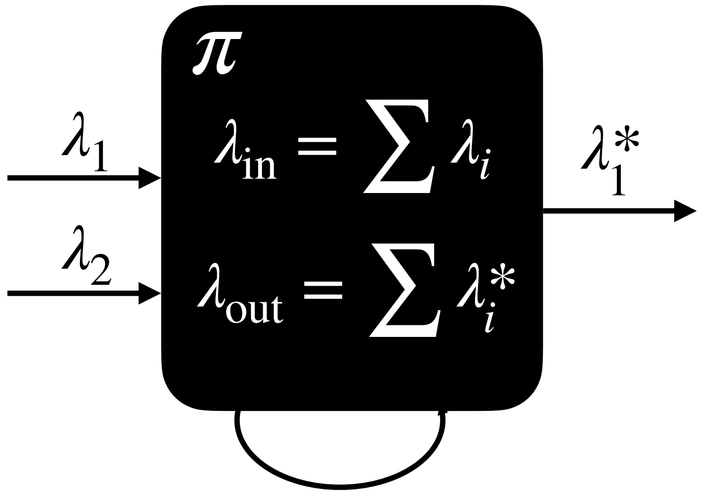
$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing budgets



UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using import



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing budgets

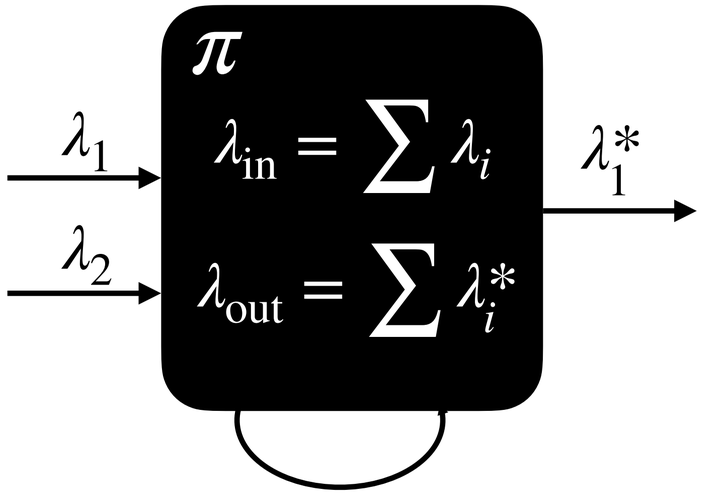
π

Budgets

t_{q} query	
t_{p} programming	

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using <u>import</u>



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing <u>budgets</u>

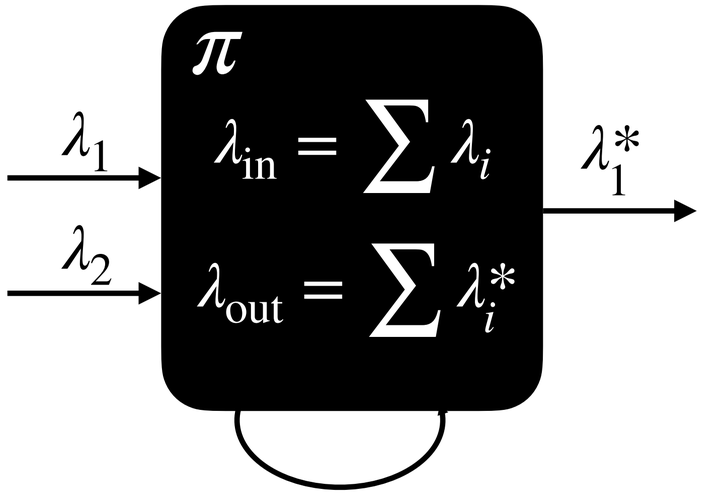
π

Budgets

t_{q} query	ℓ_{p} proving
t_{p} programming	

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using <u>import</u>



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing <u>budgets</u>

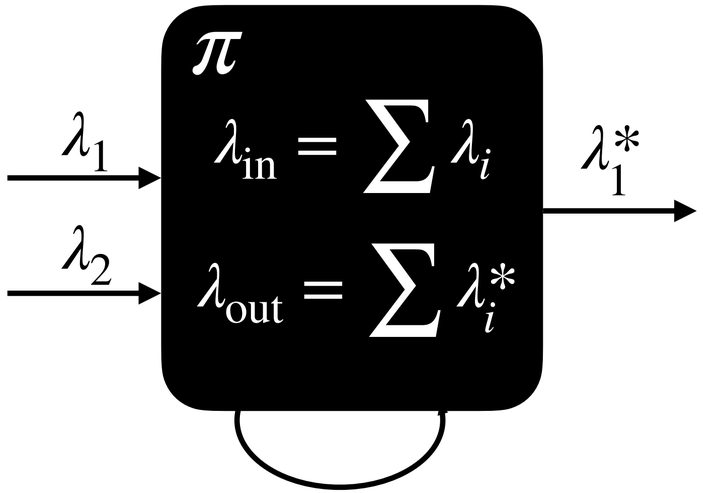
π

Budgets

t_{q} query	ℓ_{p} proving
t_{p} programming	ℓ_{v} verification

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using <u>import</u>



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing <u>budgets</u>

π

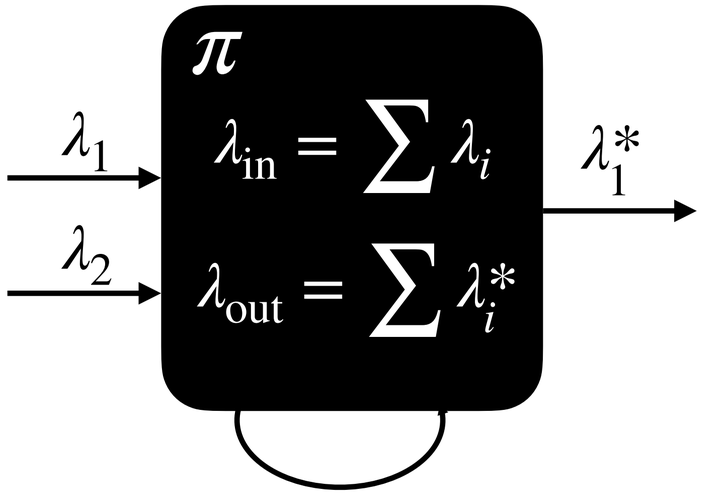
Budgets

t_{q} query	ℓ_{p} proving
t_{p} programming	ℓ_{v} verification

Budget can then be spent on:

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded using <u>import</u>



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**. We model this introducing <u>budgets</u>

π

Budgets

t_{q} query	ℓ_{p} proving
t_{p} programming	ℓ_{v} verification

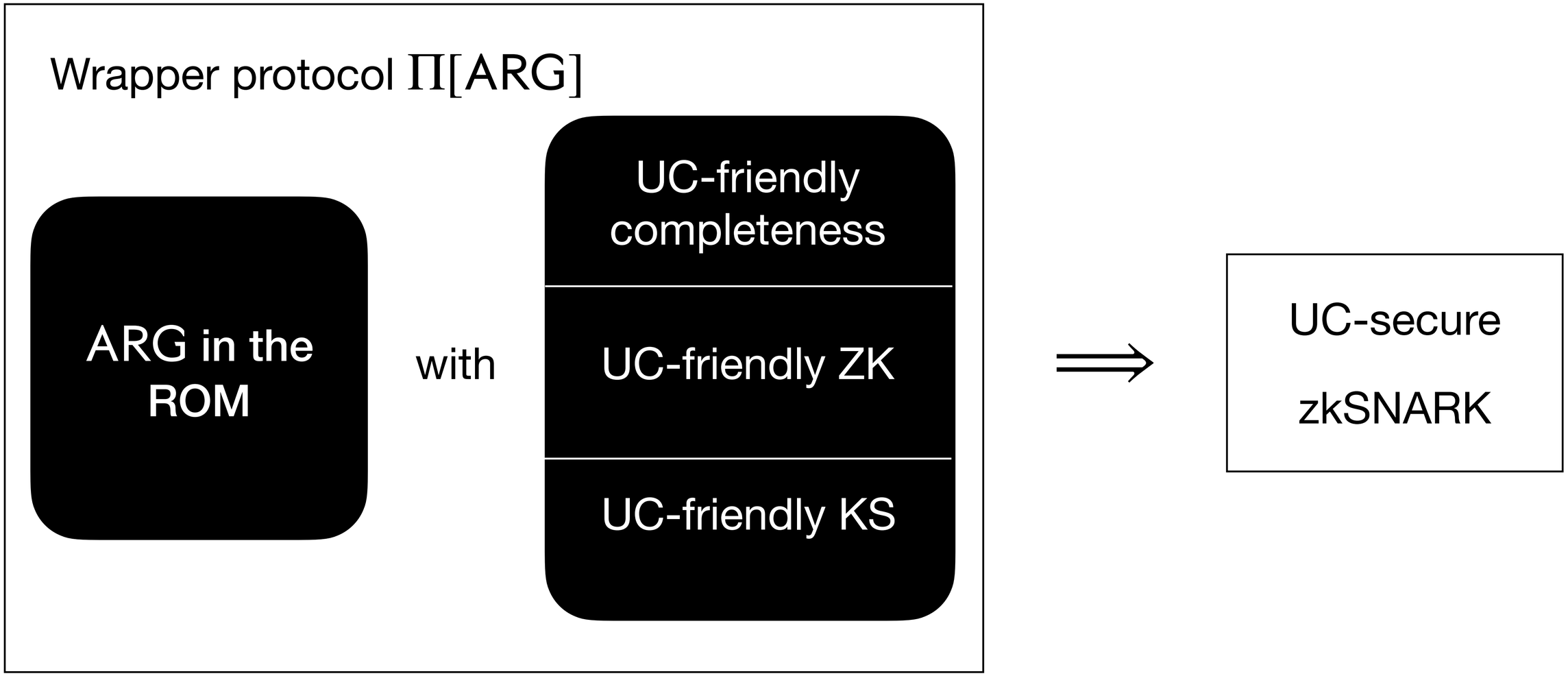
Budget can then be spent on:

GRO

Prove

Verify

Our main lemma



UC-friendly $\Longrightarrow$ UC-secure

UC-friendly $\Longrightarrow$ UC-secure

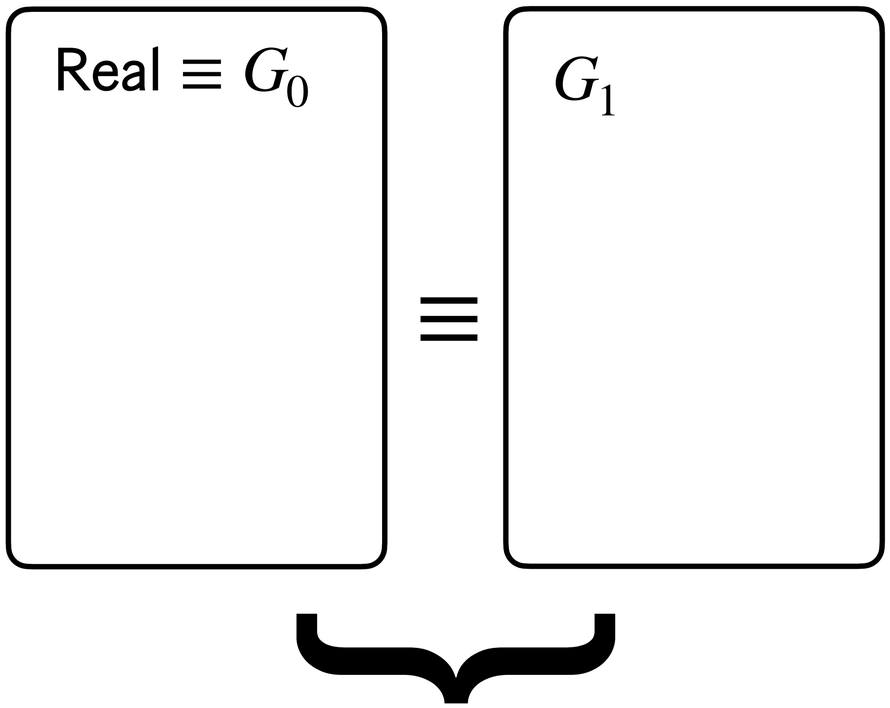
Real $\equiv G_0$

UC-friendly $\Longrightarrow$ UC-secure

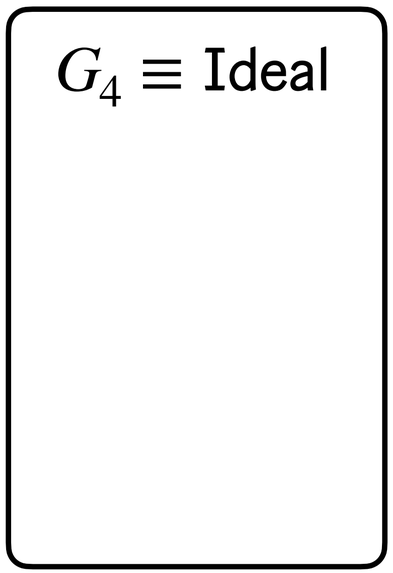
Real $\equiv G_0$

$G_4 \equiv$ Ideal

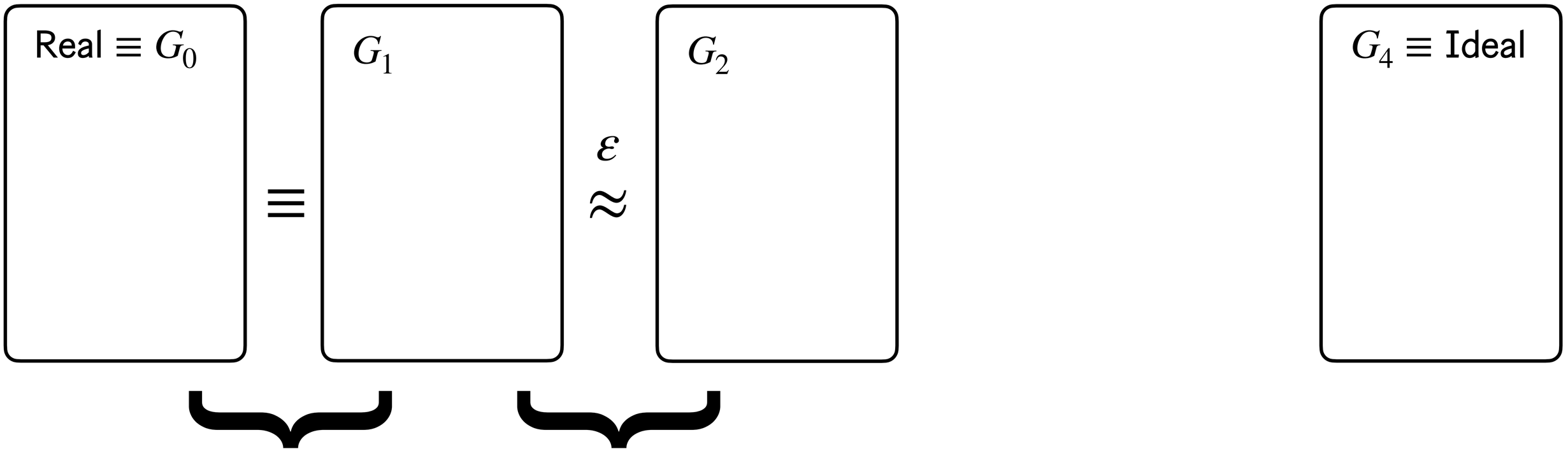
UC-friendly $\Longrightarrow$ UC-secure



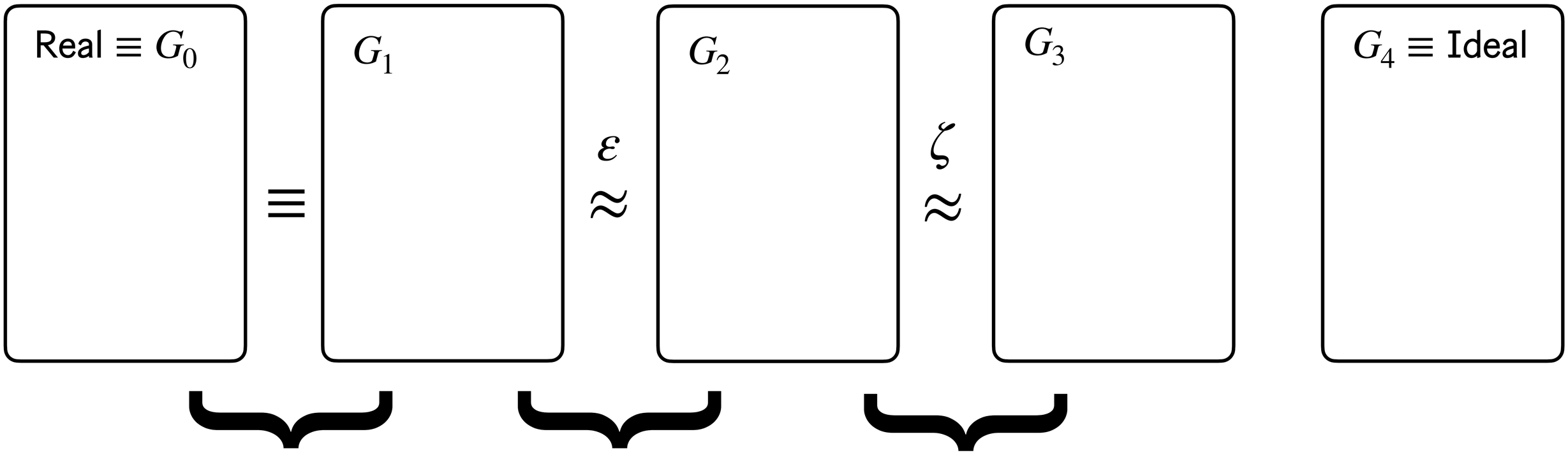
Simulator can program undetectably



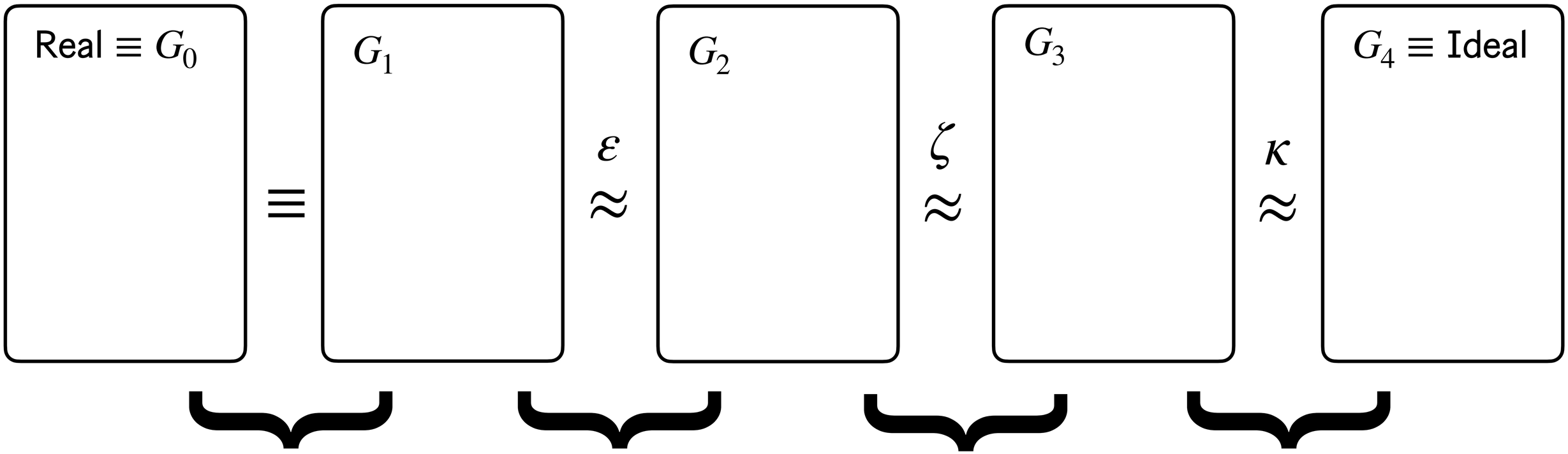
UC-friendly $\Longrightarrow$ UC-secure



UC-friendly $\Longrightarrow$ UC-secure

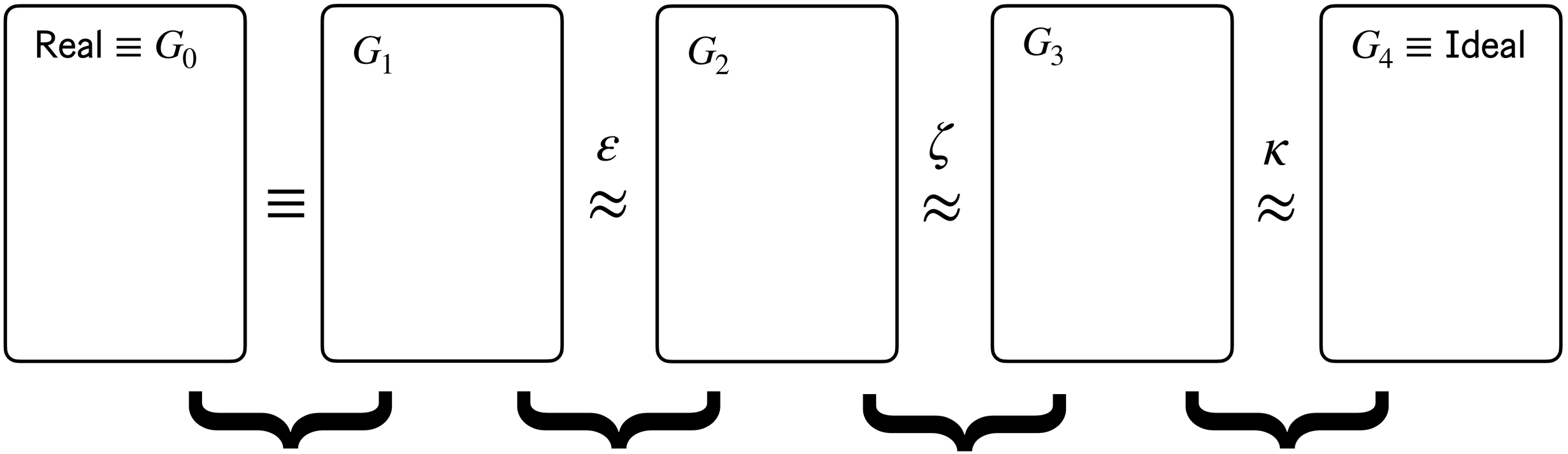


UC-friendly $\Longrightarrow$ UC-secure



UC-friendly $\Longrightarrow$ UC-secure

UC-friendly properties exactly defined for these game hops



Micali's construction I

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1253–1298

COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Micali's construction I

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1253–1298

COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Canonical construction of zkSNARK in the ROM

Micali's construction I

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1253–1298

COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Canonical construction of zkSNARK in the ROM

Straightline black-box extractor: compatible with UC!

Micali's construction I

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1253–1298

COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Canonical construction of zkSNARK in the ROM

Straightline black-box extractor: compatible with UC!

Proofs are **non-malleable**: also required for UC-security!

Micali's construction I

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1253–1298

COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Canonical construction of zkSNARK in the ROM

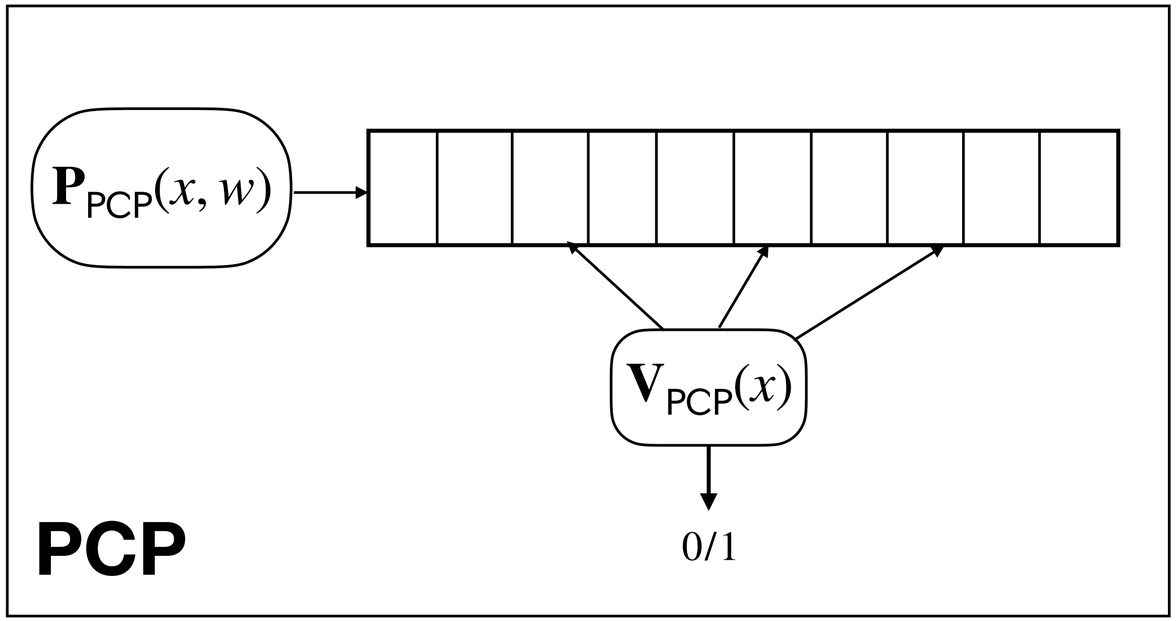
Straightline black-box extractor: compatible with UC!

Proofs are **non-malleable**: also required for UC-security!

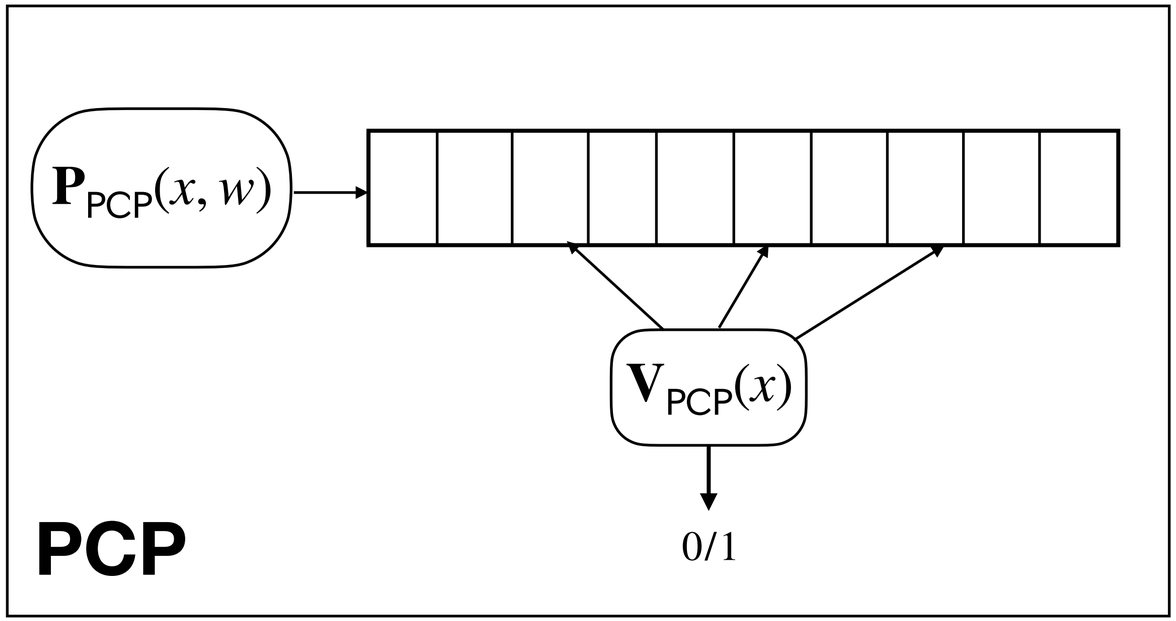
Stepping stone to BCS, which underlies **deployed** zkSNARKs

Micali’s construction II

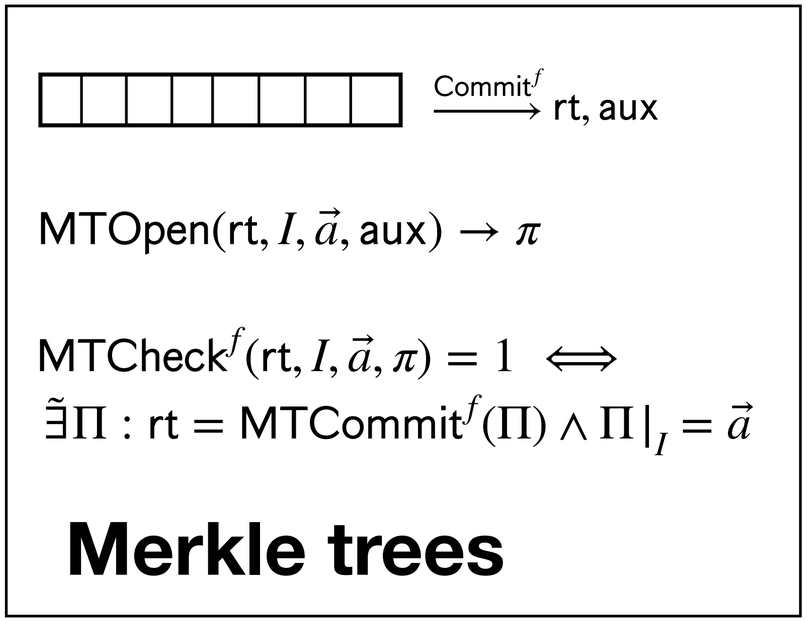
Micali's construction II



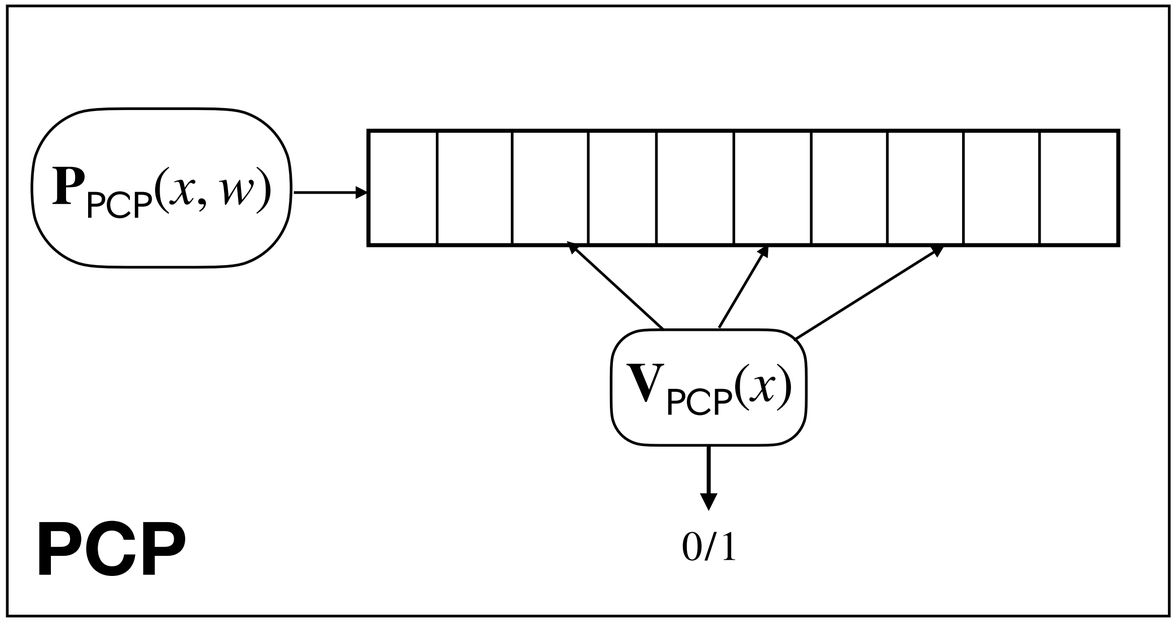
Micali's construction II



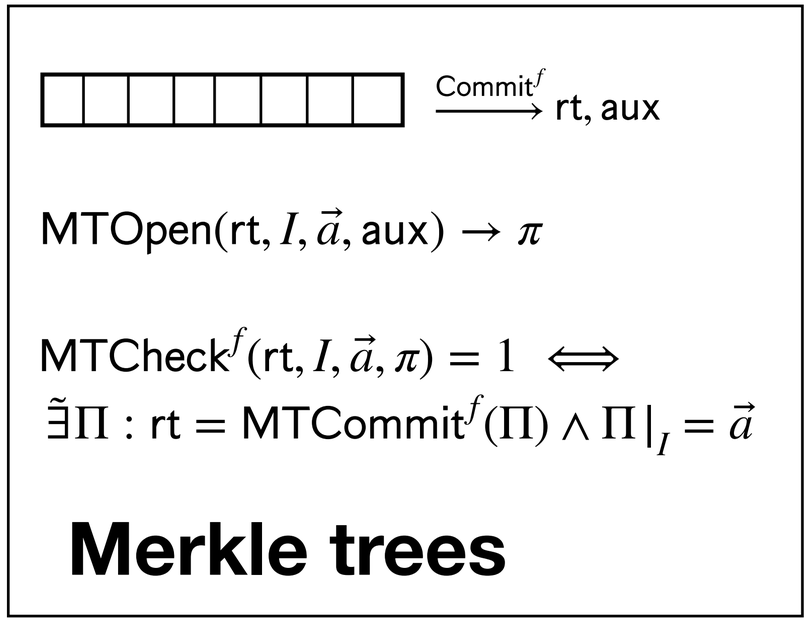
+



Micali's construction II



+

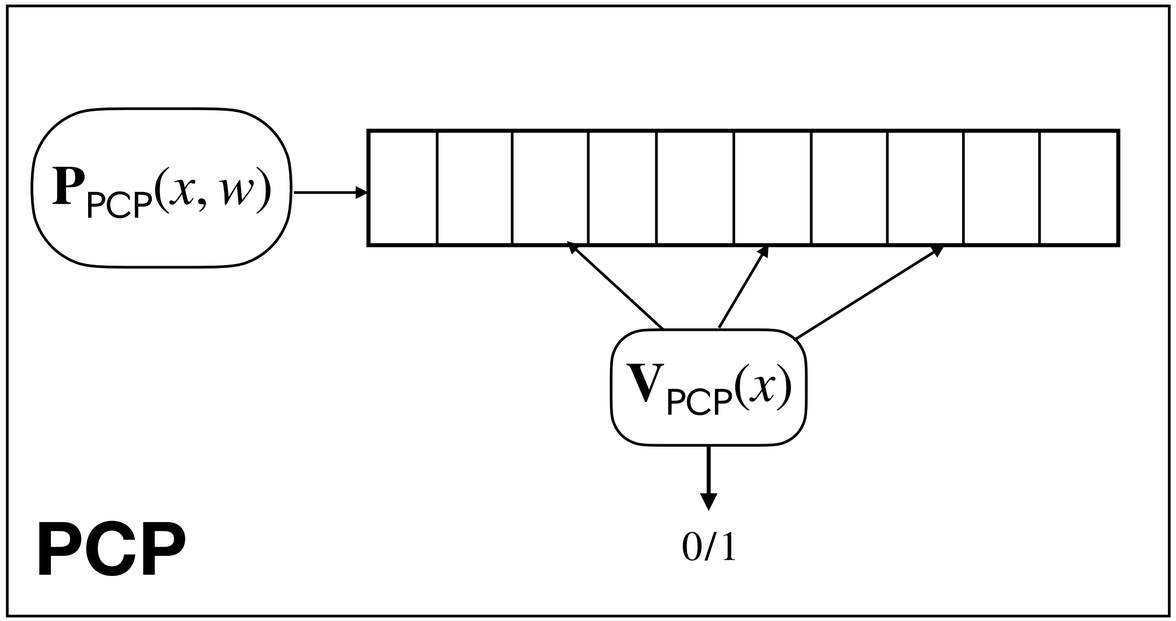


+

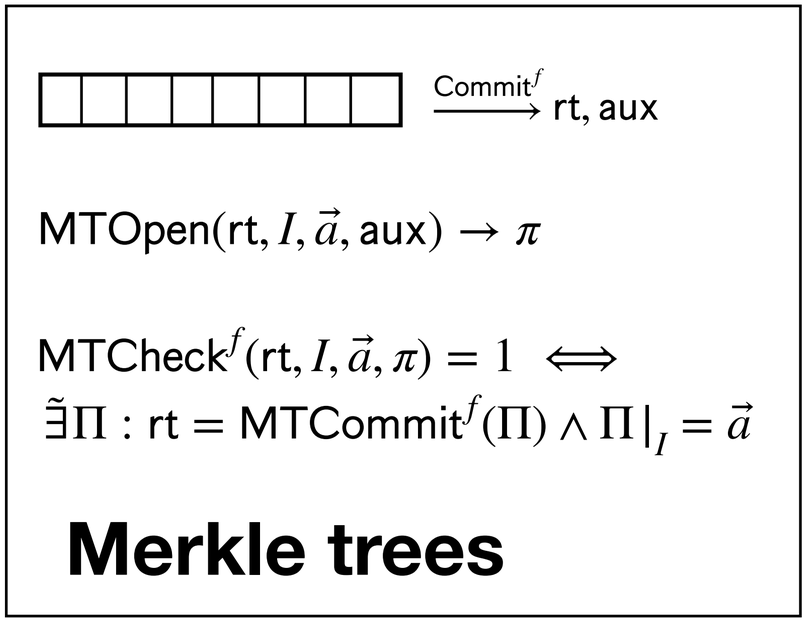
Fiat Shamir

=

Micali's construction II



+



+

Fiat Shamir

=

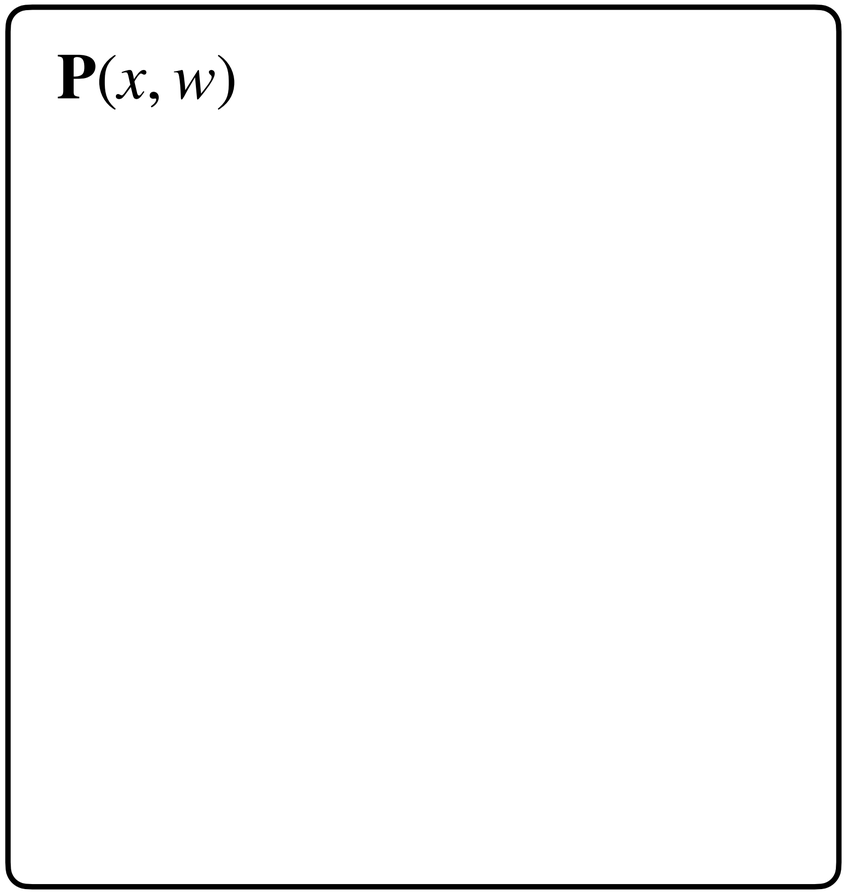
zkSNARK in the ROM

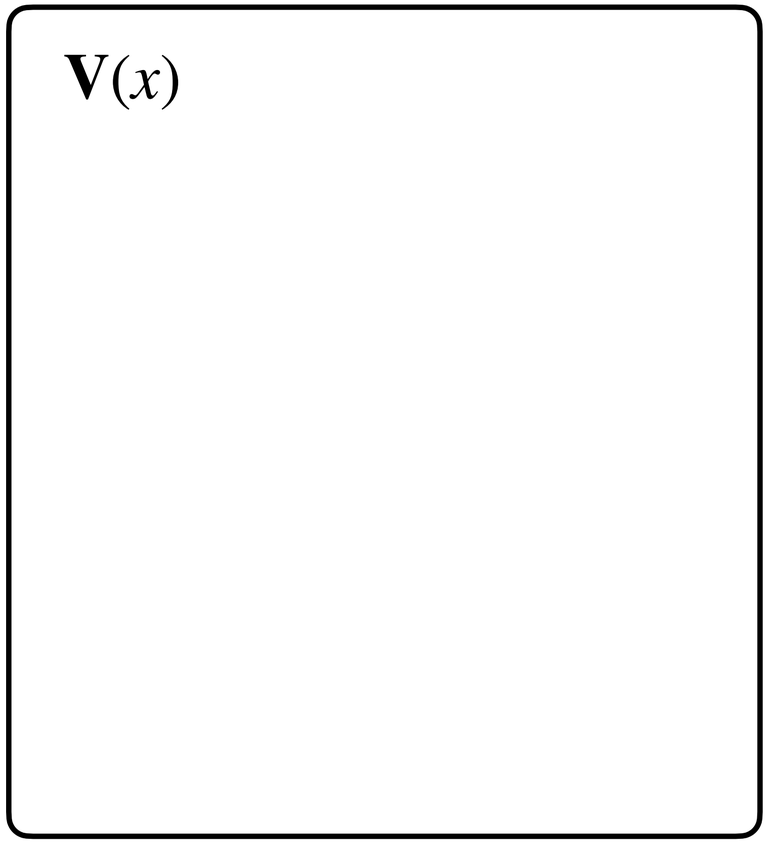
Micali’s construction III

Commit to PCP string using MT,
then apply FS transform

Micali's construction III

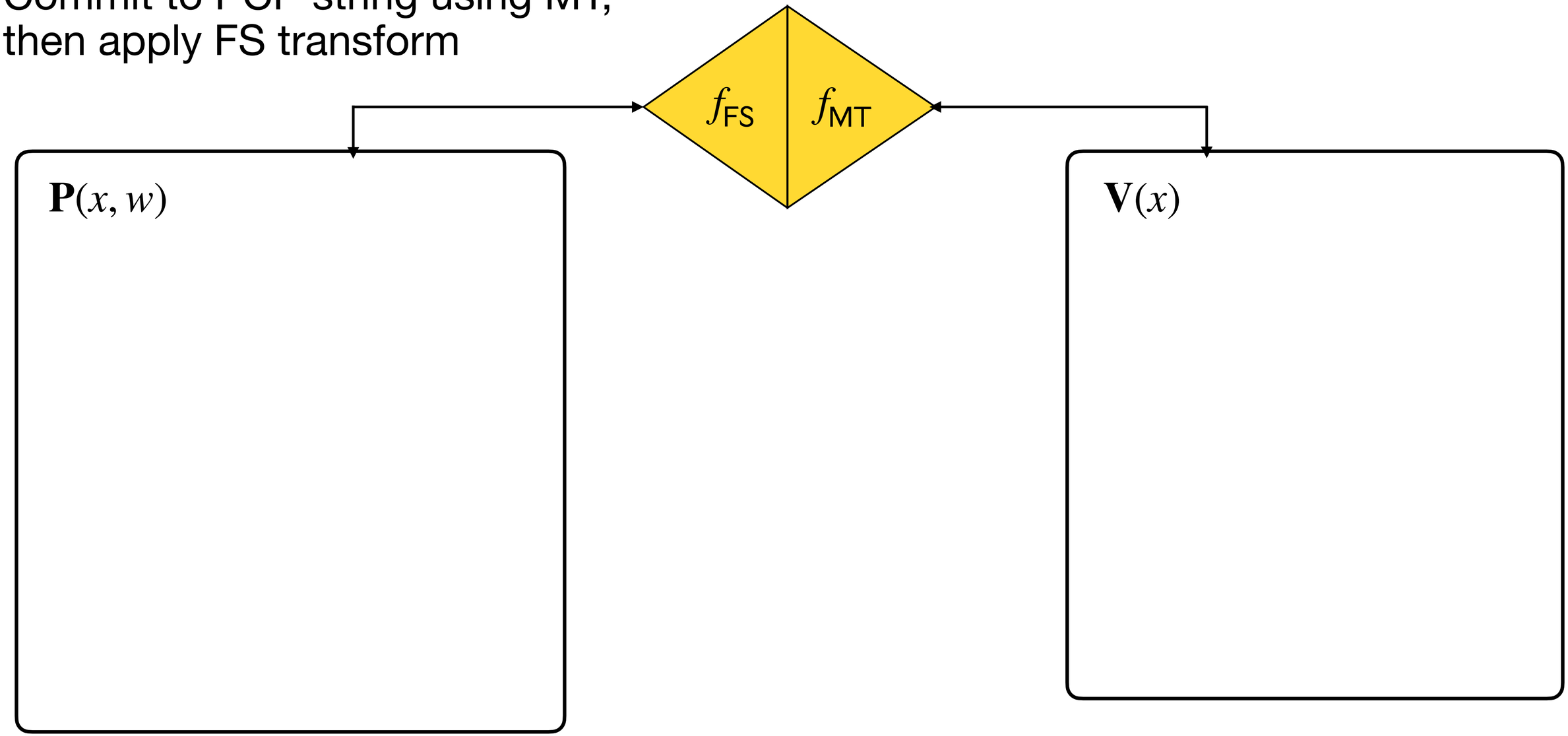
Commit to PCP string using MT,
then apply FS transform





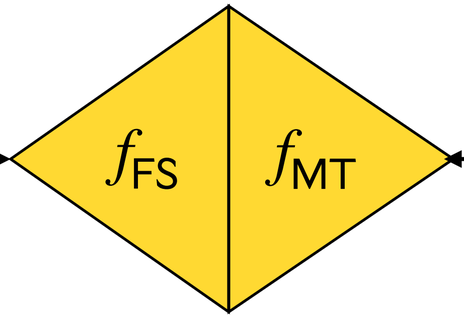
Micali's construction III

Commit to PCP string using MT,
then apply FS transform



Micali's construction III

Commit to PCP string using MT,
then apply FS transform



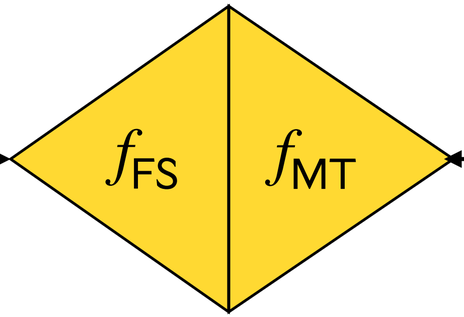
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\text{PCP}}(x, w)$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT, then apply FS transform



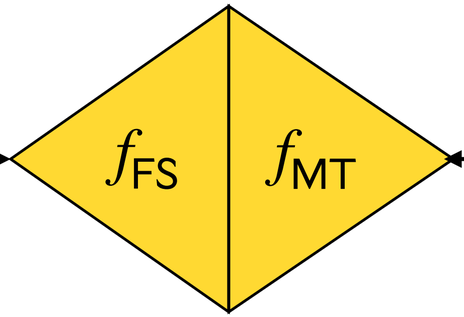
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT, then apply FS transform



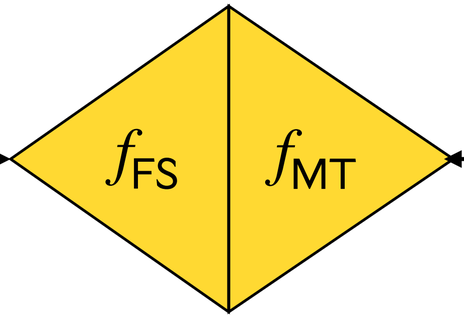
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT, then apply FS transform



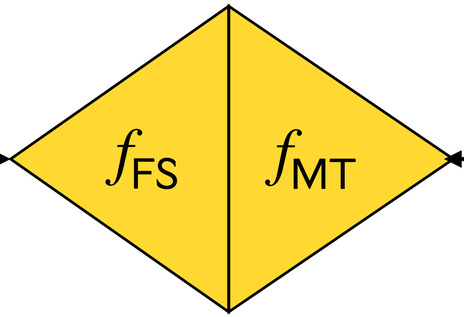
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT, then apply FS transform



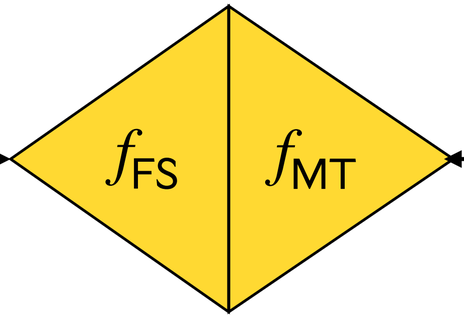
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT, then apply FS transform



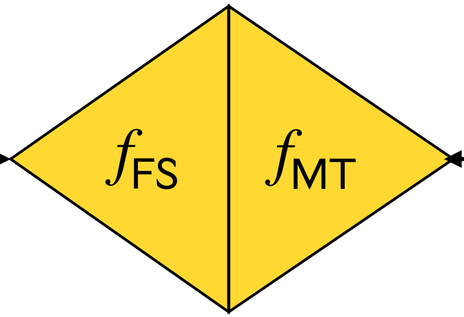
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$
- Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \mathbf{a}$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT, then apply FS transform



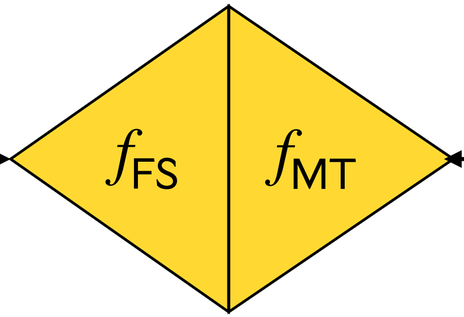
$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$
- Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \mathbf{a}$
- $\mathsf{pf} := \mathsf{MTOpen}(\mathsf{td}, Q)$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT, then apply FS transform



$\mathbf{P}(x, w)$

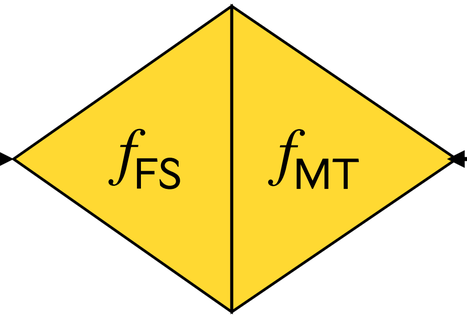
- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$
- Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \mathbf{a}$
- $\mathsf{pf} := \mathsf{MTOpen}(\mathsf{td}, Q)$

$(\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$

$\mathbf{V}(x)$

Micali's construction III

Commit to PCP string using MT, then apply FS transform



$\mathbf{P}(x, w)$

- Compute $\Pi \leftarrow \mathbf{P}_{\mathsf{PCP}}(x, w)$
- Sample $\sigma_{\mathsf{MT}} \leftarrow \{0,1\}^{r \cdot \ell}$
- $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MTCommit}^{f_{\mathsf{MT}}}(\Pi; \sigma_{\mathsf{MT}})$
- Sample $\sigma \leftarrow \{0,1\}^{r}$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$
- Run $\mathbf{V}^{\Pi}_{\mathsf{PCP}}(x; \rho)$ to obtain query-answers sets $Q, \mathbf{a}$
- $\mathsf{pf} := \mathsf{MTOpen}(\mathsf{td}, Q)$

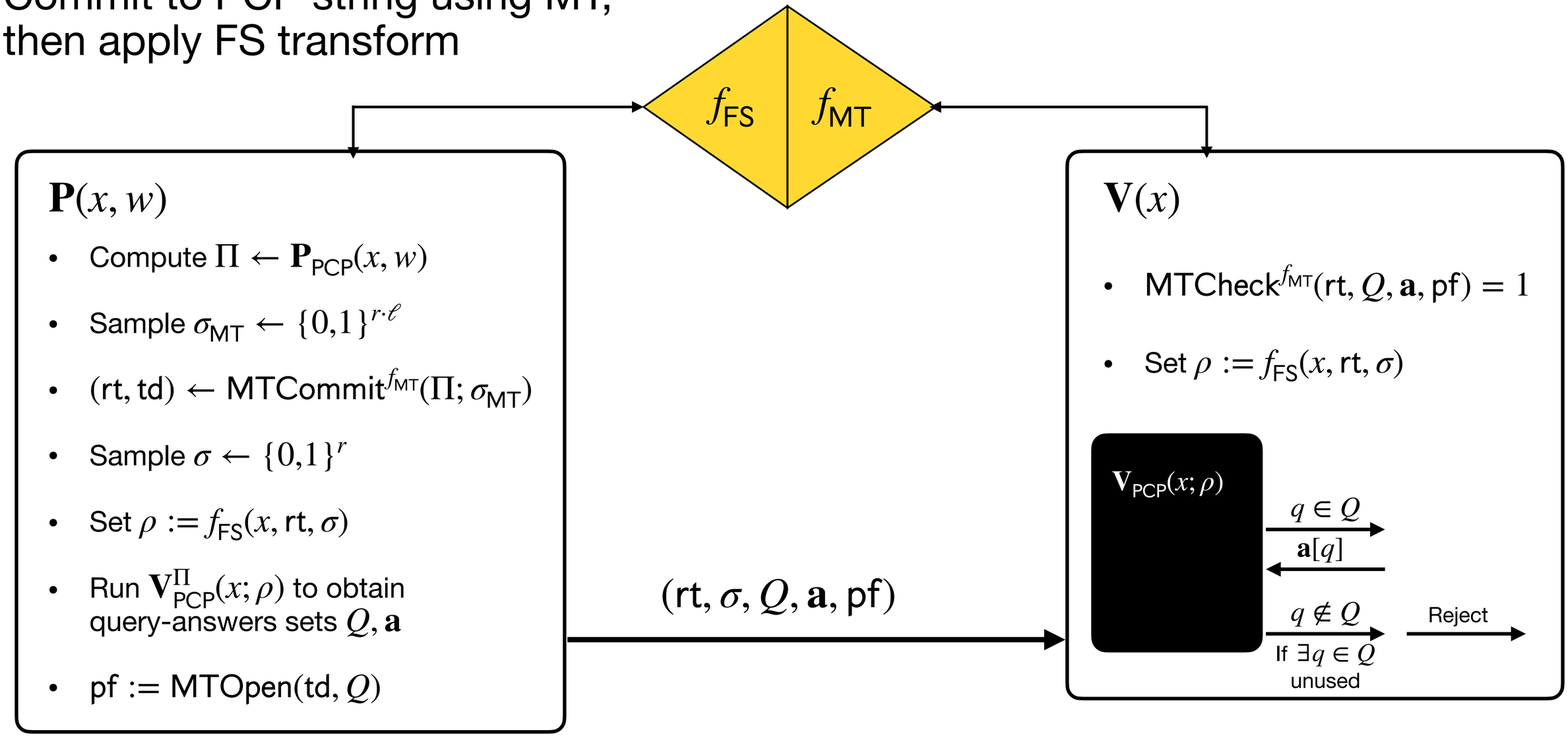
$(\mathsf{rt}, \sigma, Q, \mathbf{a}, \mathsf{pf})$

$\mathbf{V}(x)$

- $\mathsf{MTCheck}^{f_{\mathsf{MT}}}(\mathsf{rt}, Q, \mathbf{a}, \mathsf{pf}) = 1$
- Set $\rho := f_{\mathsf{FS}}(x, \mathsf{rt}, \sigma)$

Micali's construction III

Commit to PCP string using MT, then apply FS transform



UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

$$\Pr\left[\;\middle|\;\forall \mathscr{A}\right]$$

UC-friendly completeness

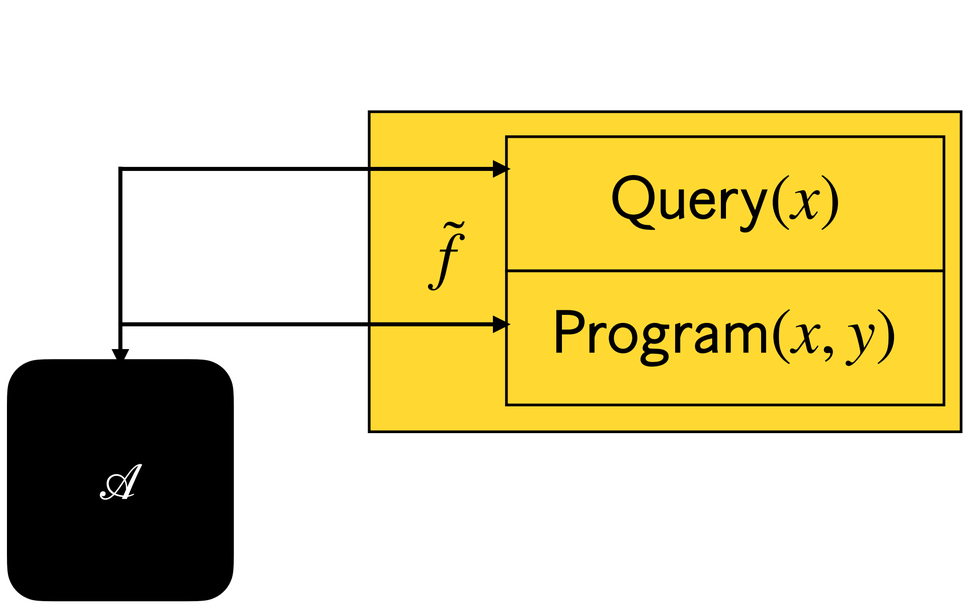
Adversary should not be able to make honestly generated proofs fail to verify.

$$\Pr\left[\; \mathcal{A} \;\middle|\; \forall \mathcal{A} \;\right]$$

UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

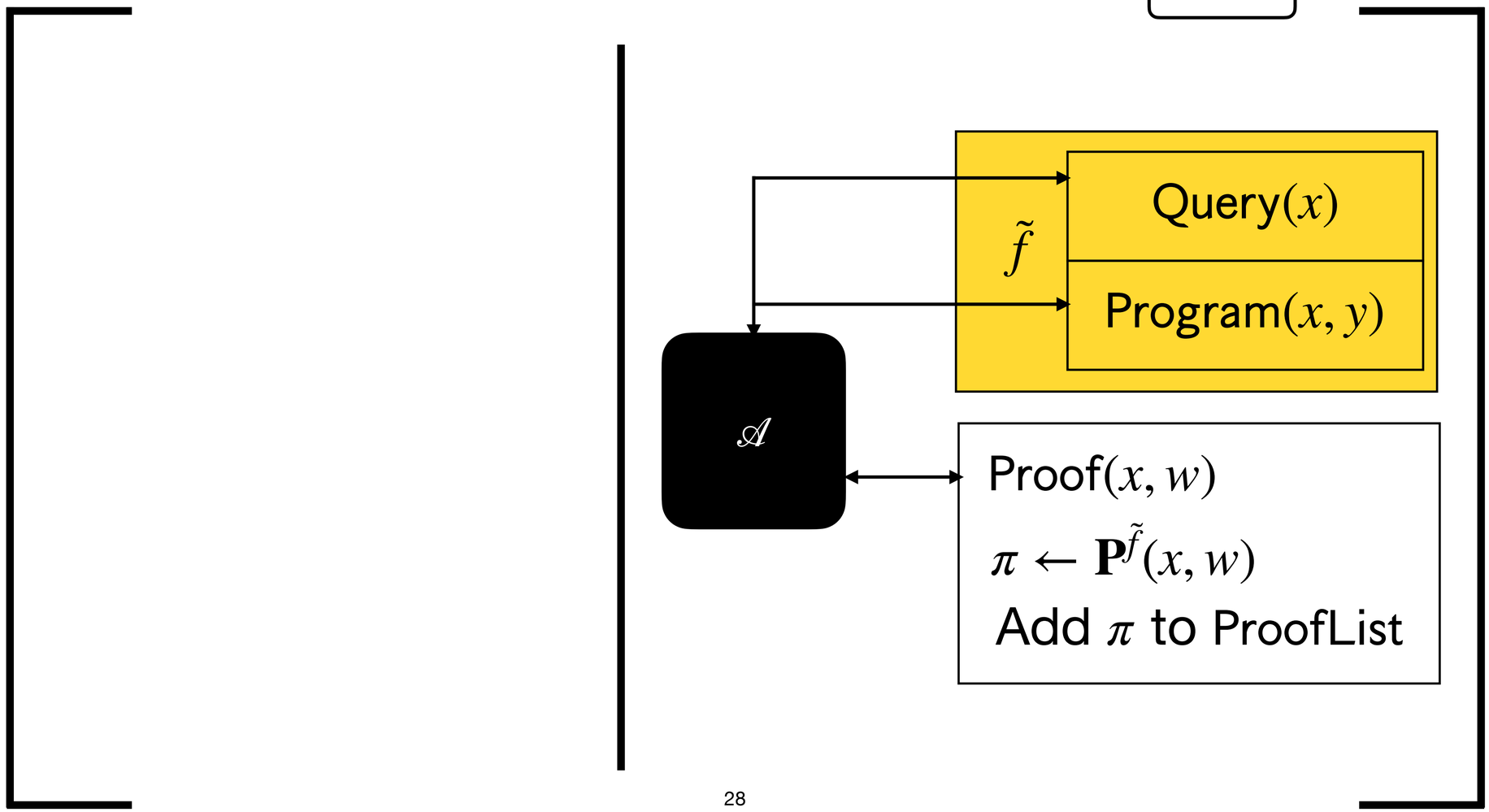
$$\Pr\Big[\;\Big|\;\forall\mathscr{A}$$



UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

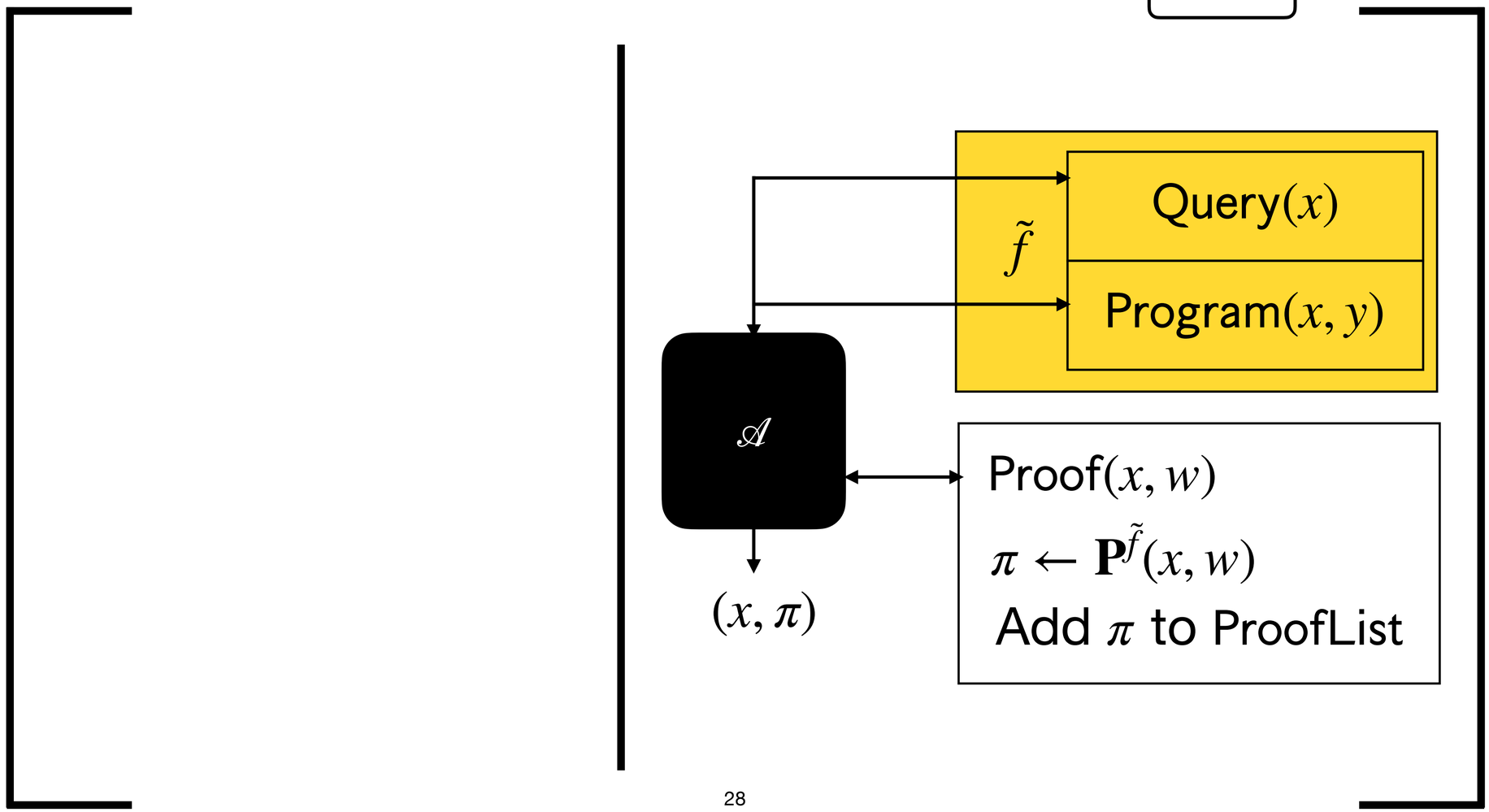
$$\Pr$$



UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

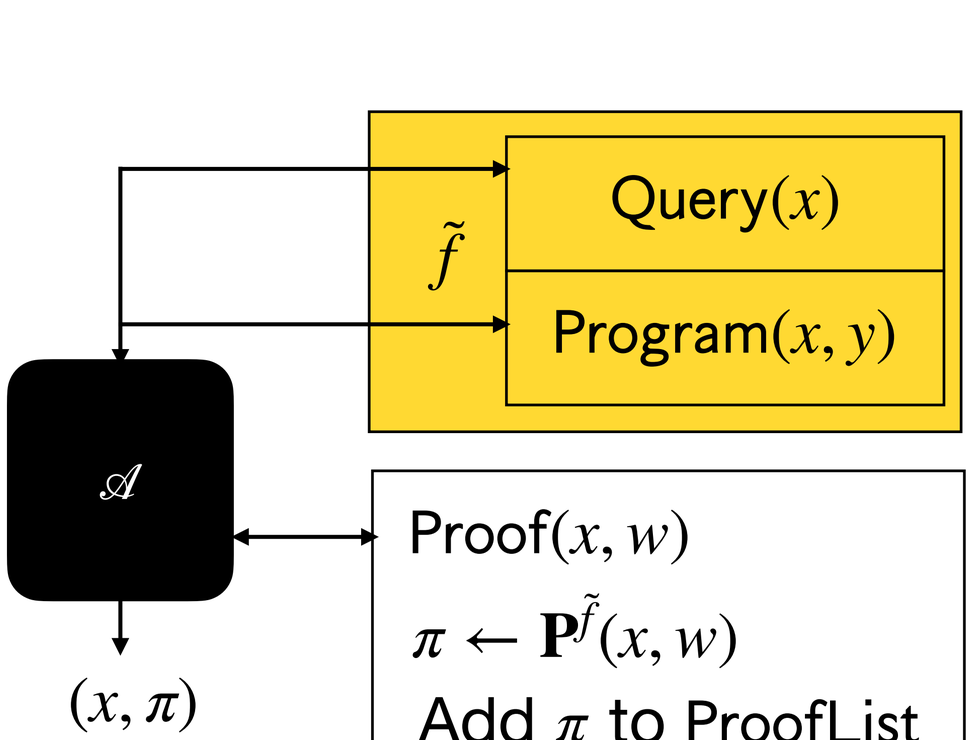
$$\Pr$$



UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

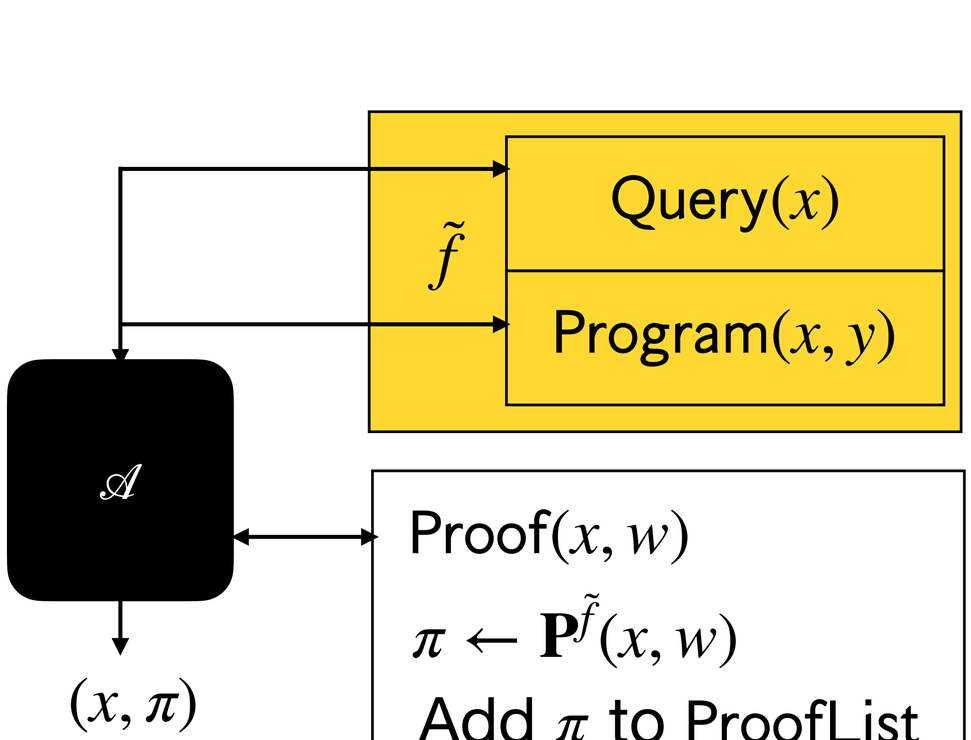
$$\Pr\left[\begin{array}{l|l} (x,\pi) \in \text{ProofList} \\ \text{and} & \forall \mathscr{A} \end{array}\right]$$



UC-friendly completeness

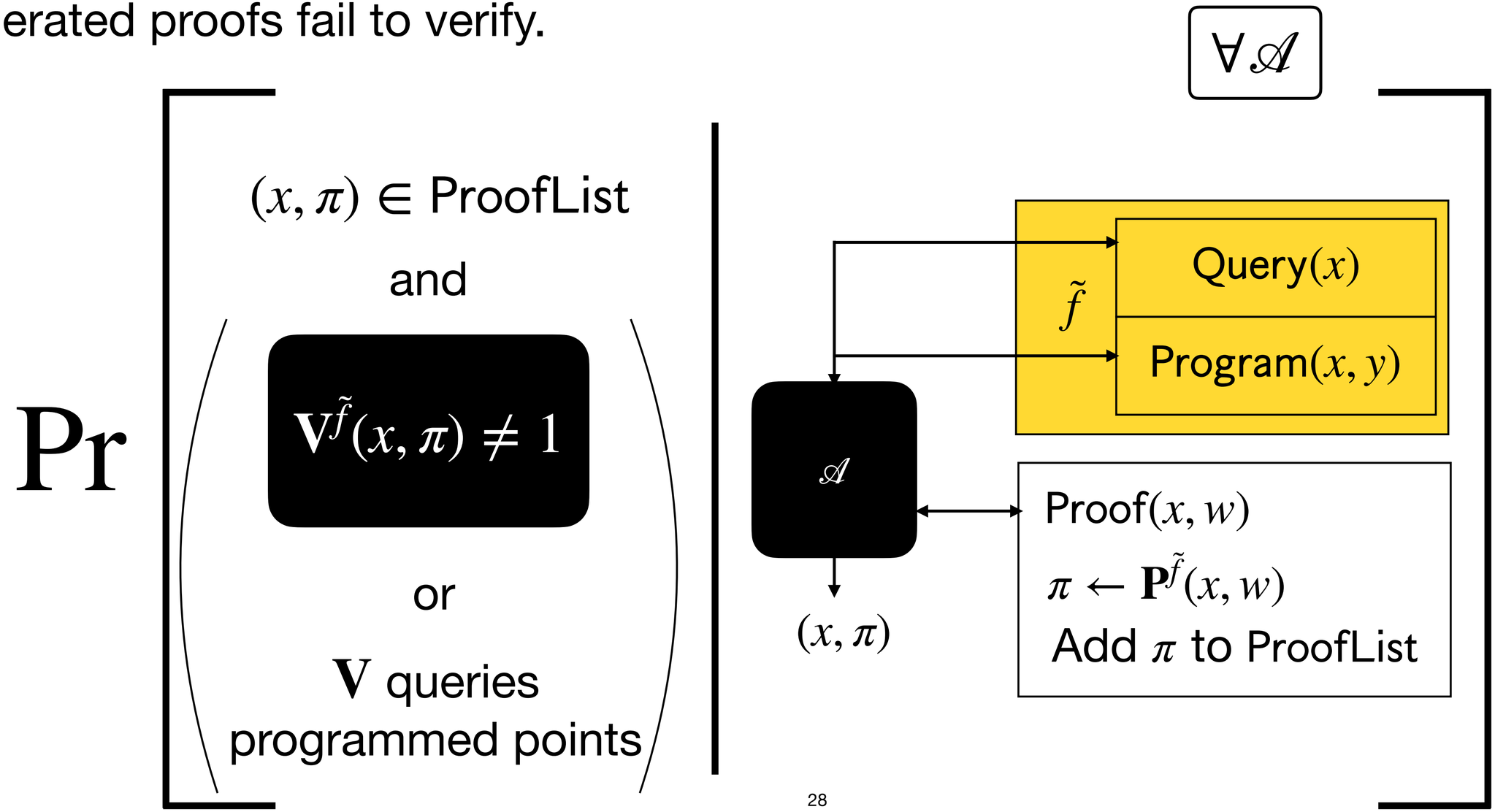
Adversary should not be able to make honestly generated proofs fail to verify.

$$\Pr\left[\begin{array}{l|l} (x,\pi) \in \text{ProofList} \text{ and } \mathbf{V}^{\tilde{f}}(x,\pi) \neq 1 & \forall\, \mathscr{A} \end{array}\right]$$



UC-friendly completeness

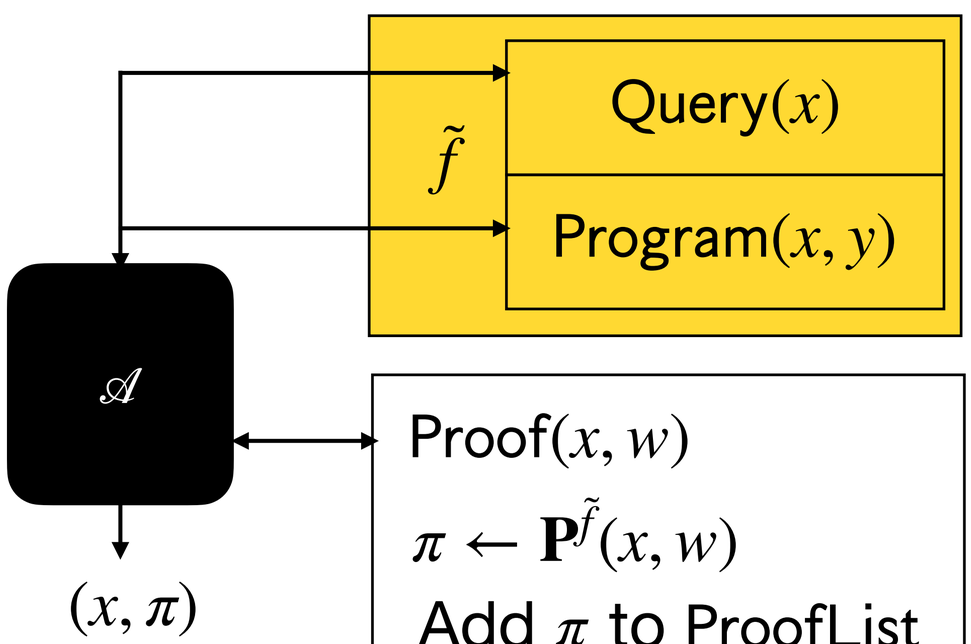
Adversary should not be able to make honestly generated proofs fail to verify.



UC-friendly completeness

Adversary should not be able to make honestly generated proofs fail to verify.

$$\Pr\left[\left(\begin{array}{c}(x,\pi) \in \text{ProofList} \\ \text{and} \\ \mathbf{V}^{\tilde{f}}(x,\pi) \neq 1 \\ \text{or} \\ \mathbf{V} \text{ queries} \\ \text{programmed points}\end{array}\right) \;\middle|\; \forall \mathscr{A}\right] \leq \varepsilon$$



UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.

UC-friendly ZK

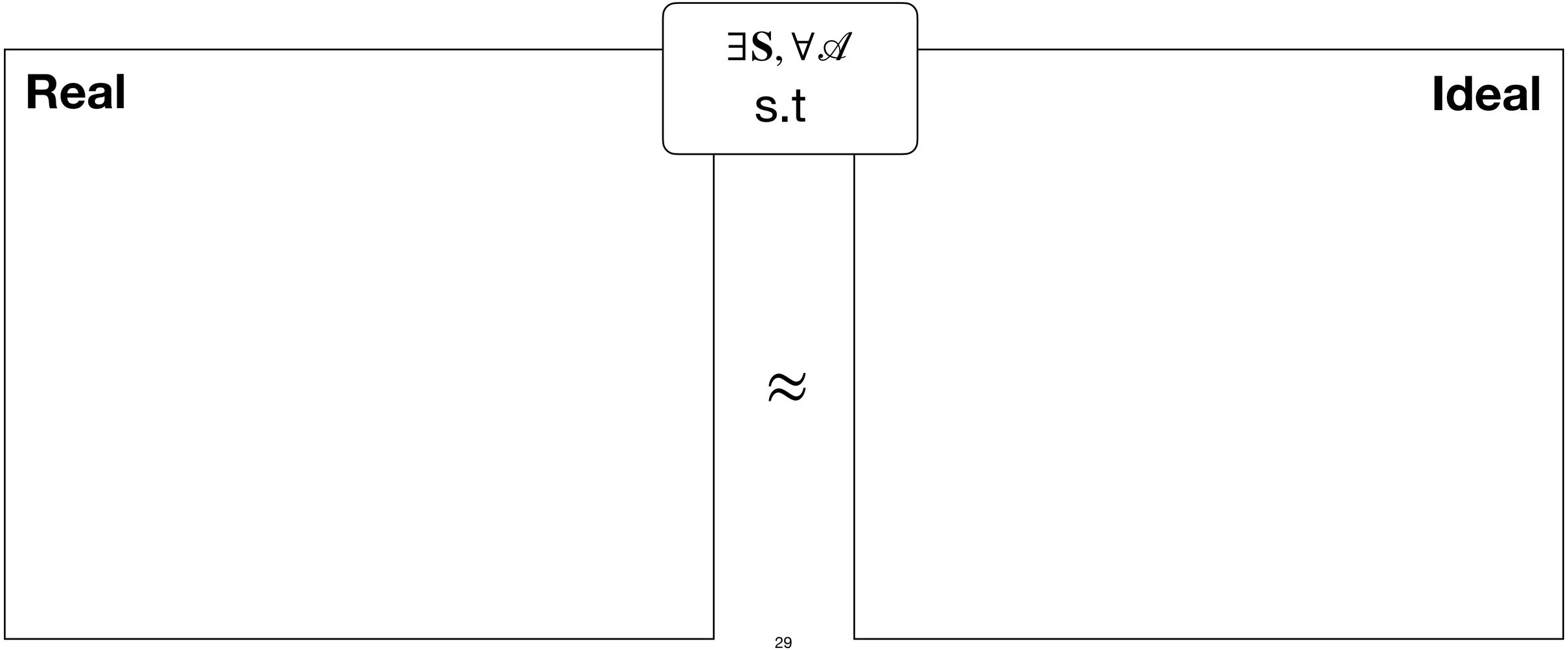
Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.

$\exists \mathbf{S}, \forall \mathscr{A}$

s.t

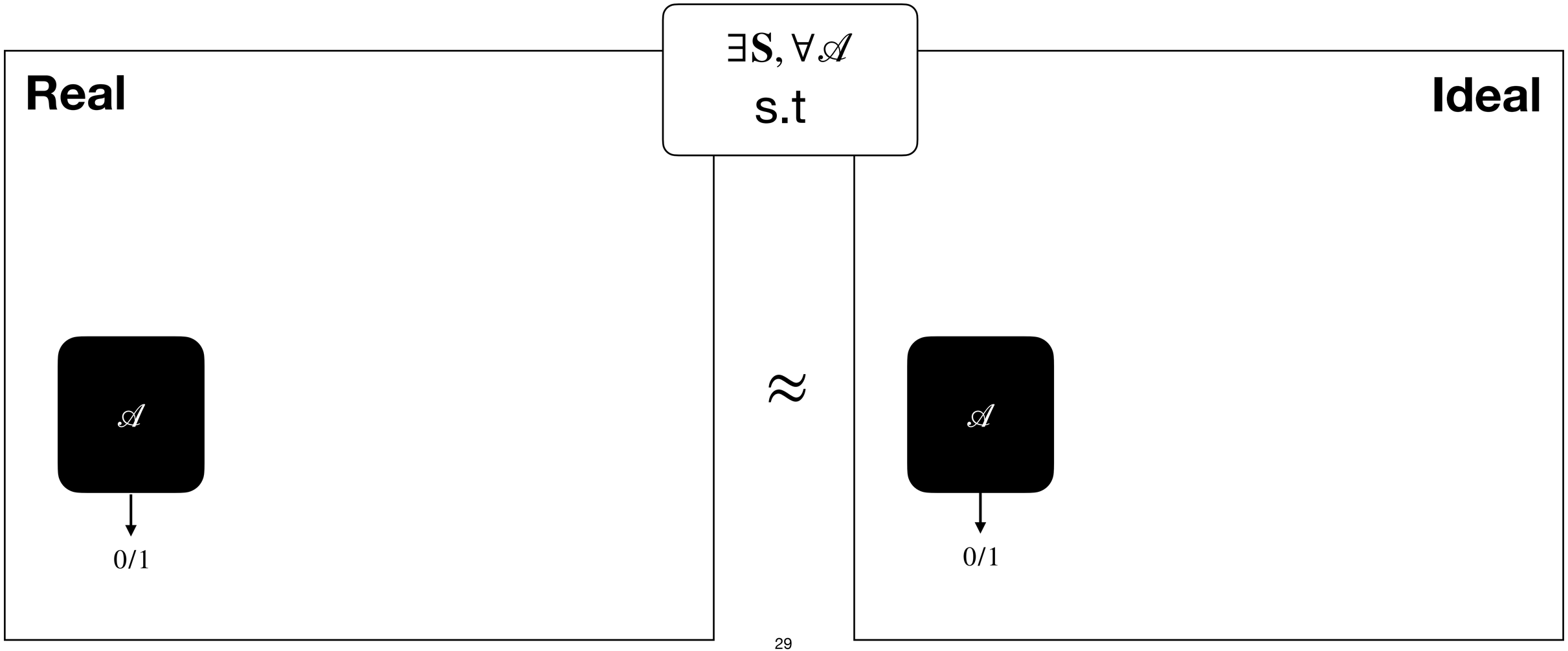
UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.



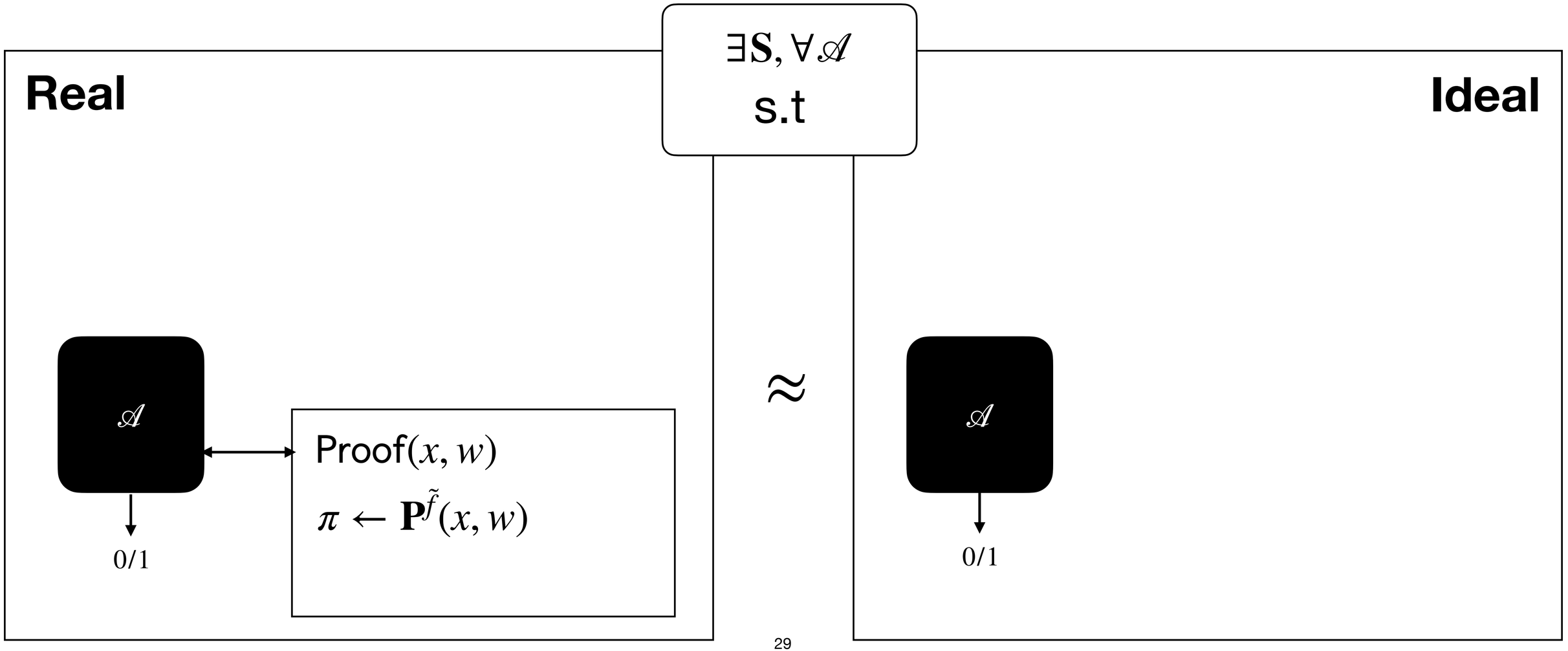
UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.



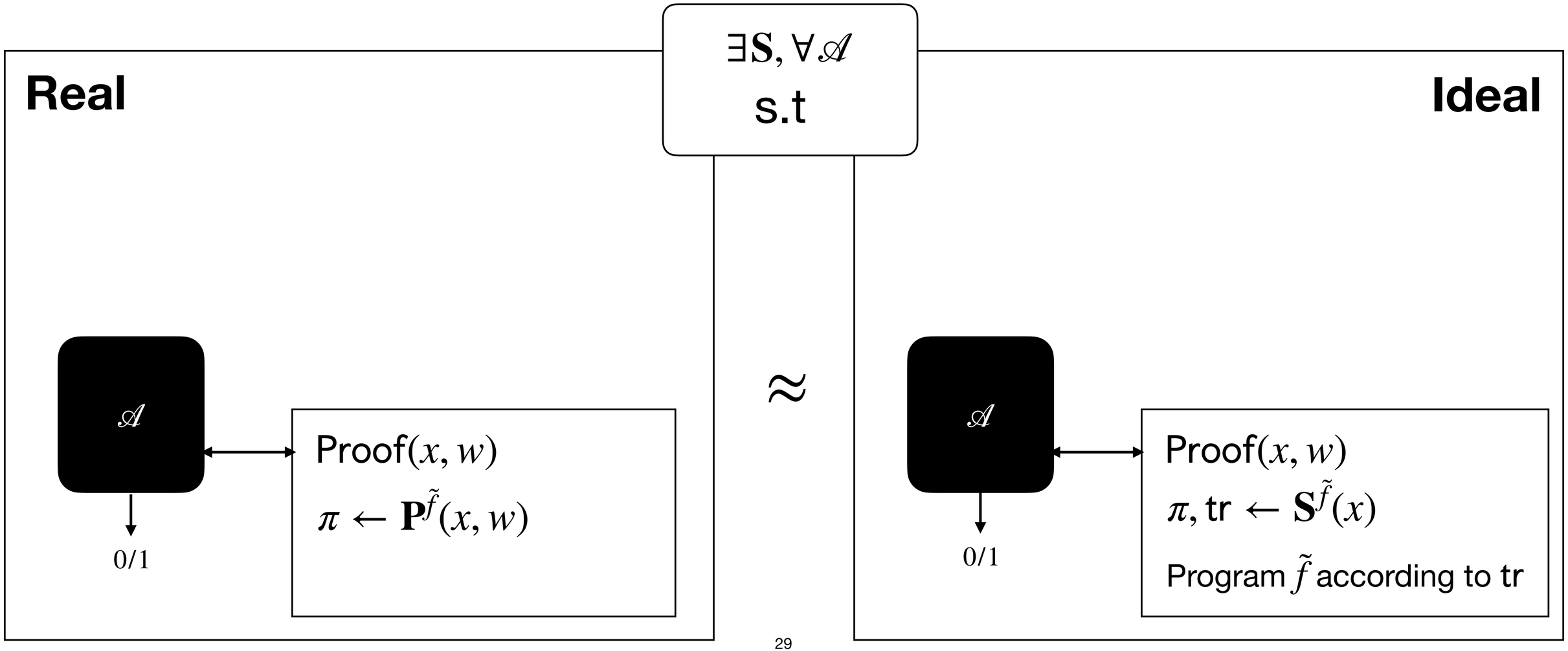
UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.



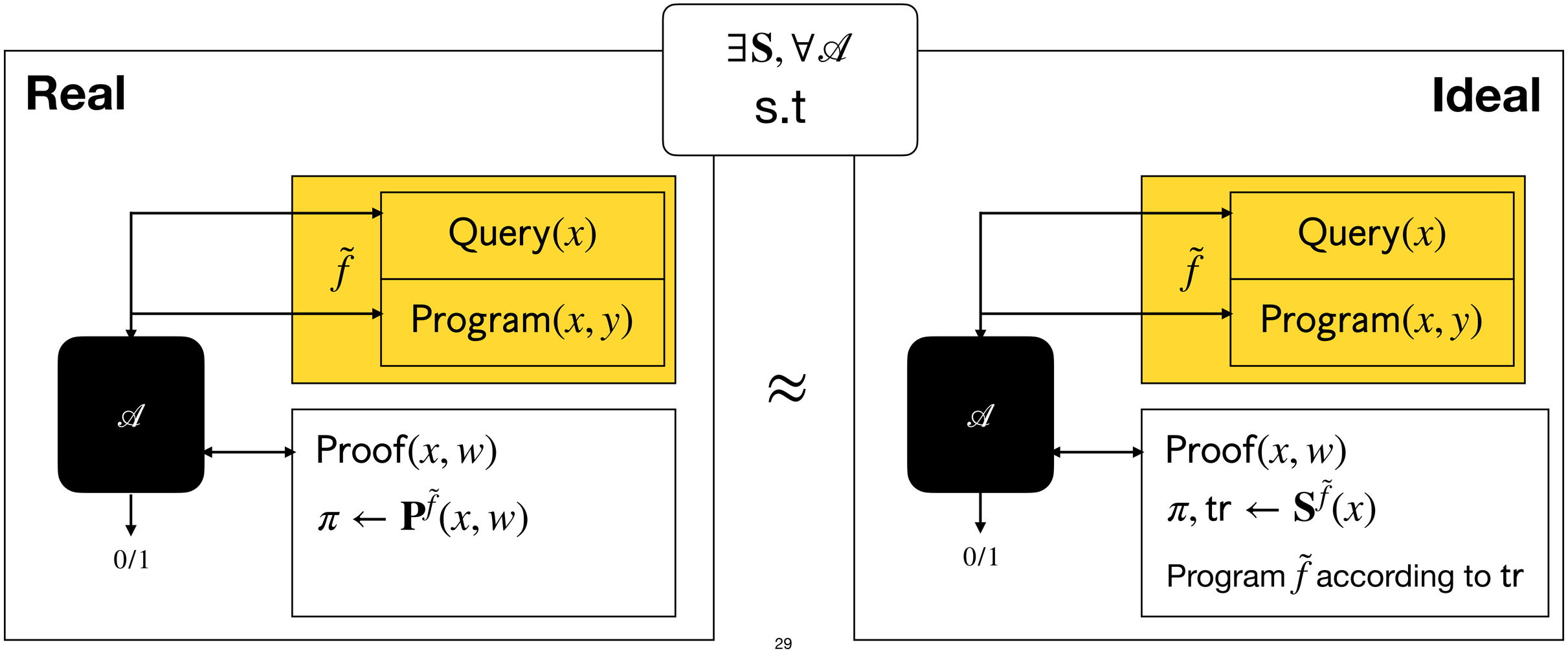
UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.



UC-friendly ZK

Adversary should not be able to distinguish real and simulated proofs, even with access to a programming oracle.



Micali has UC-friendly ZK

Micali has UC-friendly ZK

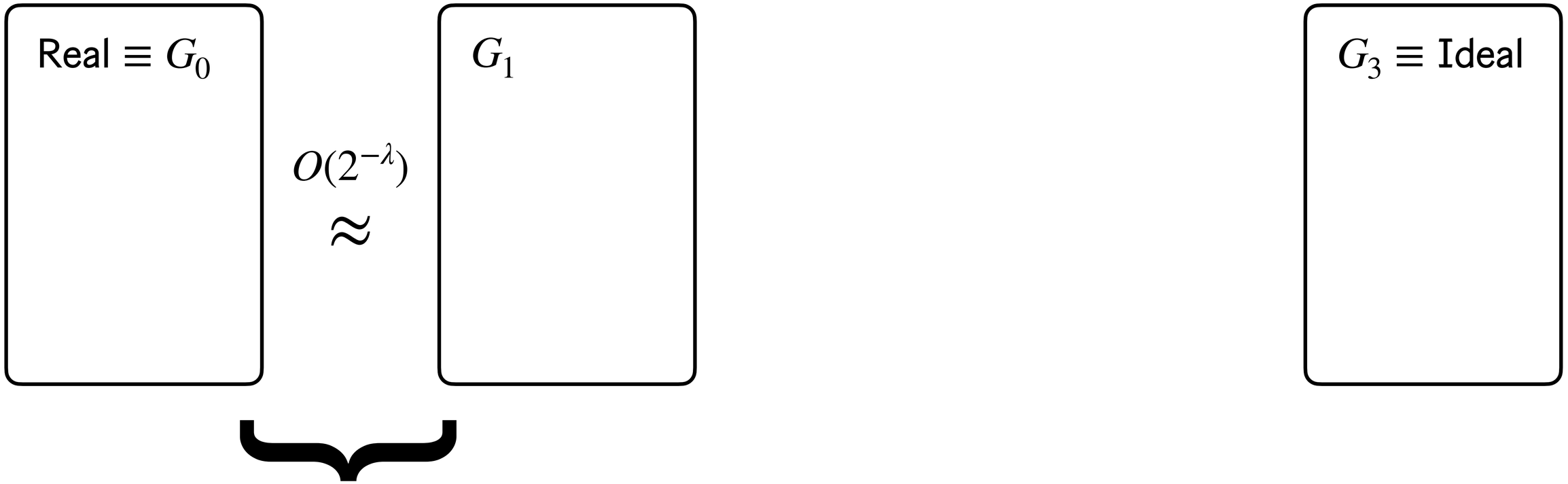
Real $\equiv G_0$

Micali has UC-friendly ZK

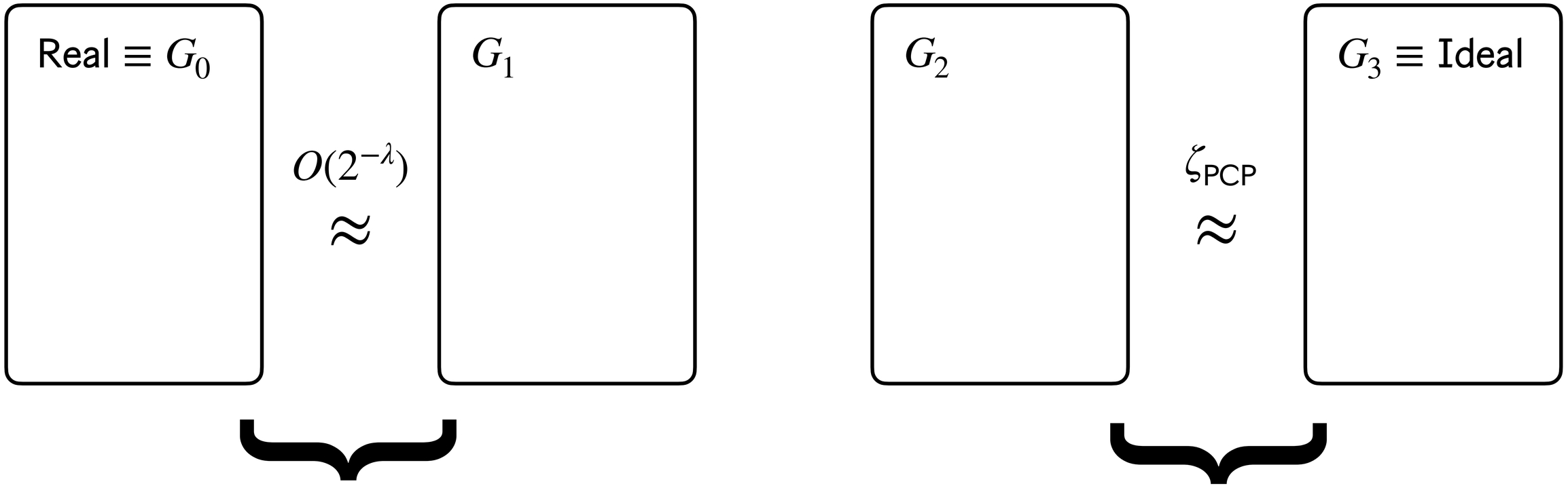
Real $\equiv G_0$

$G_3 \equiv$ Ideal

Micali has UC-friendly ZK



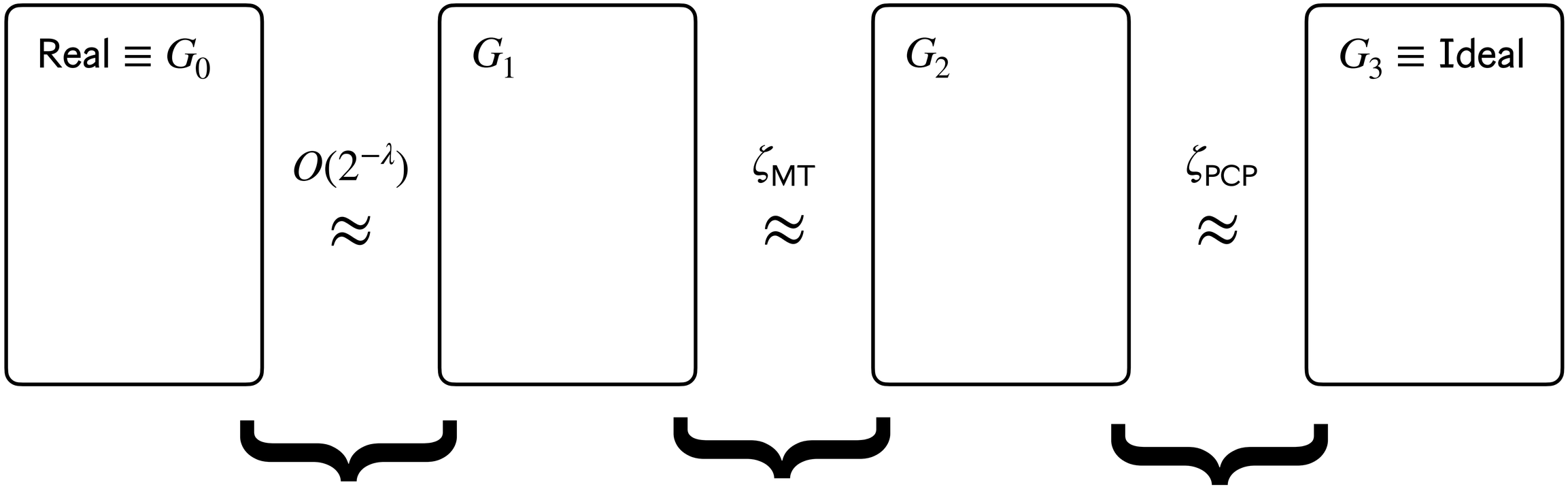
Micali has UC-friendly ZK



FS input hard to predict

PCP honest-verifier ZK

Micali has UC-friendly ZK



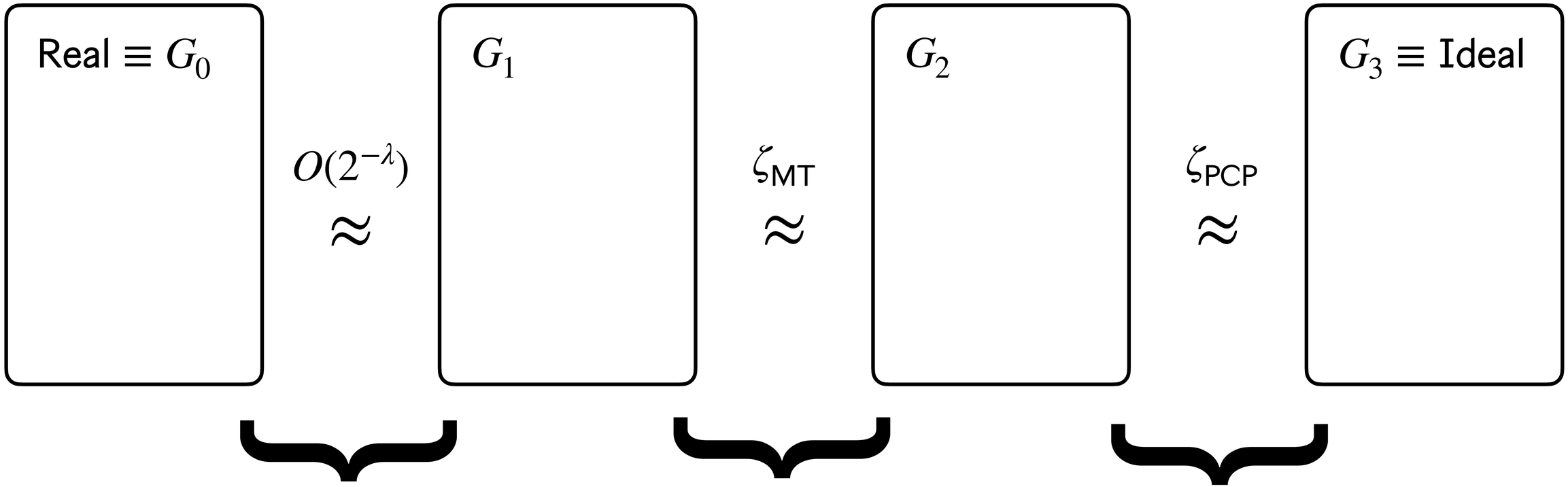
FS input hard to predict

Lemma: Merkle trees have UC-friendly hiding

PCP honest-verifier ZK

Micali has UC-friendly ZK

Follows similarly to standard Micali ZK + Merkle trees are UC-friendly.



FS input hard to predict

Lemma: Merkle trees have UC-friendly hiding

PCP honest-verifier ZK

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs
that the extractor cannot extract a witness from

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$$\exists \mathscr{E} \forall \mathscr{A}$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\left[\right]$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

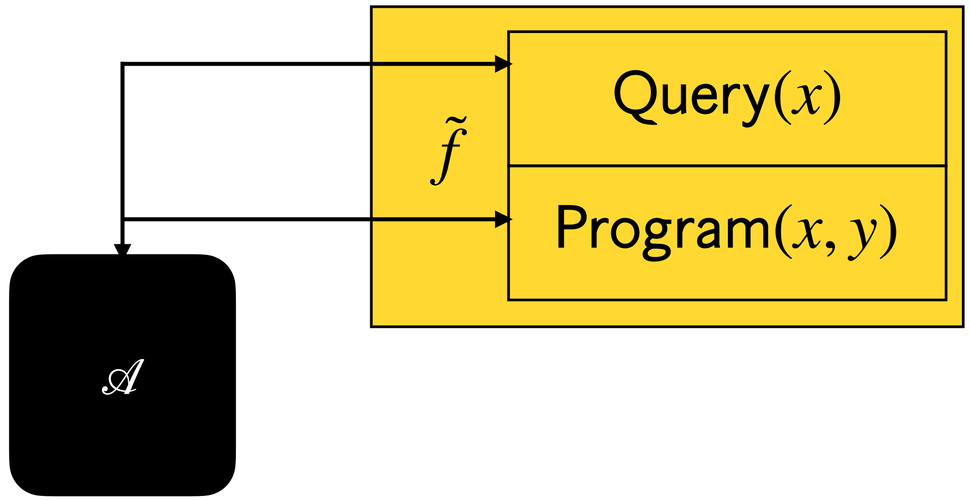
$$\Pr\left[\; \middle| \; \mathscr{A} \right]$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\left[\;\middle|\;\right]$$

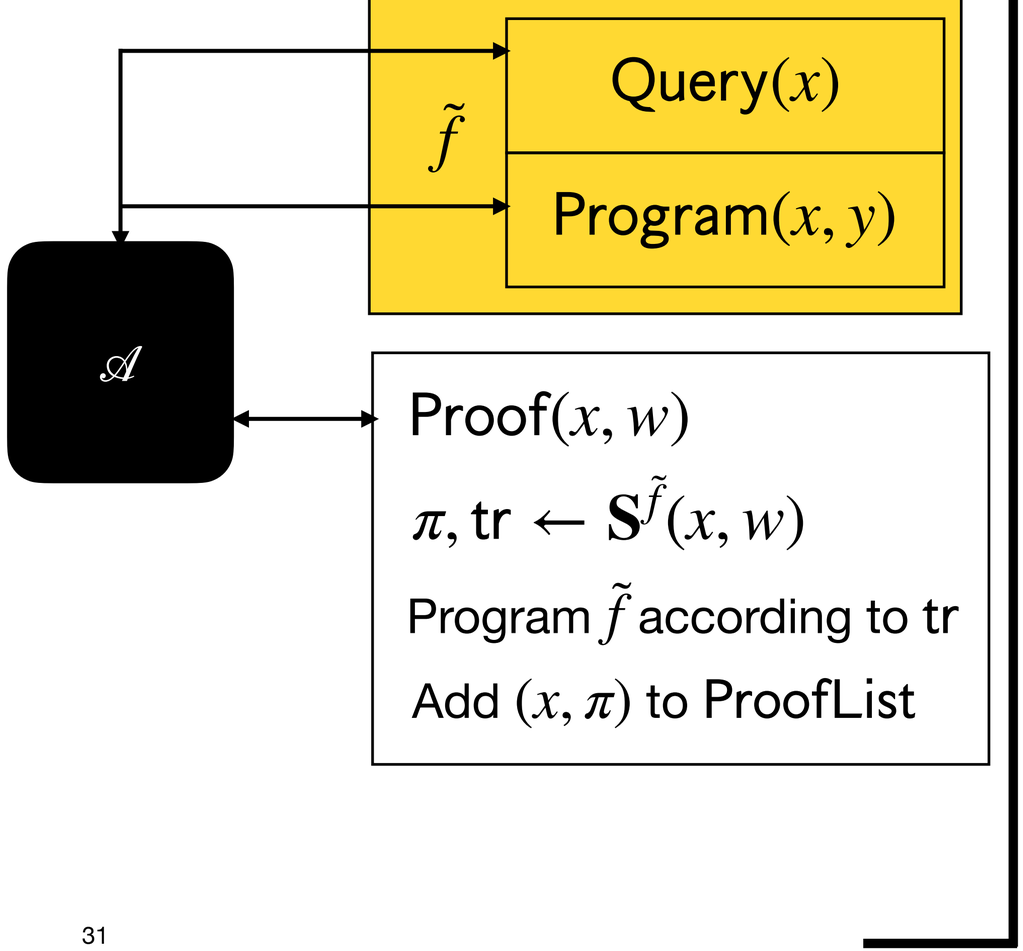


UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\Big[$$



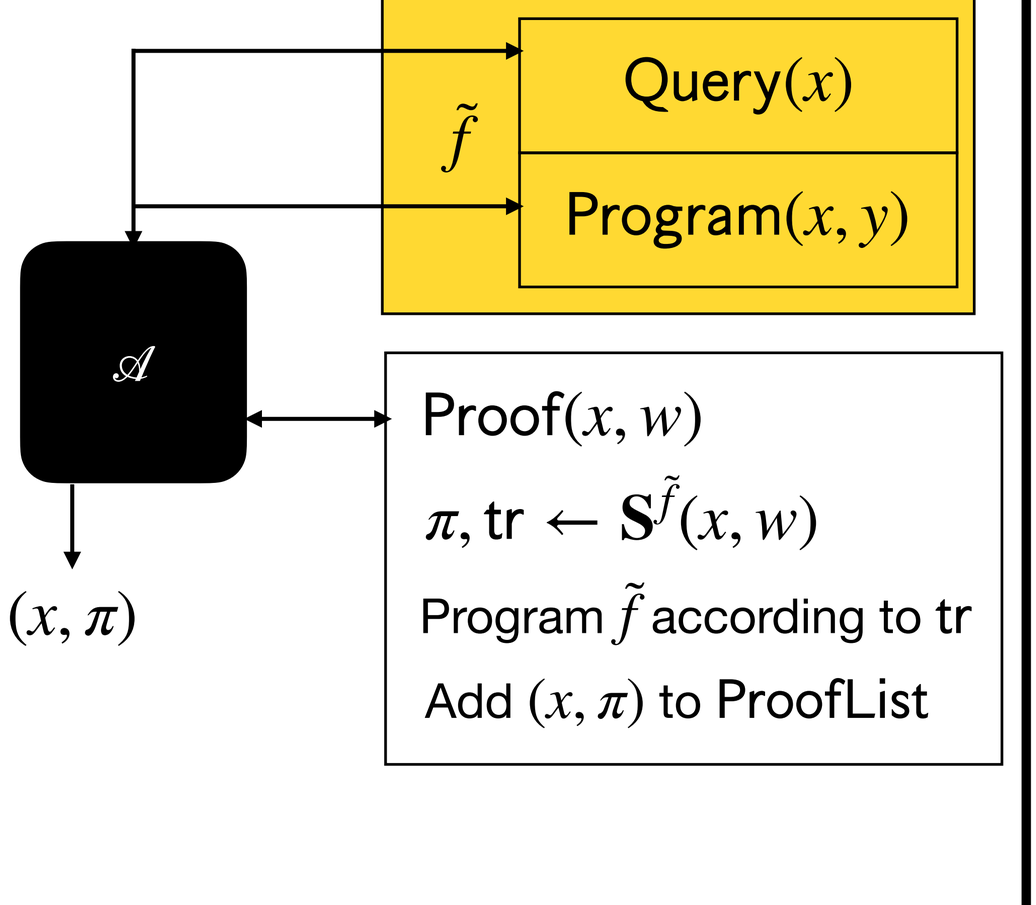
$$\Big]$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\Big[$$



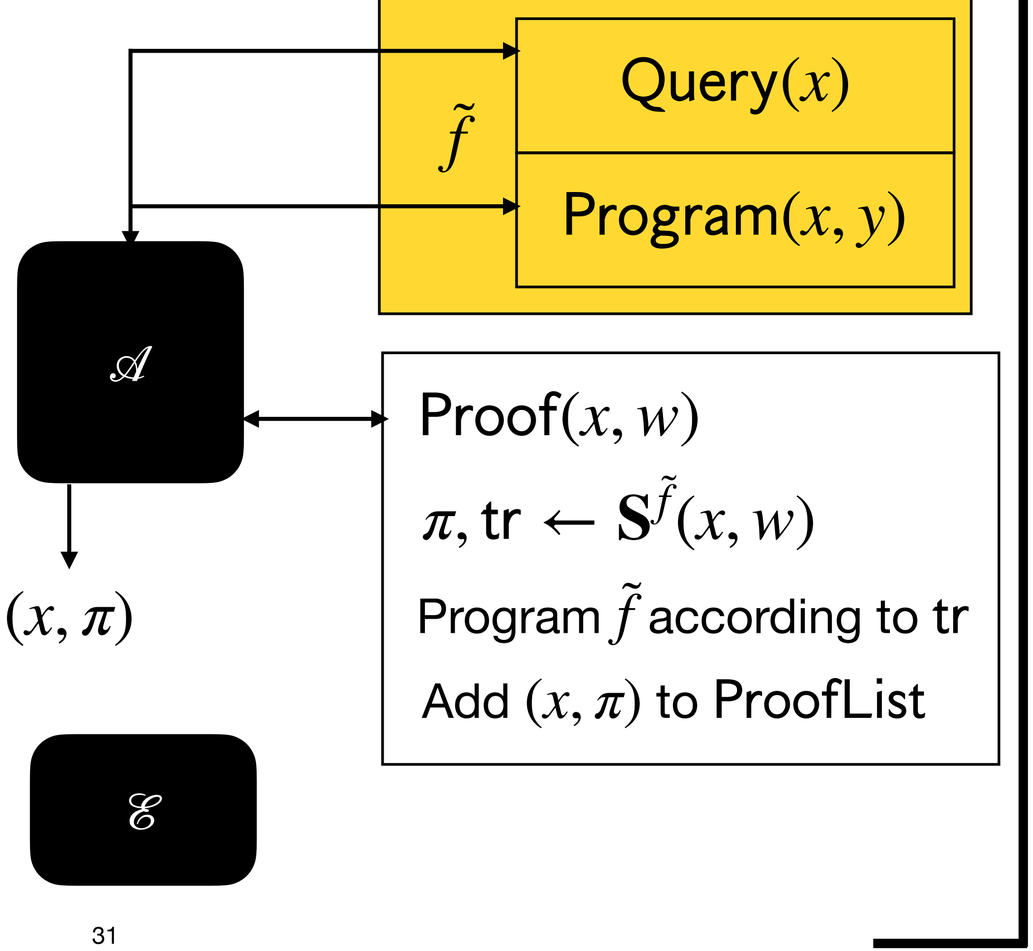
$$\Big]$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\Big[$$

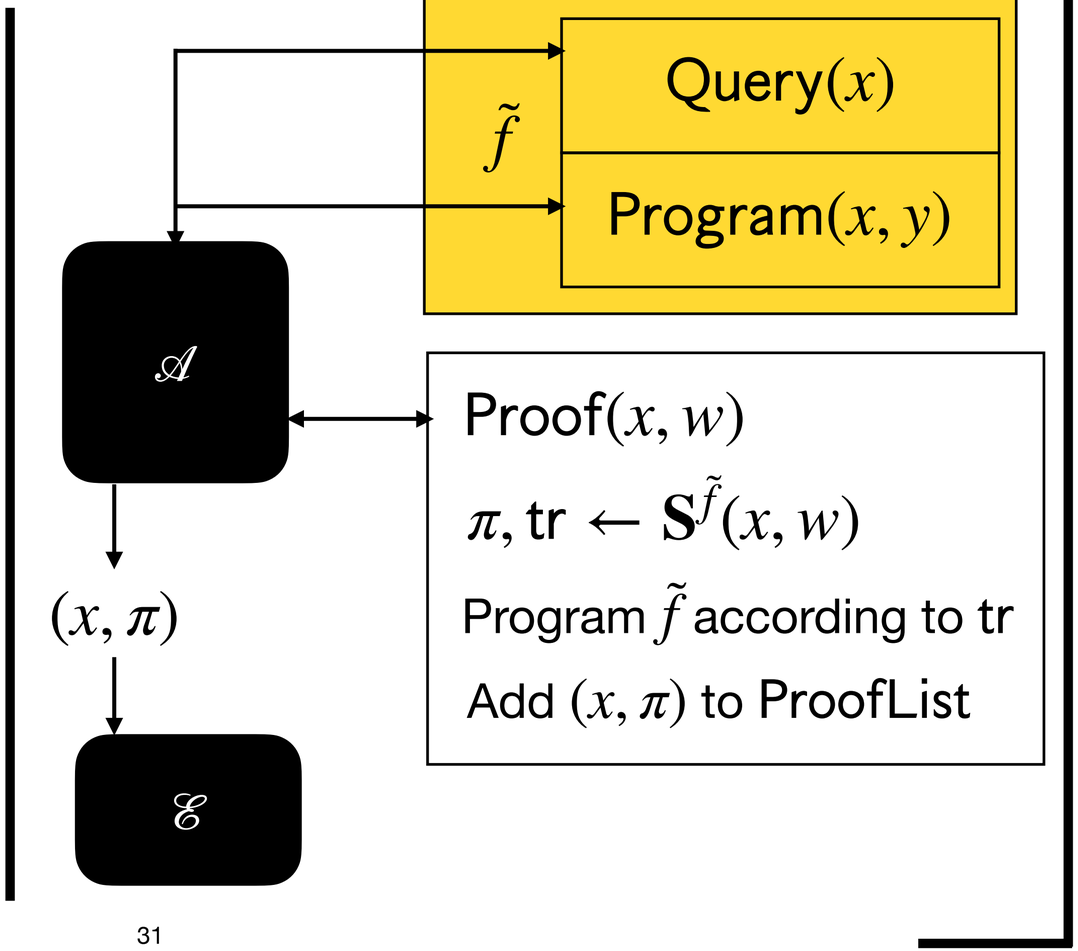


UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\Big[$$



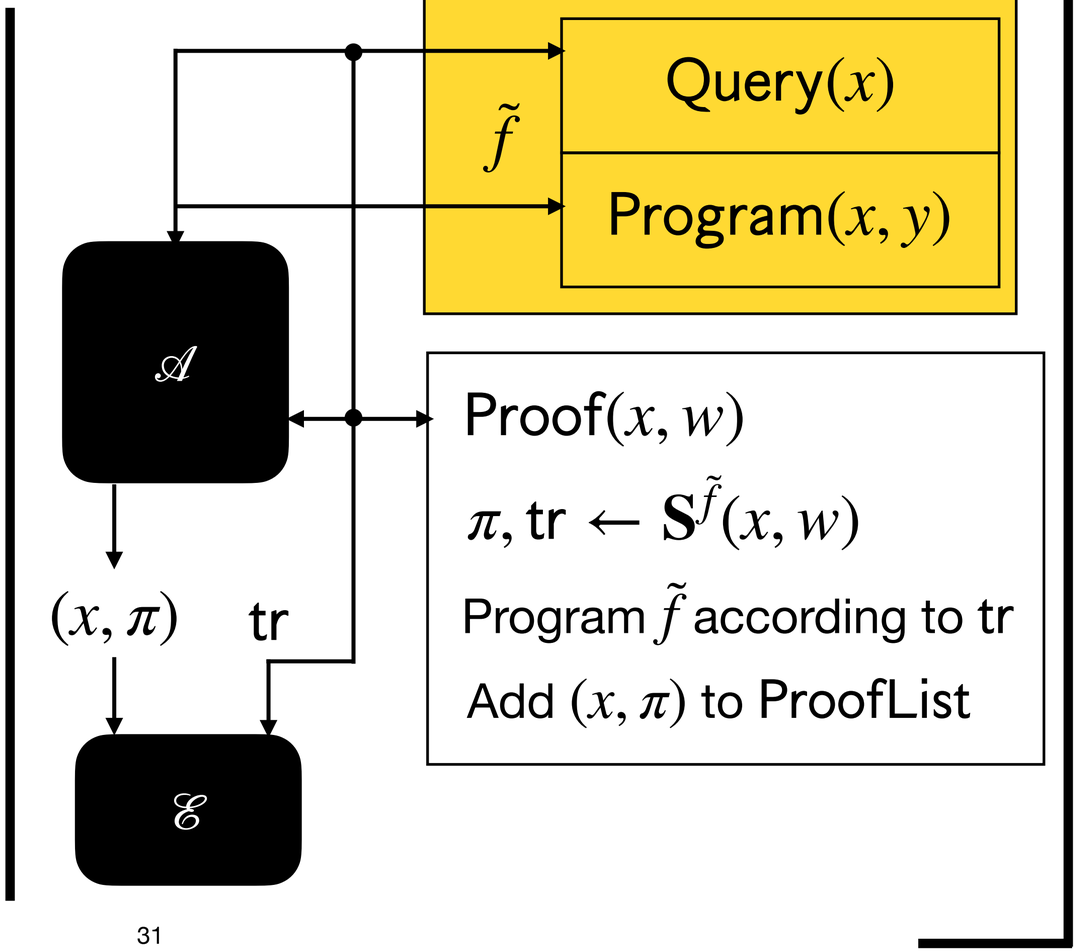
$$\Big]$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\Big[$$



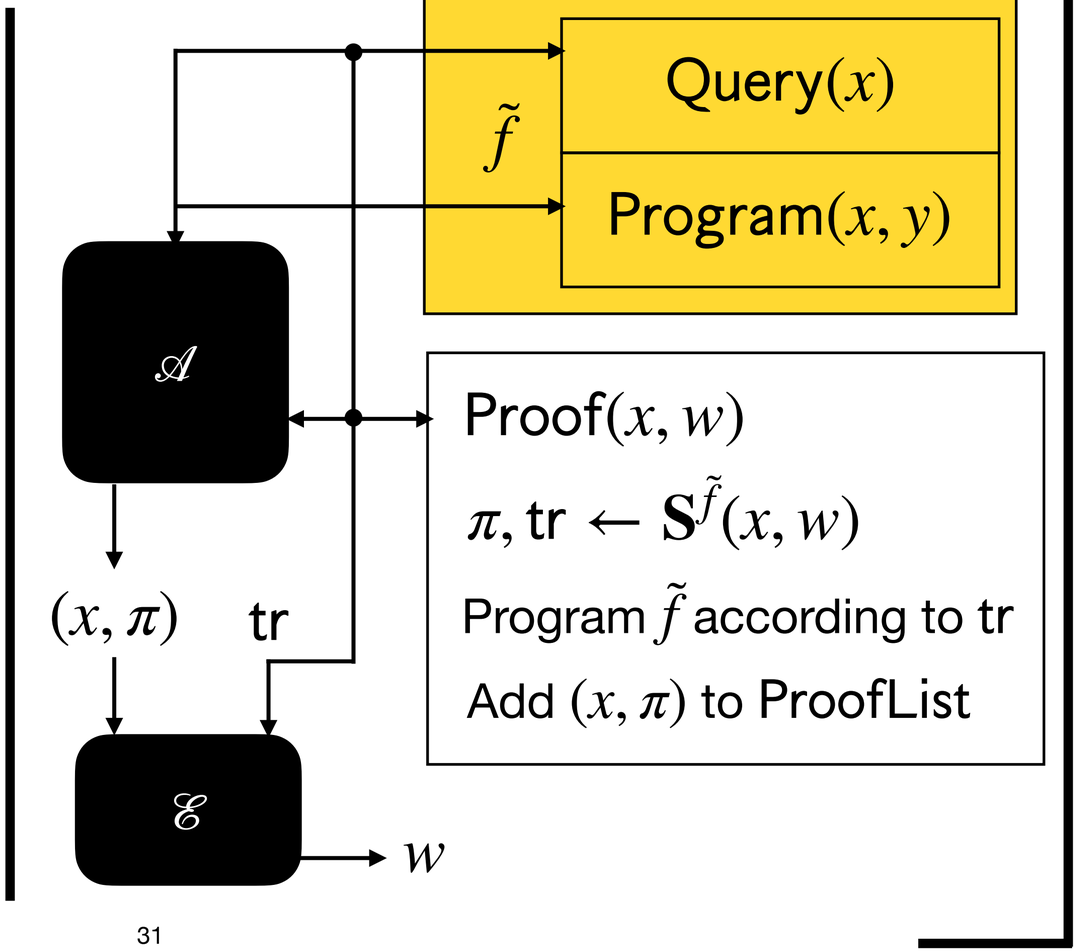
$$\Big]$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\Big[$$



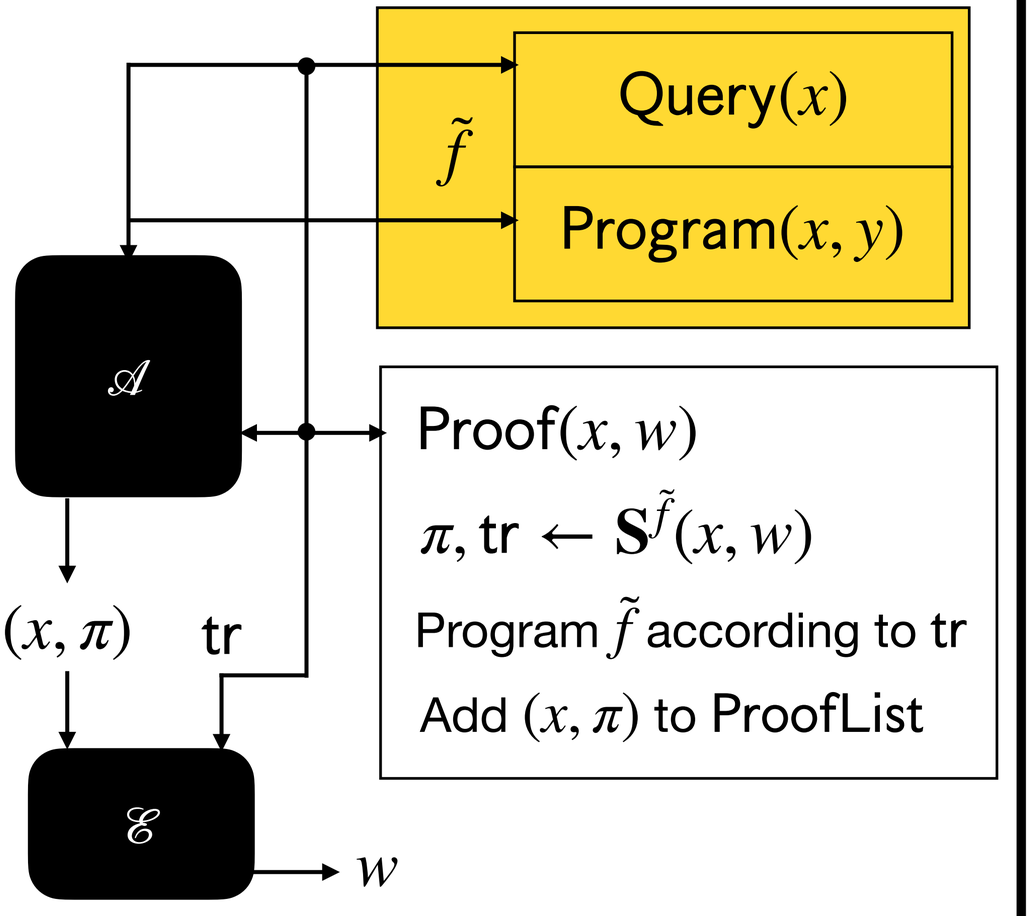
$$\Big]$$

UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\left[\mathrm{V}^{\tilde{f}}(x,\pi)=1\right]$$

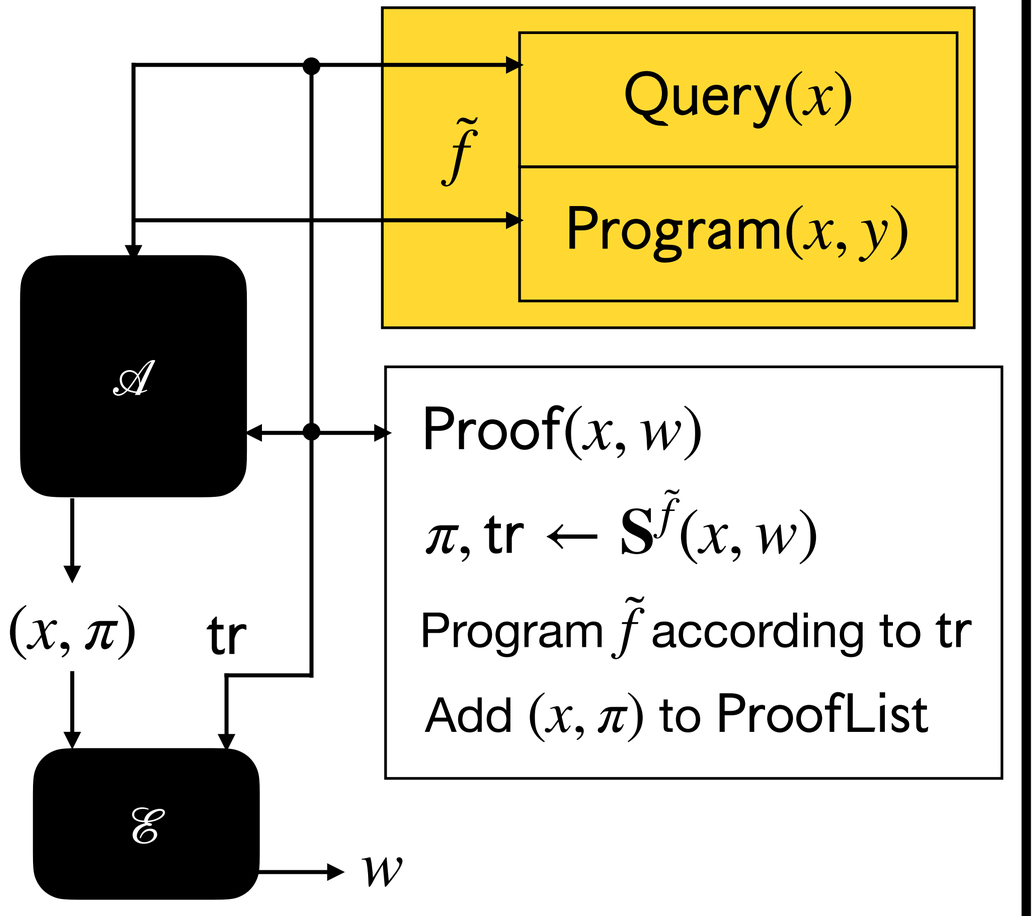


UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\left[\mathbf{V}^{\tilde{f}}(x,\pi) = 1 \;\text{ and }\; \mathbf{V} \text{ does not query programmed points}\right]$$

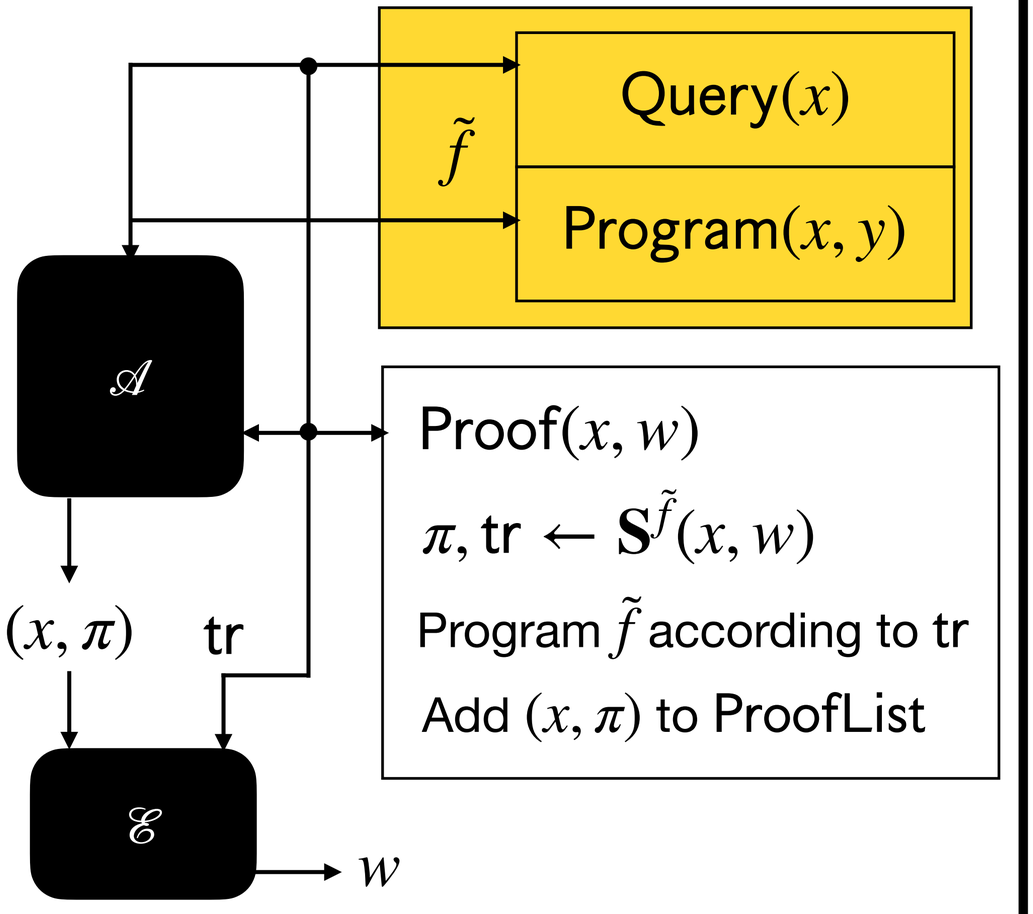


UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\left[\mathbf{V}^{\tilde{f}}(x,\pi) = 1 \;\wedge\; \mathbf{V} \text{ does not query programmed points} \;\wedge\; (x,\pi) \notin \text{ProofList}\right]$$

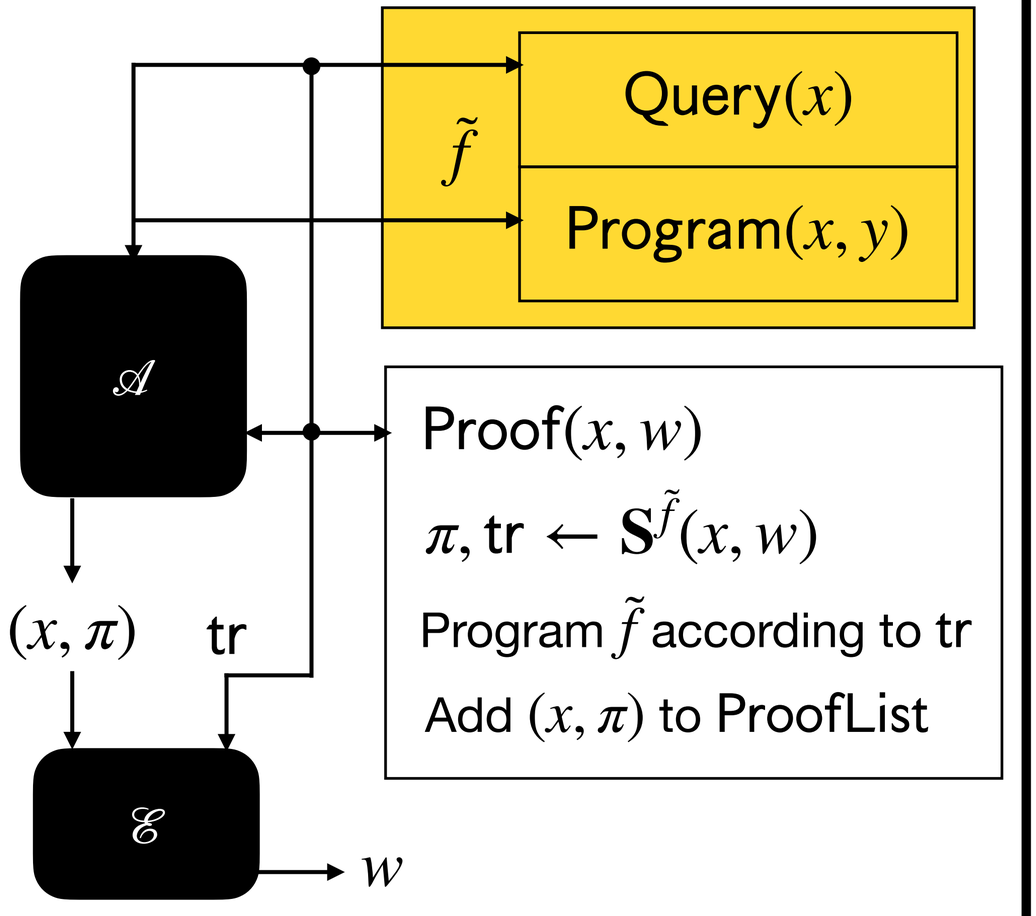


UC-friendly knowledge soundness

Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\left[\begin{array}{l} \mathbf{V}^{\tilde{f}}(x,\pi) = 1 \\ \\ \mathbf{V} \text{ does not query programmed points} \\ \\ (x,\pi) \notin \text{ProofList} \\ \\ (x,w) \notin R \end{array}\right]$$



UC-friendly knowledge soundness

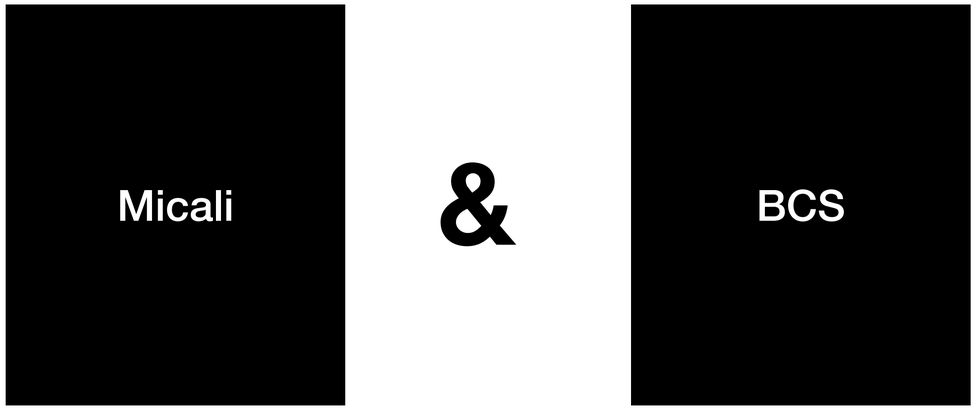
Adversary should not be able to generate fresh proofs that the extractor cannot extract a witness from

$\exists \mathscr{E} \forall \mathscr{A}$

$$\Pr\left[\begin{array}{l} \mathbf{V}^{\tilde{f}}(x,\pi) = 1 \\ \\ \mathbf{V} \text{ does not query programmed points} \\ \\ (x,\pi) \notin \text{ProofList} \\ \\ (x,w) \notin R \end{array} \;\middle|\; \begin{array}{l} \mathscr{A} \text{ has oracle access to } \tilde{f}, \text{Query}(x), \text{Program}(x,y) \\ \\ \text{Proof}(x,w): \\ \pi, \text{tr} \leftarrow \mathbf{S}^{\tilde{f}}(x,w) \\ \text{Program } \tilde{f} \text{ according to tr} \\ \text{Add } (x,\pi) \text{ to ProofList} \\ \\ (x,\pi) \rightarrow \mathscr{E}, \quad \text{tr} \rightarrow \mathscr{E} \\ \\ \mathscr{E} \rightarrow w \end{array}\right] \leq \kappa$$

Conclusion

Conclusion



These zkSNARKs are UC-secure in the GROM

8.6 UC-secure zkSNARKs from Micali

We combine the results in Sections 8.3 to 8.5 to show that, when instantiated with a suitable PCP, the Micali construction yields a UC-secure zkSNARK.

Theorem 8.14. *Let* PCP *be a probabilistically checkable proof with:*

- *(resp. strong) honest-verifier zero knowledge (Definition 8.3) with error* ζ_{PCP}.
- *knowledge soundness (Definition 8.2) with error* κ_{PCP}.

Set $\mathsf{MT} := \mathsf{MT}[\lambda, \Sigma, \mathsf{I}, \mathsf{r}_{\mathsf{MT}}]$ *and* $\mathsf{ARG} := \mathsf{Micali}[\mathsf{PCP}, \mathsf{r}]$. *Then* $\Pi_{\mathsf{a}}[\mathsf{ARG}]$ $(t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$*-UC-realizes* $\mathcal{F}_{\mathrm{aARG}}$ *in the* GRO*-hybrid model with simulation overhead* $\ell_{\mathsf{p}} \cdot (\mathsf{I}(n), \mathsf{I}(n) \cdot \mathsf{q}(n) + 1)$ *and error*

$$z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$$

In the above we let:

- $z_{\mathrm{UC}}(\epsilon_{\mathrm{ARG}}, \zeta_{\mathrm{ARG}}, \kappa_{\mathrm{ARG}}, \lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) := \epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}}) + \zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}) + \kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Theorem 6.1,*
- $\epsilon_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 8.7.*
- $\zeta_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 8.11,*
- $\kappa_{\mathrm{ARG}}(\lambda, n, t_{\mathsf{q}}, t_{\mathsf{p}}, \ell_{\mathsf{p}}, \ell_{\mathsf{v}})$ *as in Lemma 8.13.*

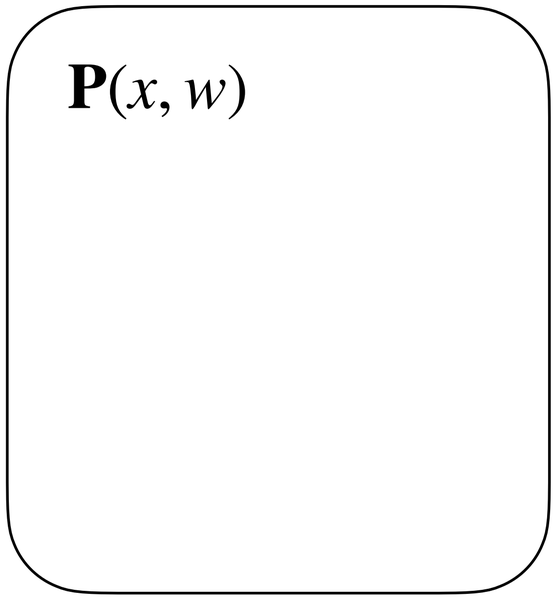
Concrete security bounds!

Thank you!

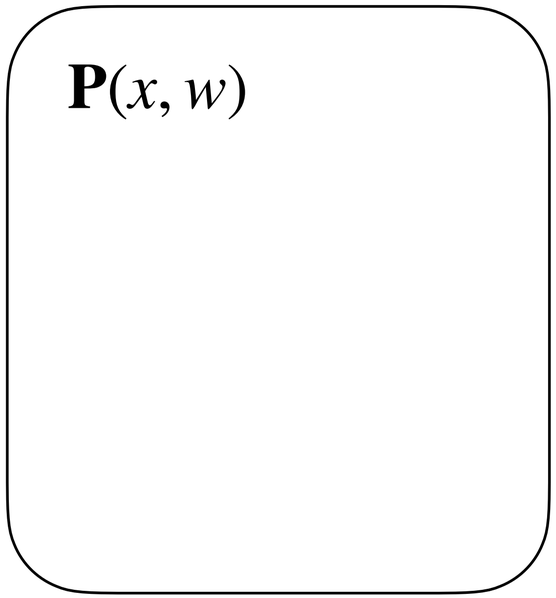
Extra slides

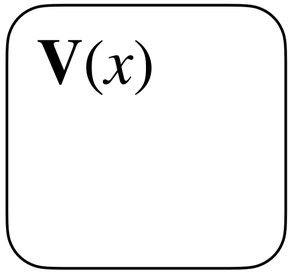
zkSNARKs (in the ROM)

zkSNARKs (in the ROM)

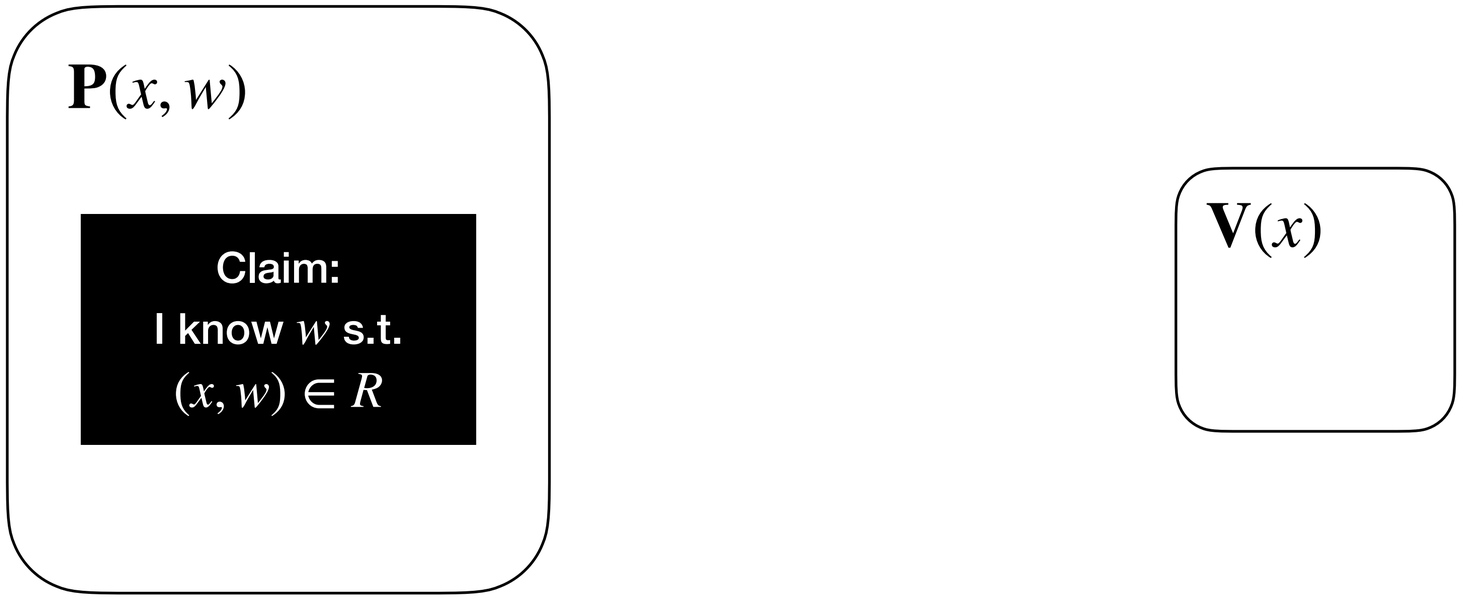


zkSNARKs (in the ROM)

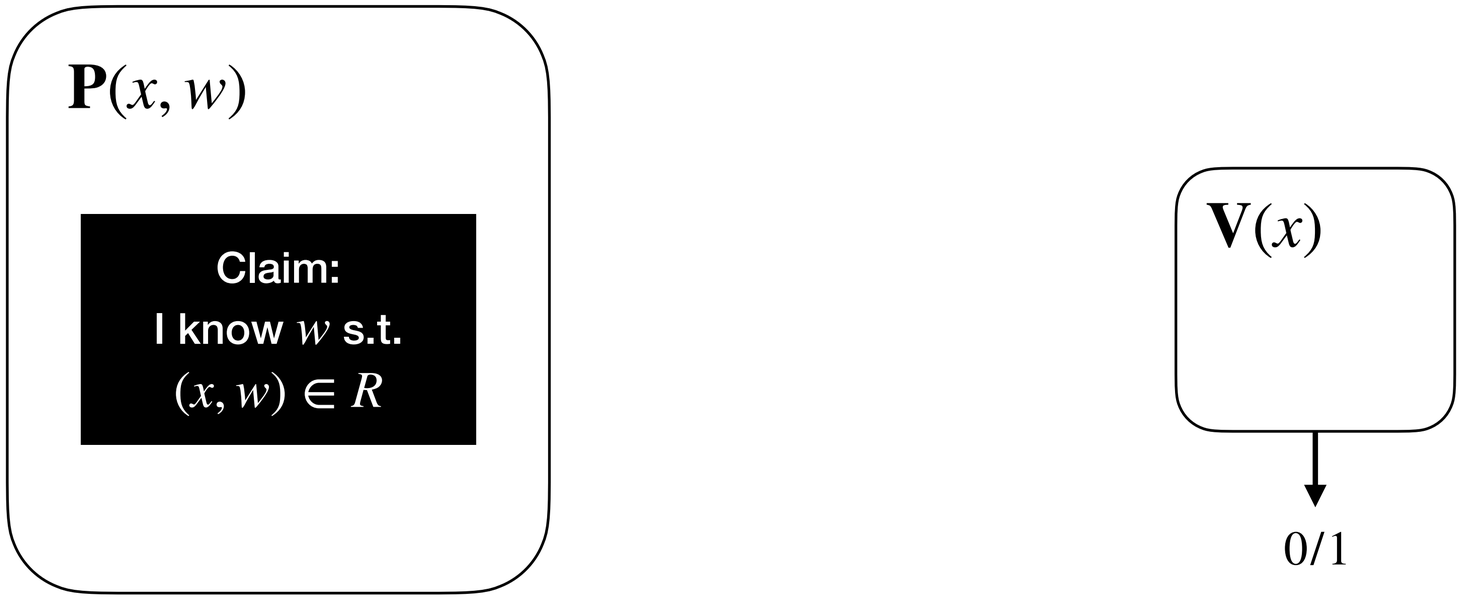




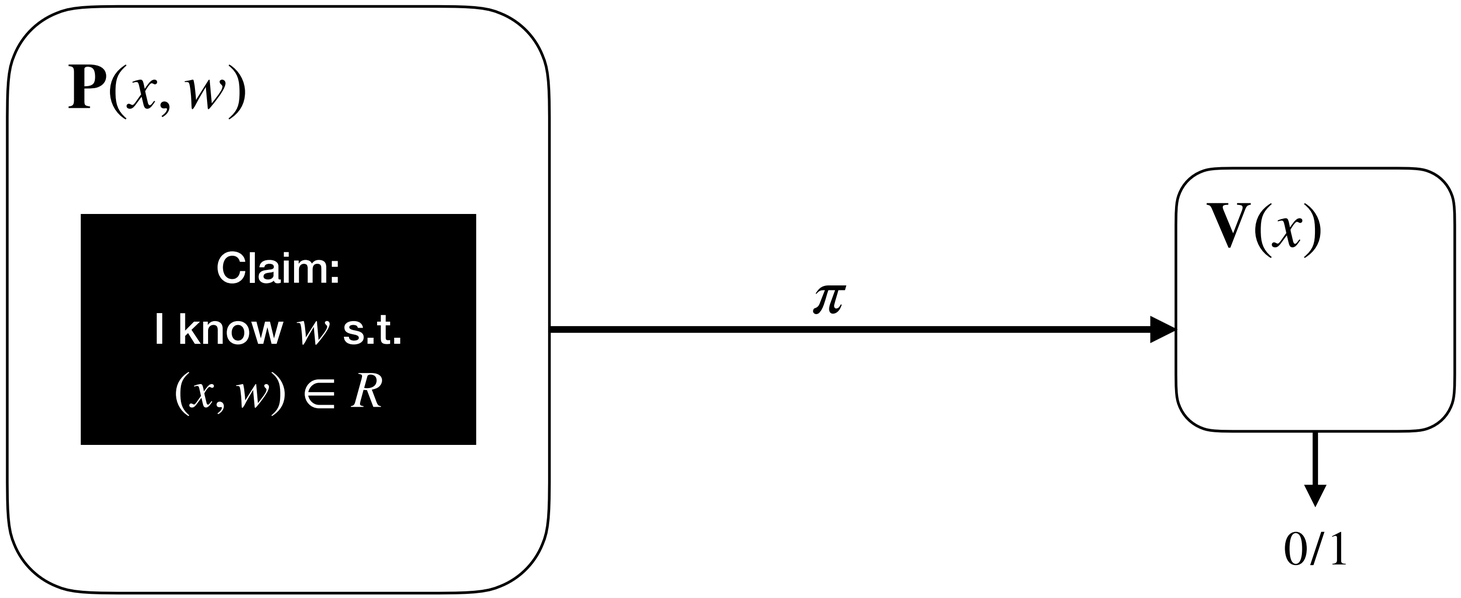
zkSNARKs (in the ROM)



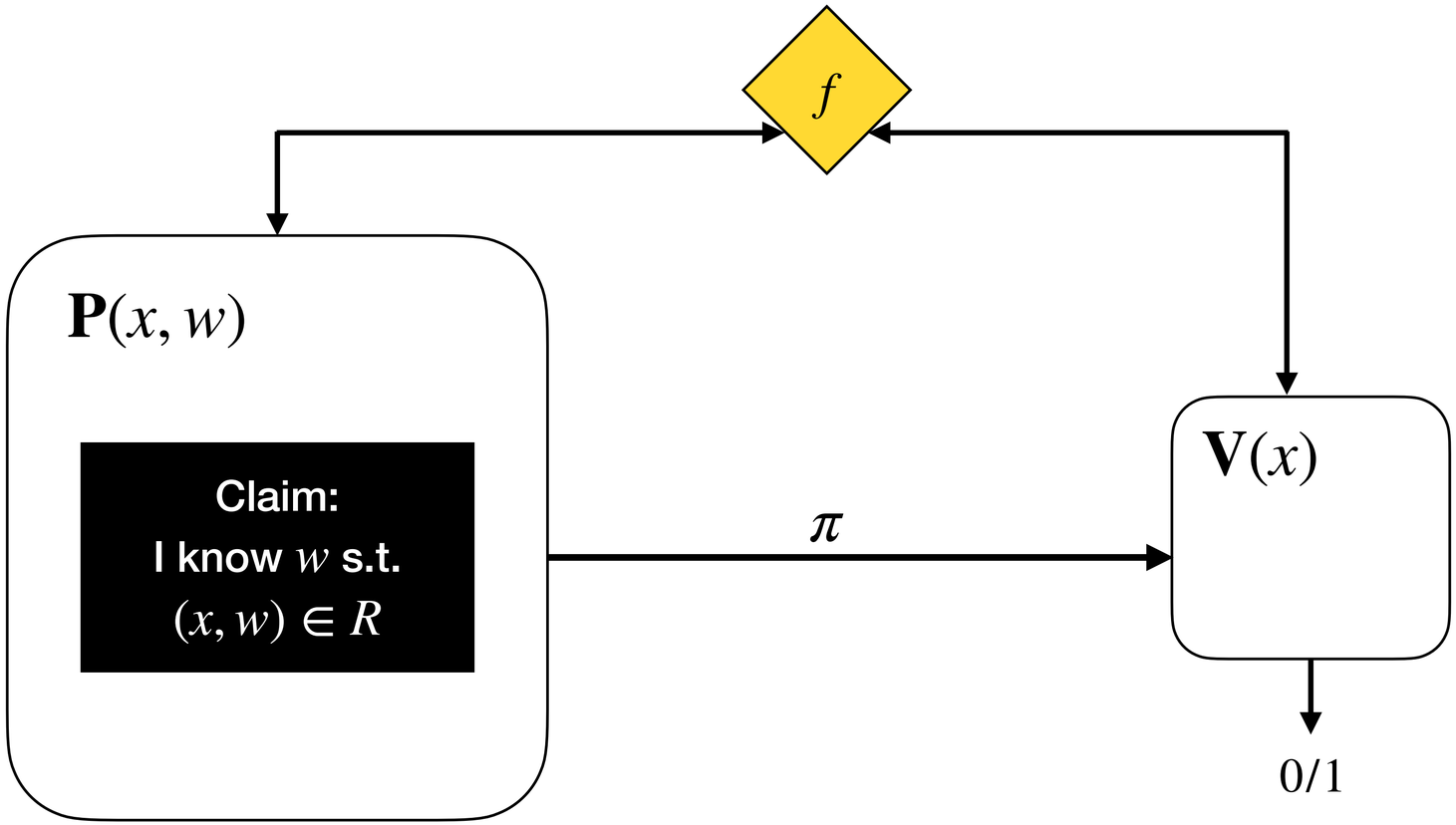
zkSNARKs (in the ROM)



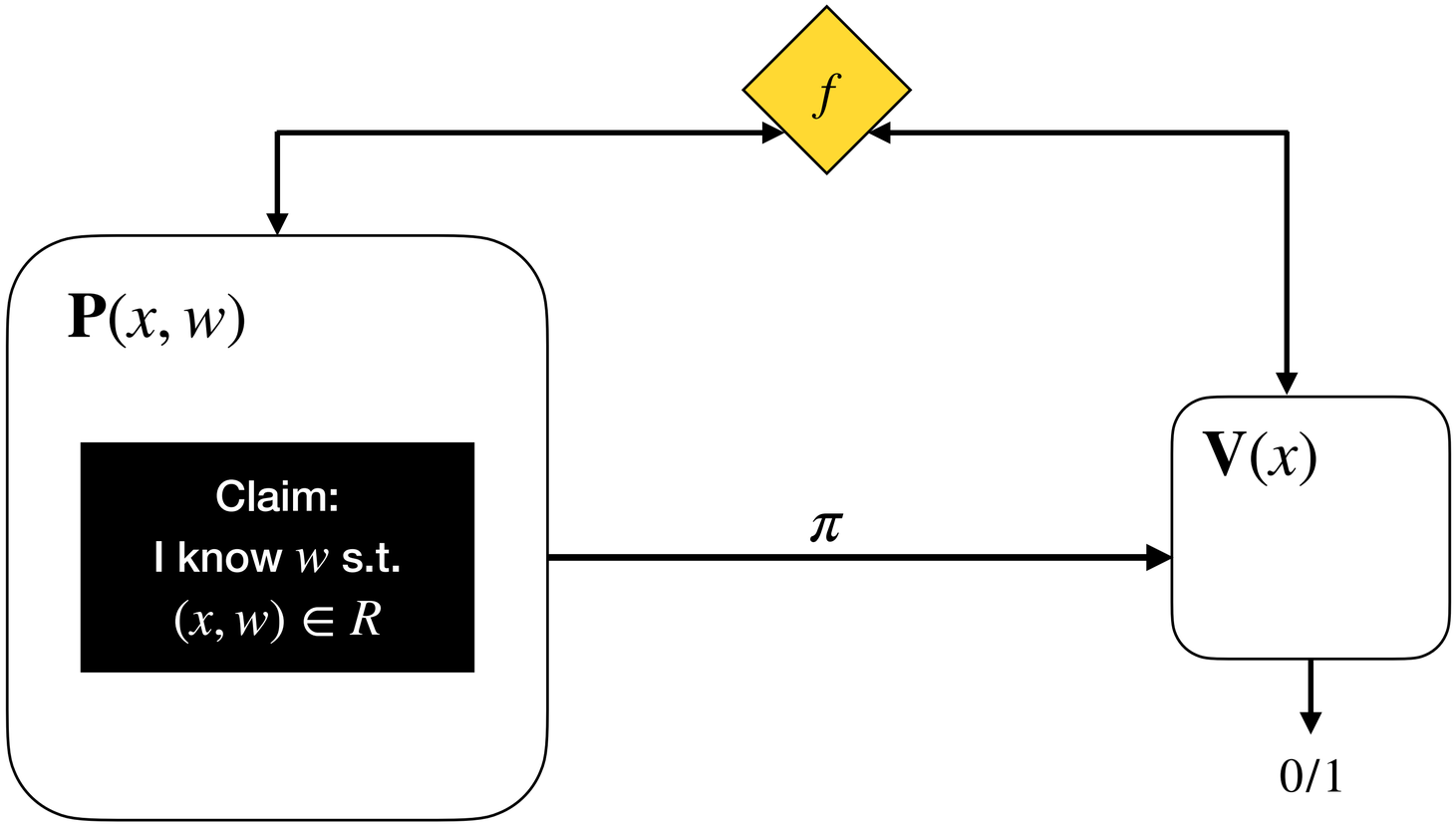
zkSNARKs (in the ROM)



zkSNARKs (in the ROM)

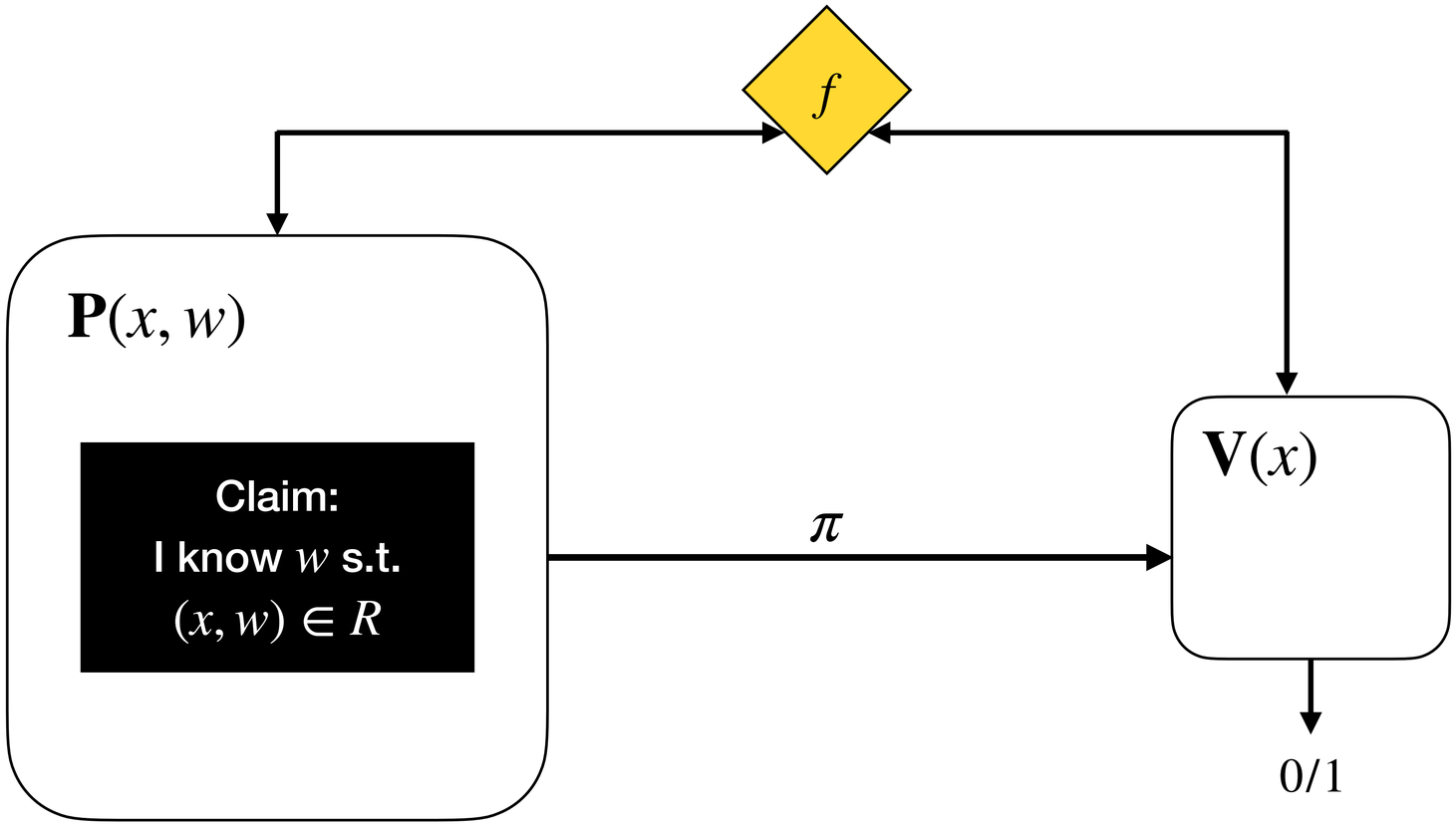


zkSNARKs (in the ROM)



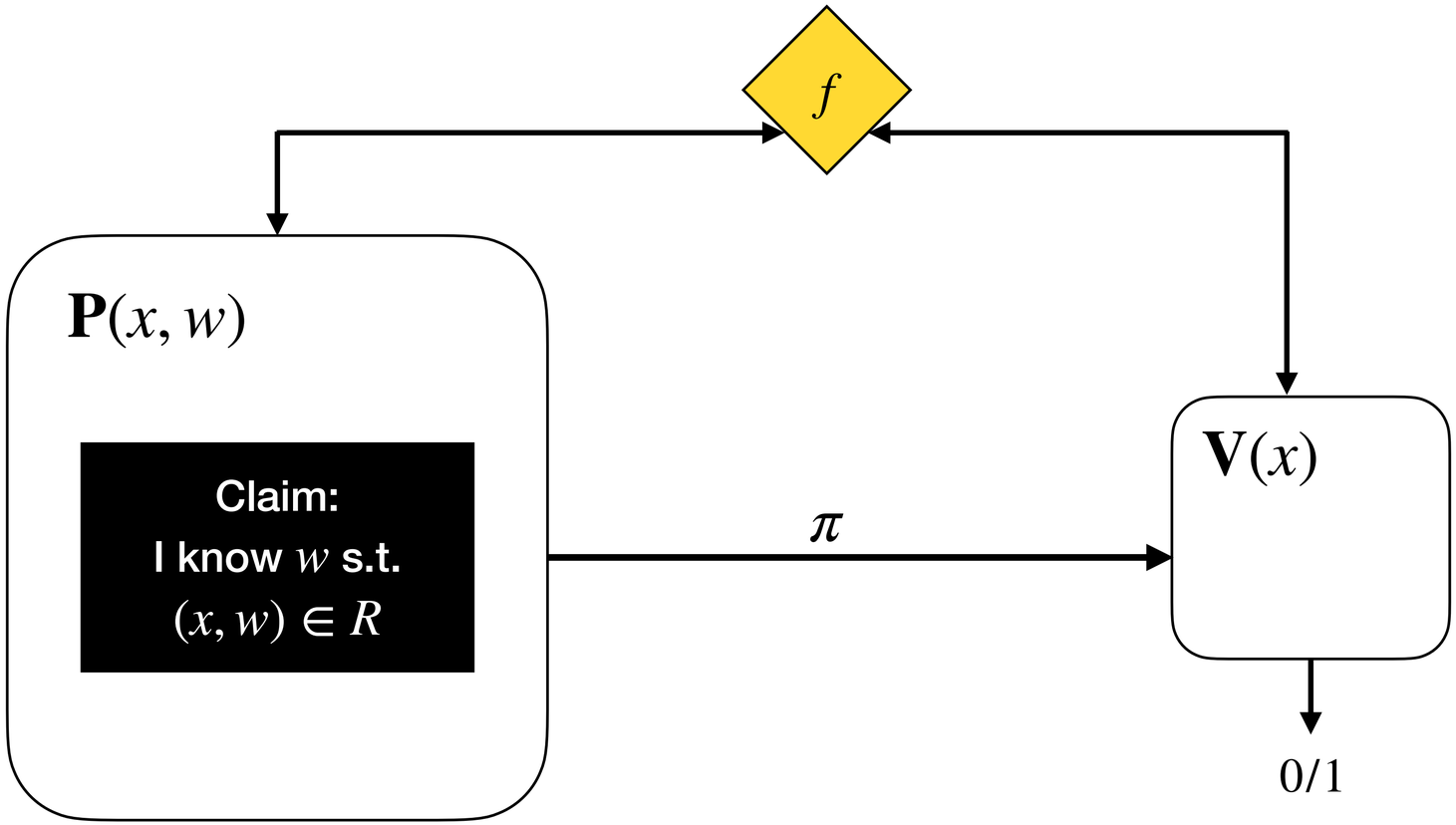
- Zero-Knowledge
  - $\exists \mathbf{S}: \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$

zkSNARKs (in the ROM)



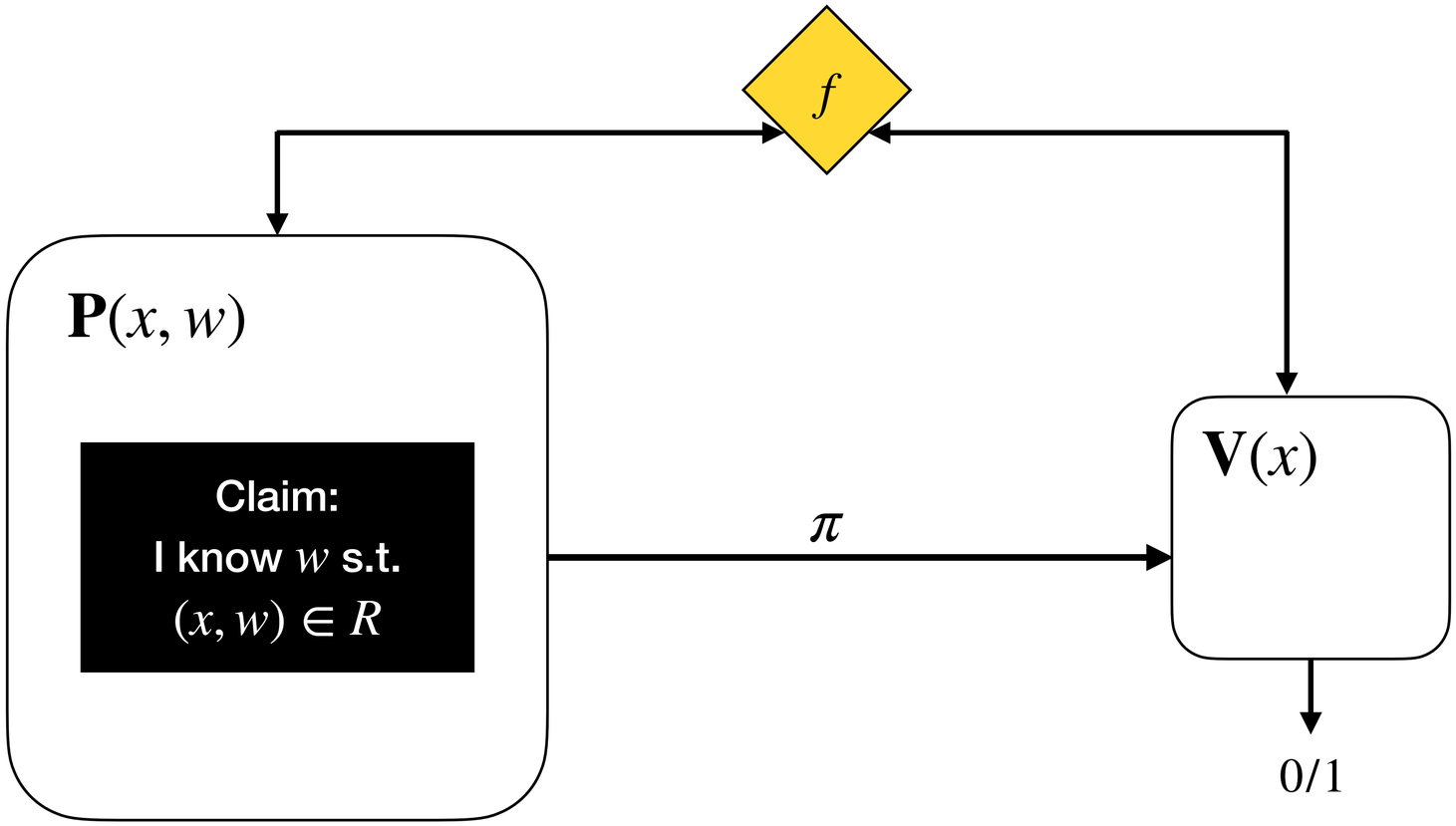
- Zero-Knowledge
  - $\exists \mathbf{S} : \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - $|\pi| \ll |w|$

zkSNARKs (in the ROM)



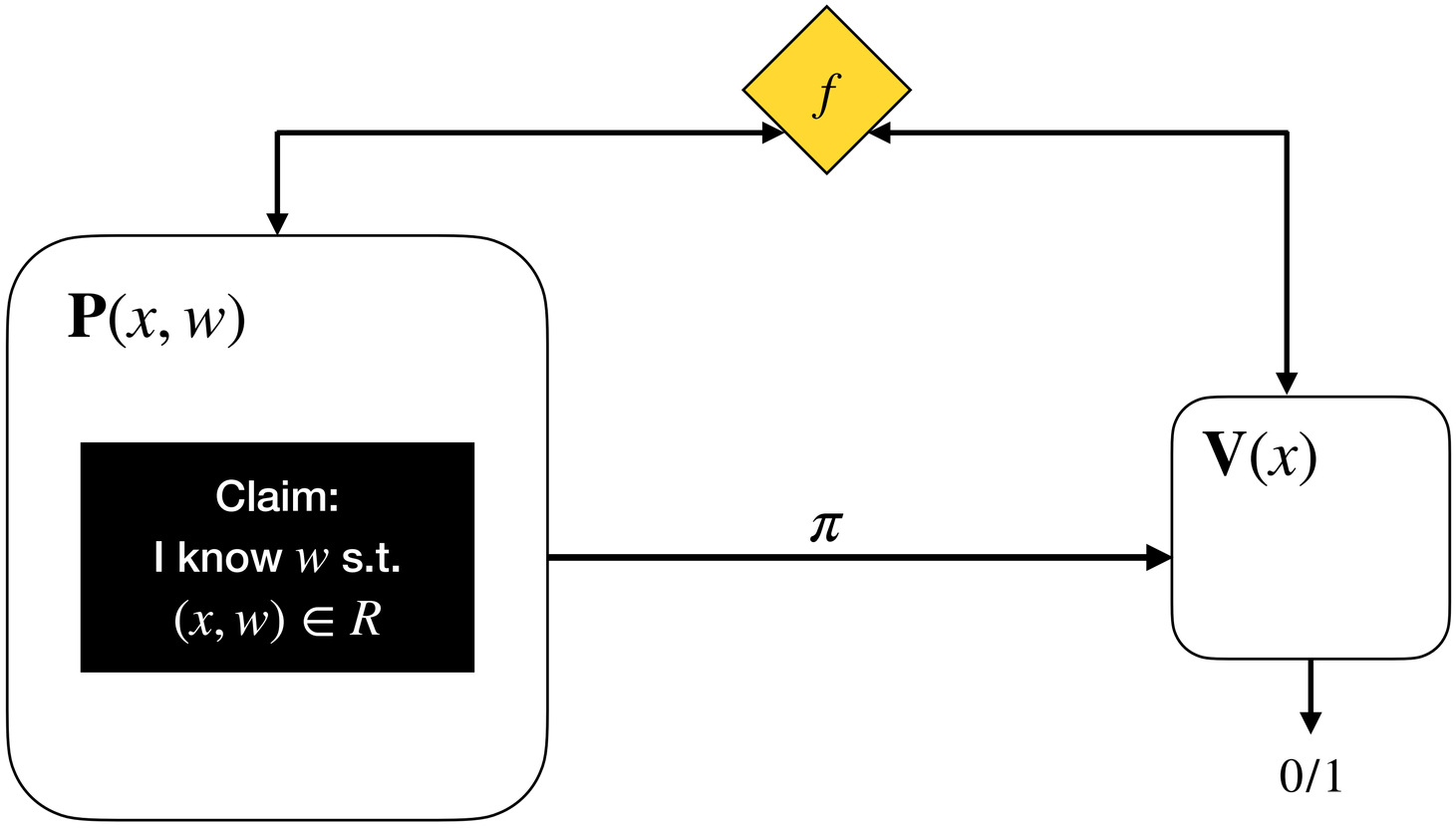
- Zero-Knowledge
  - $\exists \mathbf{S}: \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - $|\pi| \ll |w|$
- Non-interactive

zkSNARKs (in the ROM)



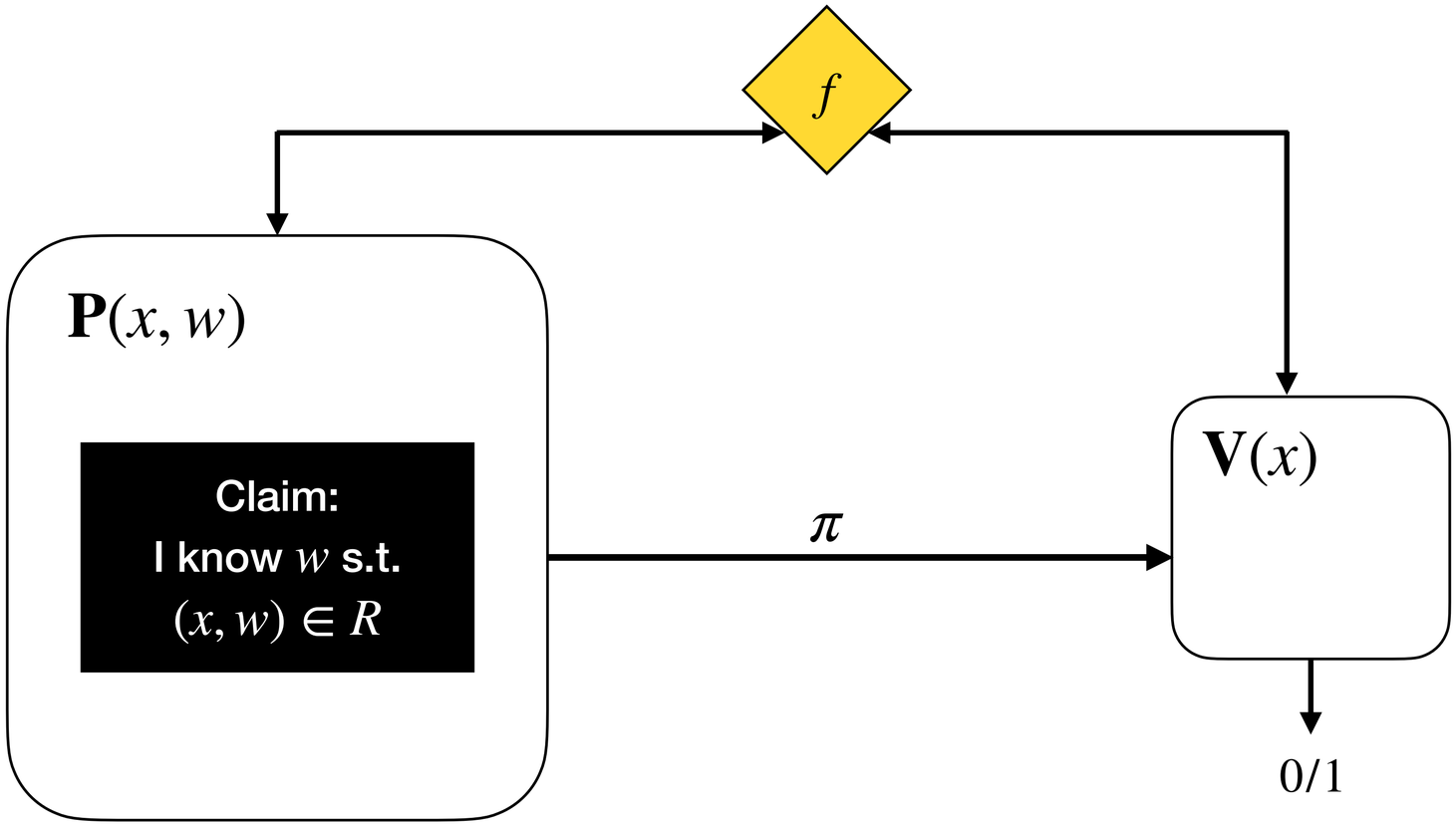
- Zero-Knowledge
  - $\exists \mathbf{S} : \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - $|\pi| \ll |w|$
- Non-interactive
- Argument of Knowledge

zkSNARKs (in the ROM)



- Zero-Knowledge
  - $\exists \mathbf{S} : \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - $|\pi| \ll |w|$
- Non-interactive
- Argument of Knowledge
  - $\exists \mathbf{E} : \ \mathbf{V}^f(x, \pi \leftarrow \tilde{\mathbf{P}}) = 1$

zkSNARKs (in the ROM)

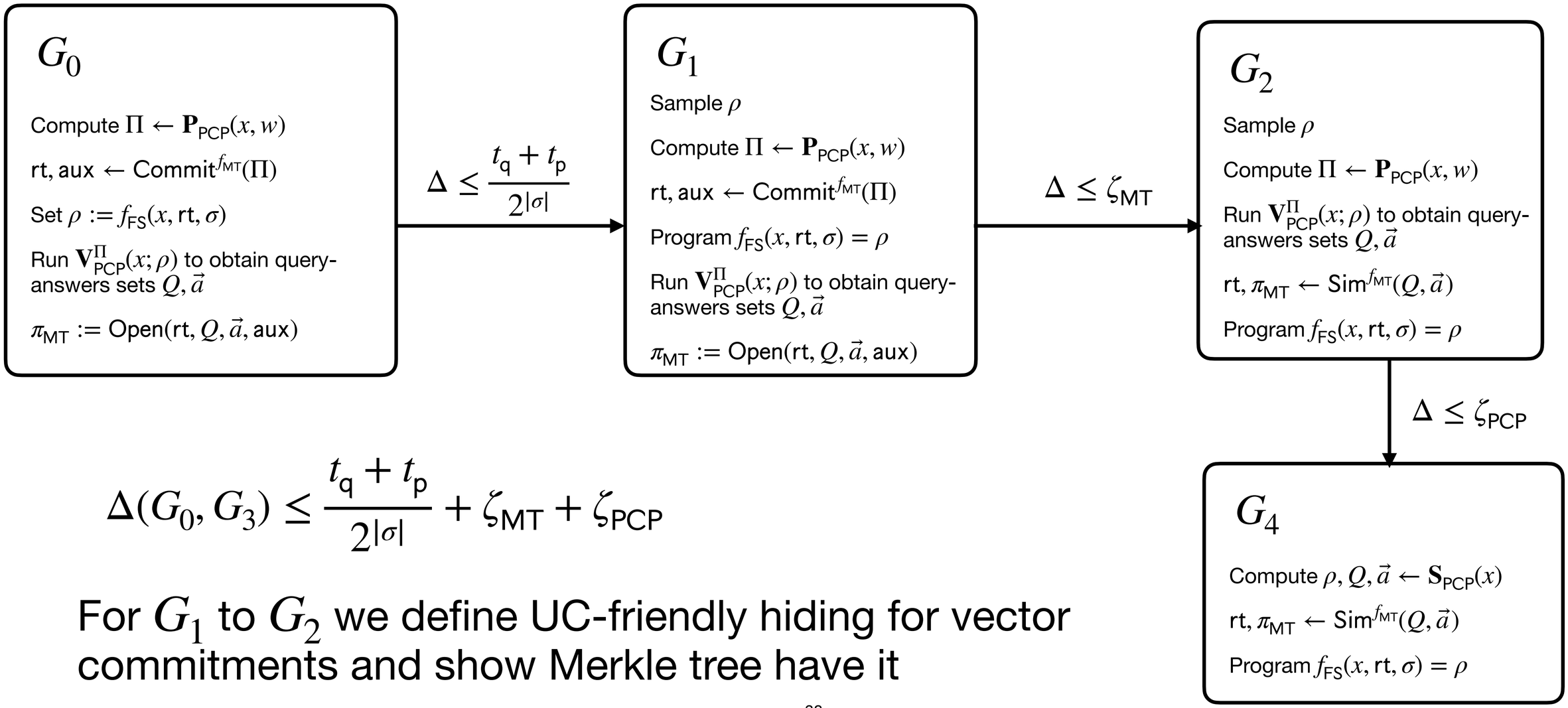


- Zero-Knowledge
  - $\exists \mathbf{S} : \ \mathbf{P}^f(x, w) \approx \mathbf{S}^f(x)$
- Succinct
  - $|\pi| \ll |w|$
- Non-interactive
- Argument of Knowledge
  - $\exists \mathbf{E} : \ \mathbf{V}^f(x, \pi \leftarrow \tilde{\mathbf{P}}) = 1$
    $\implies (x, \mathbf{E}(x, \pi, \mathsf{tr}_{\tilde{\mathbf{P}}})) \in R$

What if we only care about scalability?
Dropping ZK

- Often, SNARKs are deployed without ZK
- We consider this out of scope for this work but (at an high level) believe that:
  - The techniques here would still work and can be simplified.
  - Remove UC-friendly ZK and move to non-programmable GROM.
  - UC-completeness then reduces to perfect completeness.
  - Knowledge sound PCP/IOP suffices for Micali/BCS.

Micali has UC-friendly ZK



$$\Delta(G_0, G_3) \leq \frac{t_q + t_p}{2^{|\sigma|}} + \zeta_{\mathsf{MT}} + \zeta_{\mathsf{PCP}}$$

For G_1 to G_2 we define UC-friendly hiding for vector commitments and show Merkle tree have it

Micali has UC-friendly completeness

Micali has UC-friendly completeness

UC-friendly completeness

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.

UC-friendly completeness

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.

UC-friendly completeness

Perfect completeness
of the PCP

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.
- Disallow this attack with two natural properties:

UC-friendly completeness

Perfect completeness
of the PCP

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.
- Disallow this attack with two natural properties:
  - **Monotone proofs** (verifier does not query points not previously queried by the prover)

UC-friendly completeness

Perfect completeness
of the PCP

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.
- Disallow this attack with two natural properties:
  - **Monotone proofs** (verifier does not query points not previously queried by the prover)

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.
- Disallow this attack with two natural properties:
  - **Monotone proofs** (verifier does not query points not previously queried by the prover)
  - **Unpredictable queries** (hard to program points prover will query)

Micali has UC-friendly completeness

- Assuming **PCP perfect completeness**, honest proof are rejected only if the verifier queries a previously programmed point.
- Disallow this attack with two natural properties:
  - **Monotone proofs** (verifier does not query points not previously queried by the prover)
  - **Unpredictable queries** (hard to program points prover will query)

Micali has UC-friendly KS

Micali has UC-friendly KS

UC-friendly KS of Micali

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.

UC-friendly KS of Micali

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.

UC-friendly KS of Micali

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.

UC-friendly KS of Micali

Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.
- Reduce to **state-restoration KS** (implied by KS of PCP)

UC-friendly KS of Micali

Micali has UC-friendly KS

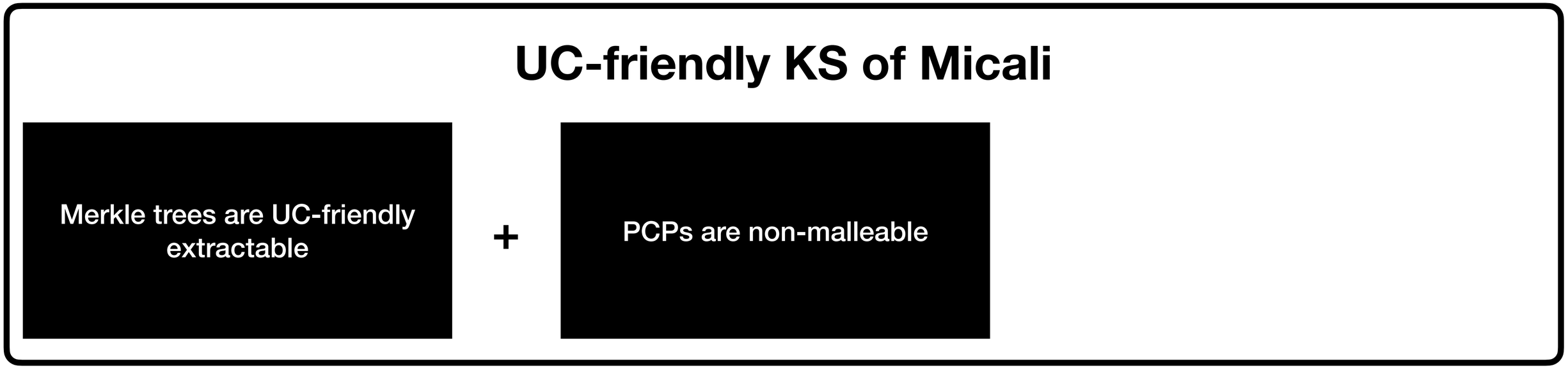
- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.
- Reduce to **state-restoration KS** (implied by KS of PCP)

UC-friendly KS of Micali

Merkle trees are UC-friendly extractable

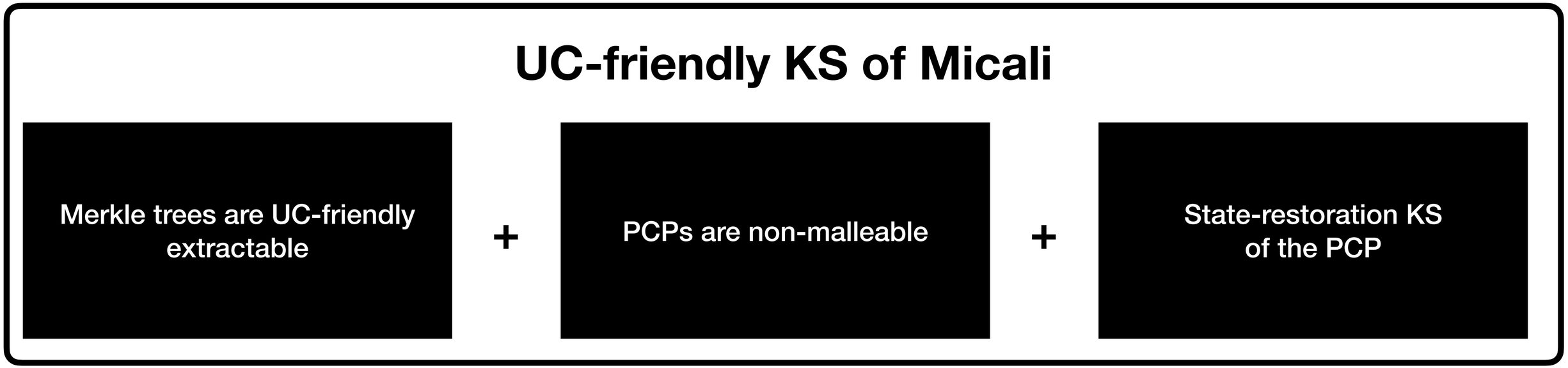
Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.
- Reduce to **state-restoration KS** (implied by KS of PCP)



Micali has UC-friendly KS

- UC-friendly KS implies simulation-extractability.
- Merkle trees are **non-malleable** already.
- In Micali, makes proofs **non-malleable**.
- Reduce to **state-restoration KS** (implied by KS of PCP)



Related works

Known UC-secure zkSNARKs

Related works

Known UC-secure zkSNARKs

Non-Witness Succinct

Related works

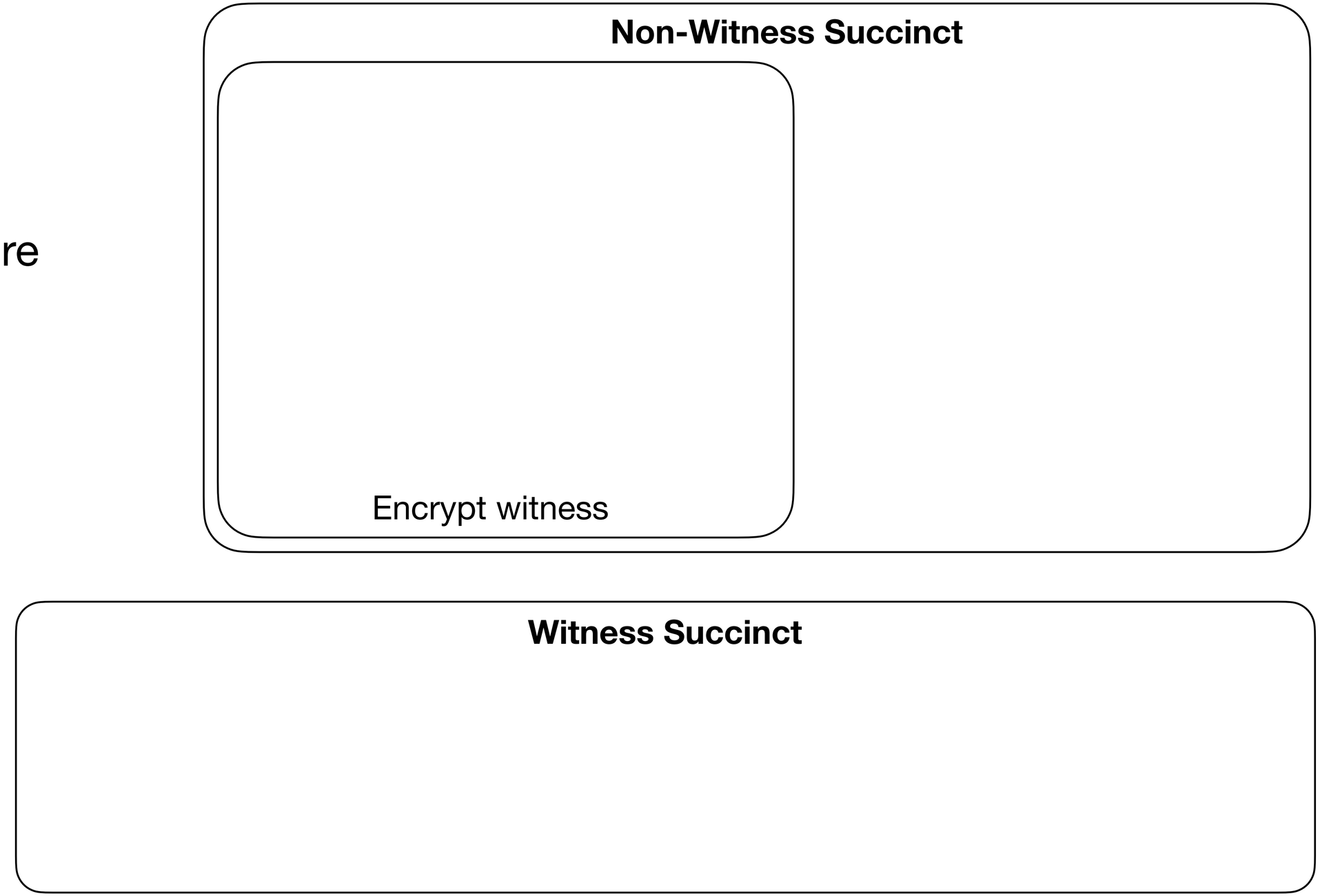
Known UC-secure zkSNARKs

Non-Witness Succinct

Witness Succinct

Related works

Known UC-secure zkSNARKs



Related works

Known UC-secure zkSNARKs

Non-Witness Succinct

CØCØ: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]

Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically*

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]

TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat

Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Encrypt witness

Witness Succinct

Related works

Known UC-secure zkSNARKs

Non-Witness Succinct

CØCØ: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡] T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•] Elaine Shi[‡]

Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically*

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]

TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat

Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Encrypt witness

Compile Σ-protocol

Witness Succinct

Related works

Known UC-secure zkSNARKs

Non-Witness Succinct

C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]

Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically*

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]

TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat

Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Encrypt witness

Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Compile Σ-protocol

Witness Succinct

Related works

Known UC-secure zkSNARKs

Non-Witness Succinct

C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]

Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically*

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]

TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat

Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Encrypt witness

Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Compile Σ-protocol

Witness Succinct

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

Commit witness using PCS

Related works

Known UC-secure zkSNARKs

Non-Witness Succinct

C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡]
T-H. Hubert Chan[*] Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•]
Elaine Shi[‡]

Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically*

Behzad Abdolmaleki[1], Sebastian Ramacher[2], and Daniel Slamanig[2]

TIRAMISU: Black-Box Simulation Extractable NIZKs in the Updatable CRS Model

Karim Baghery and Mahdi Sedaghat

Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

Behzad Abdolmaleki[1], Noemi Glaeser[1,2], Sebastian Ramacher[3], and Daniel Slamanig[3]

Encrypt witness

Universally Composable Σ-protocols in the Global Random-Oracle Model

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions

Anna Lysyanskaya and
Leah Namisa Rosenbloom

Compile Σ-protocol

Witness Succinct

Witness-Succinct Universally-Composable SNARKs*

Chaya Ganesh[1], Yashvanth Kondi[2], Claudio Orlandi[2], Mahak Pancholi[2], Akira Takahashi[3], and Daniel Tschudi[4]

Commit witness using PCS

zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

This work!

Challenge I

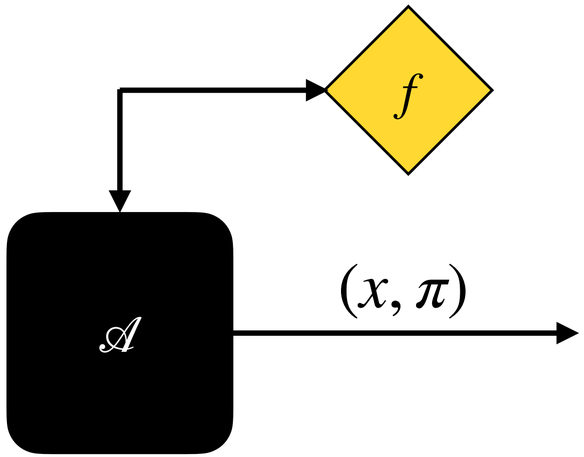
Challenge I

Rewinding

extractor

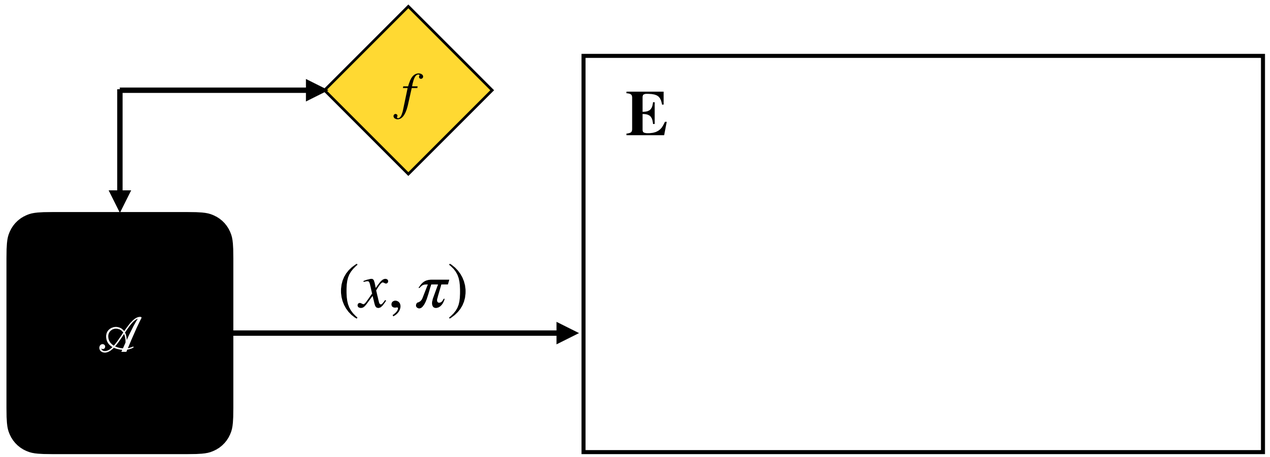
Challenge I

Rewinding extractor



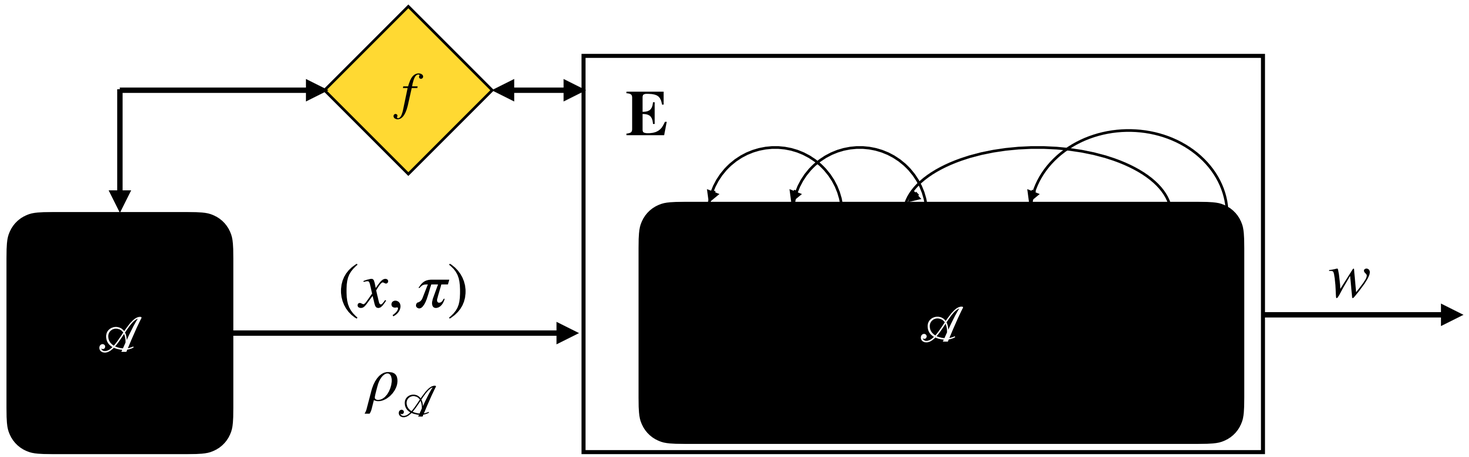
Challenge I

Rewinding extractor



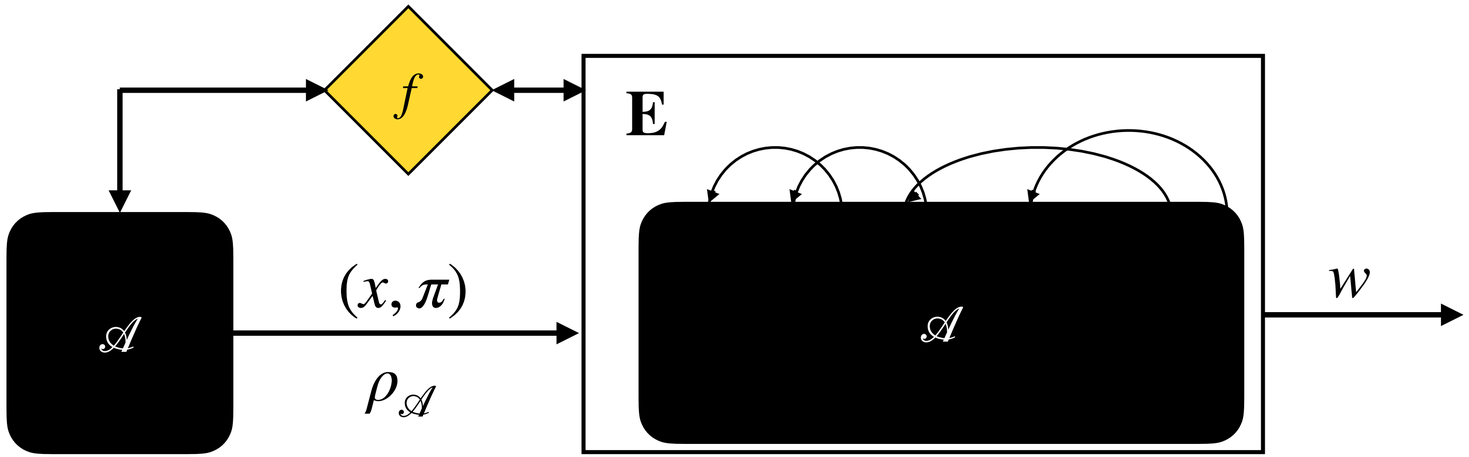
Challenge I

Rewinding extractor



Challenge I

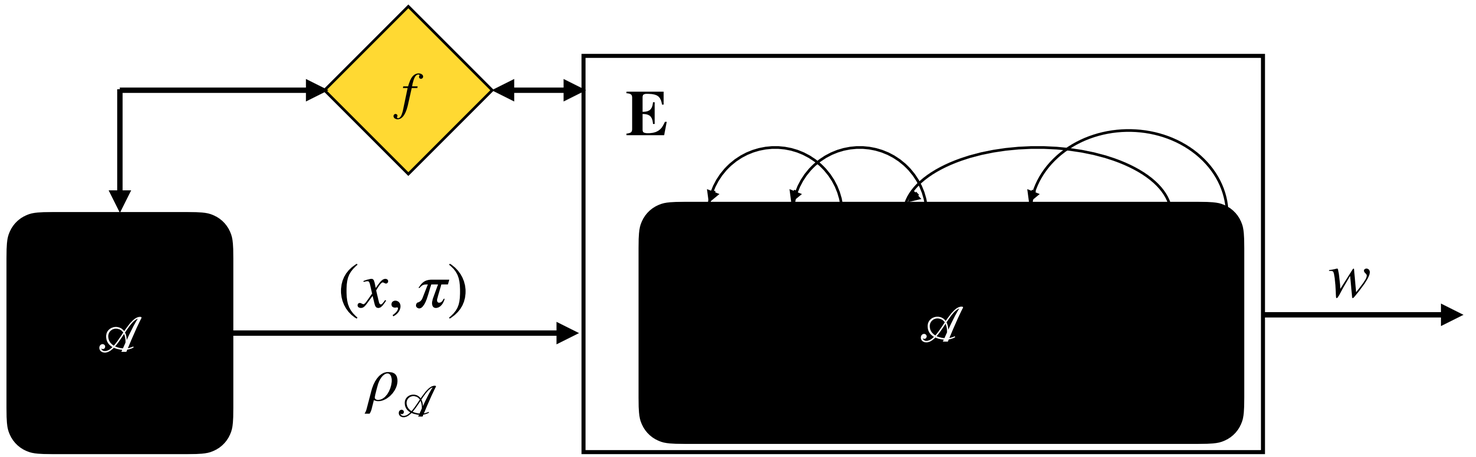
Rewinding extractor



Not allowed in UC!

Challenge I

Rewinding extractor

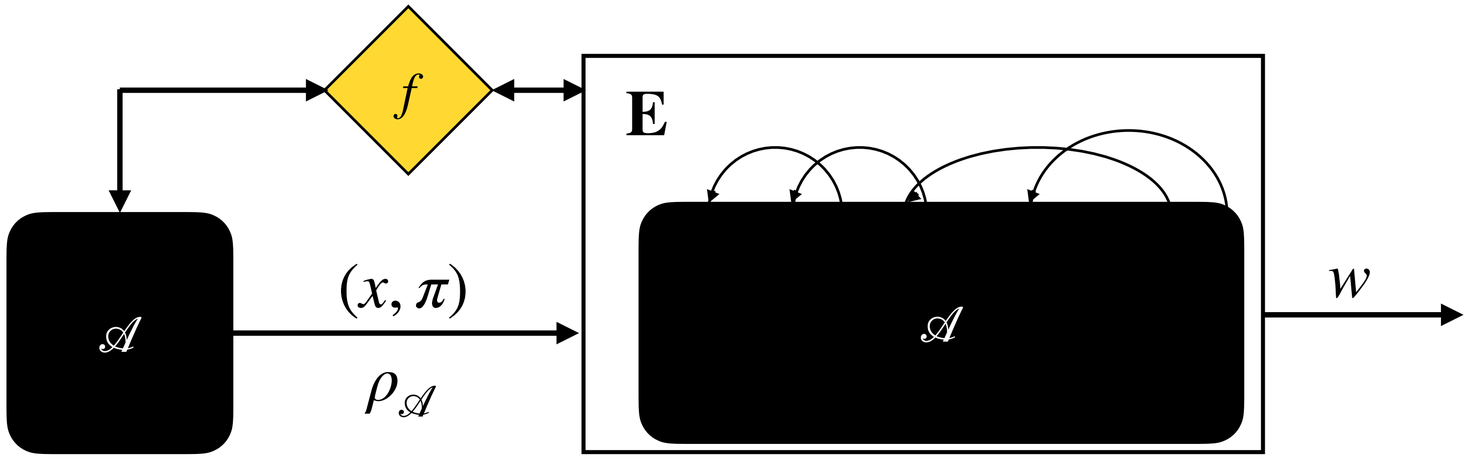


Not allowed in UC!

For UC-security, extractor must be **black-box** and **straight-line**, as we cannot rewind the environment, and security is $\exists \mathcal{S} \forall \mathcal{E}$

Challenge I

Rewinding extractor



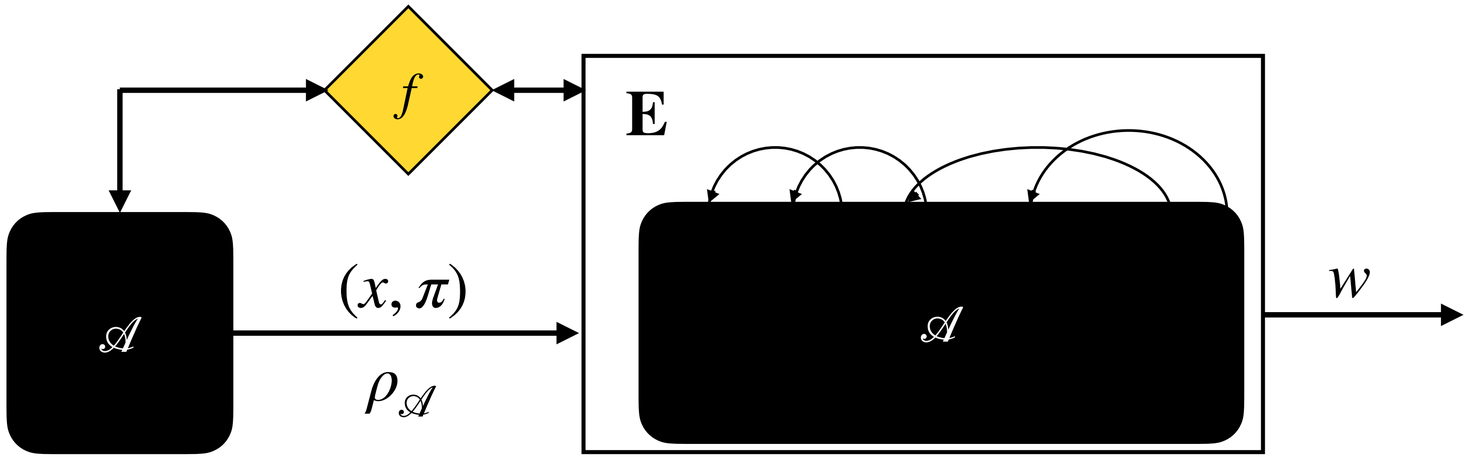
Not allowed in UC!

Straightline (black-box) extractor

For UC-security, extractor must be **black-box** and **straight-line**, as we cannot rewind the environment, and security is $\exists \mathcal{S} \forall \mathcal{E}$

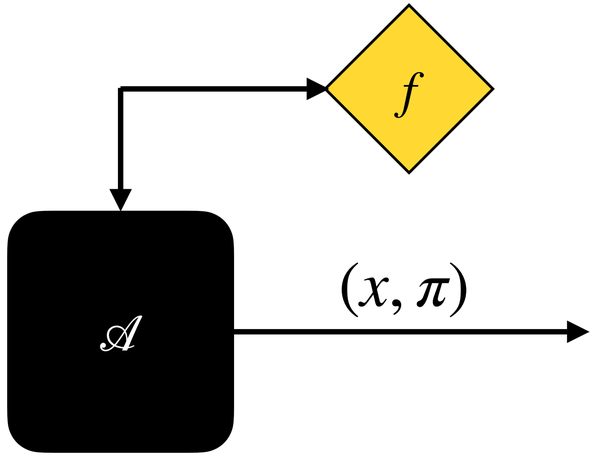
Challenge I

Rewinding extractor



Not allowed in UC!

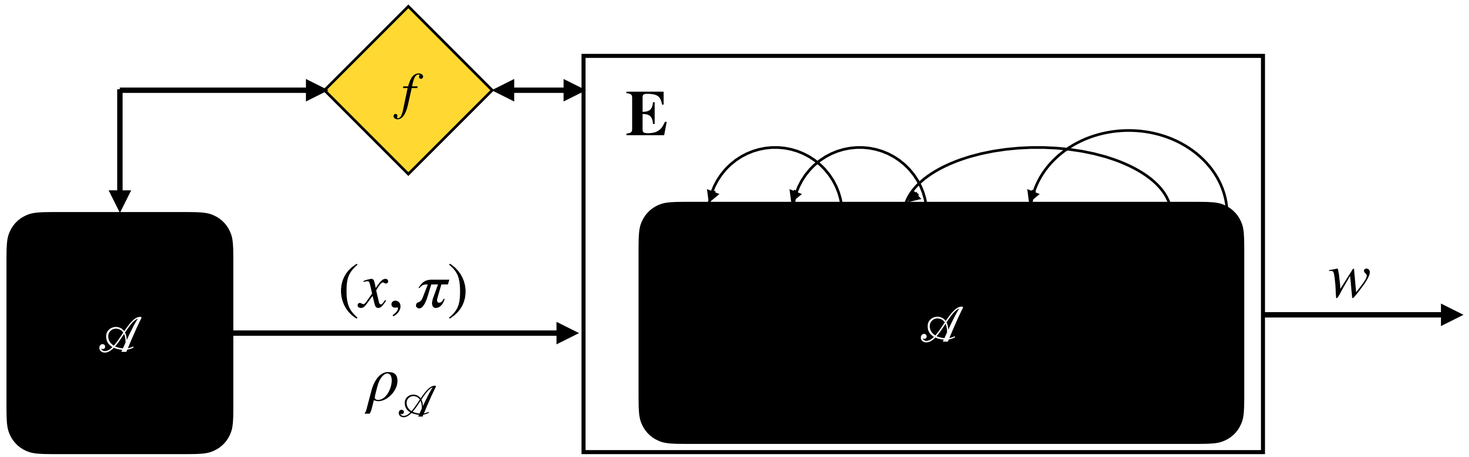
Straightline (black-box) extractor



For UC-security, extractor must be **black-box** and **straight-line**, as we cannot rewind the environment, and security is $\exists \mathcal{S} \forall \mathcal{E}$

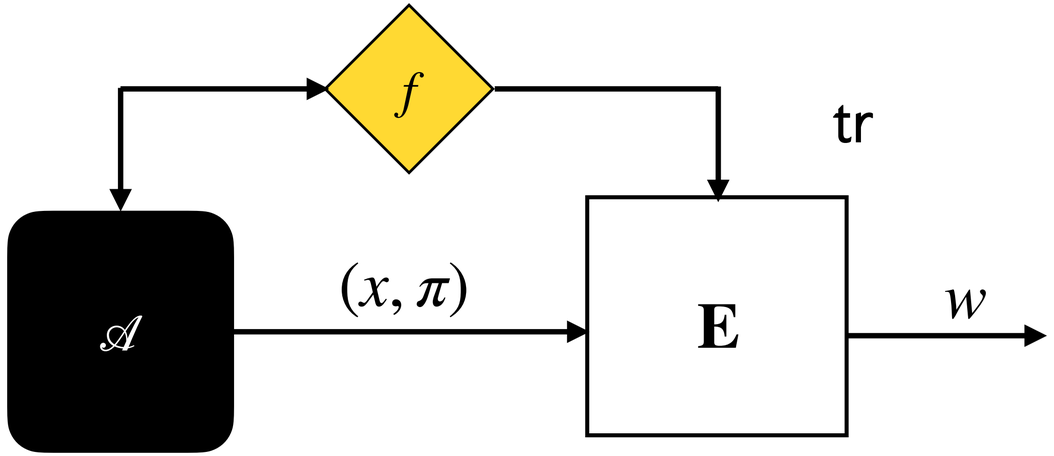
Challenge I

Rewinding extractor



Not allowed in UC!

Straightline (black-box) extractor



For UC-security, extractor must be **black-box** and **straight-line**, as we cannot rewind the environment, and security is $\exists \mathcal{S} \forall \mathcal{E}$

Challenge II

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

Challenge II

Our $\mathcal{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathcal{A}$

$$\Pr\left[\quad\middle|\quad\right] \leq \kappa$$

Challenge II

Our $\mathcal{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathcal{A}$

$$\Pr\left[\;\middle|\; \mathcal{A} \;\right] \leq \kappa$$

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

$$\Pr\left[\; \mathscr{A}^{\,f} \;\right] \leq \kappa$$

Challenge II

Our $\mathscr{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathscr{A}$

$$\Pr\left[\; \mathscr{A} \text{ with } f \;\right] \leq \kappa$$

Challenge II

Our $\mathcal{F}_{\mathrm{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathcal{A}$

$$\Pr\left[\begin{array}{l} \mathbf{Sim}(x) \\ \pi, \mathsf{tr}_\pi \leftarrow \mathbf{S}^f(x), \mathcal{Q} := \mathcal{Q} \cup \{\pi\} \\ \text{Program } f \text{ according to } \mathsf{tr}_\pi \\ \mathcal{A} \leftrightarrow f \end{array}\right] \leq \kappa$$

Challenge II

Our $\mathcal{F}_{\text{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathcal{A}$

$$\Pr\left[\begin{array}{l} \mathbf{Sim}(x) \\ \pi, \mathsf{tr}_\pi \leftarrow \mathbf{S}^f(x), \mathcal{Q} := \mathcal{Q} \cup \{\pi\} \\ \text{Program } f \text{ according to } \mathsf{tr}_\pi \\ \mathcal{A} \leftrightarrow f \\ \mathcal{A} \rightarrow (x, \pi) \end{array}\right] \leq \kappa$$

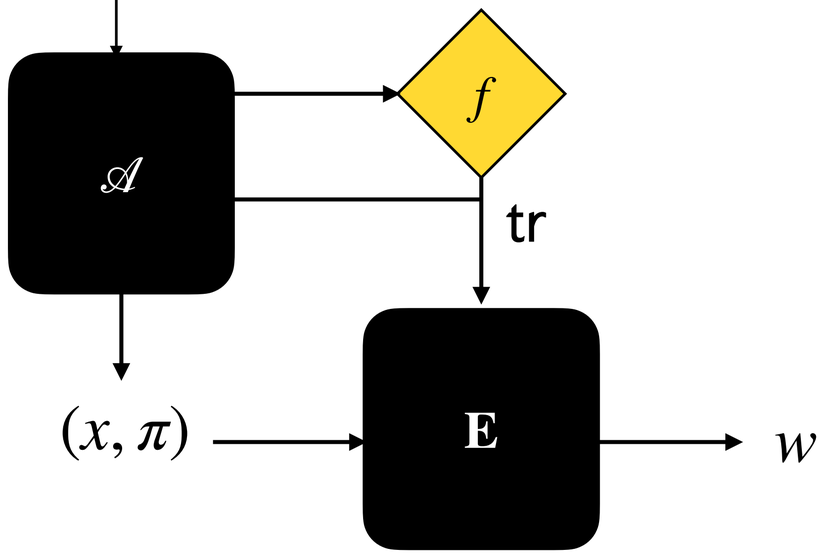
Challenge II

Our $\mathcal{F}_{\text{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathcal{A}$

$$\Pr\left[\left[\begin{array}{l}\mathbf{Sim}(x)\\ \pi, \mathsf{tr}_\pi \leftarrow \mathbf{S}^f(x), \mathcal{Q} := \mathcal{Q} \cup \{\pi\}\\ \text{Program } f \text{ according to } \mathsf{tr}_\pi\end{array}\right] \leq \kappa\right]$$



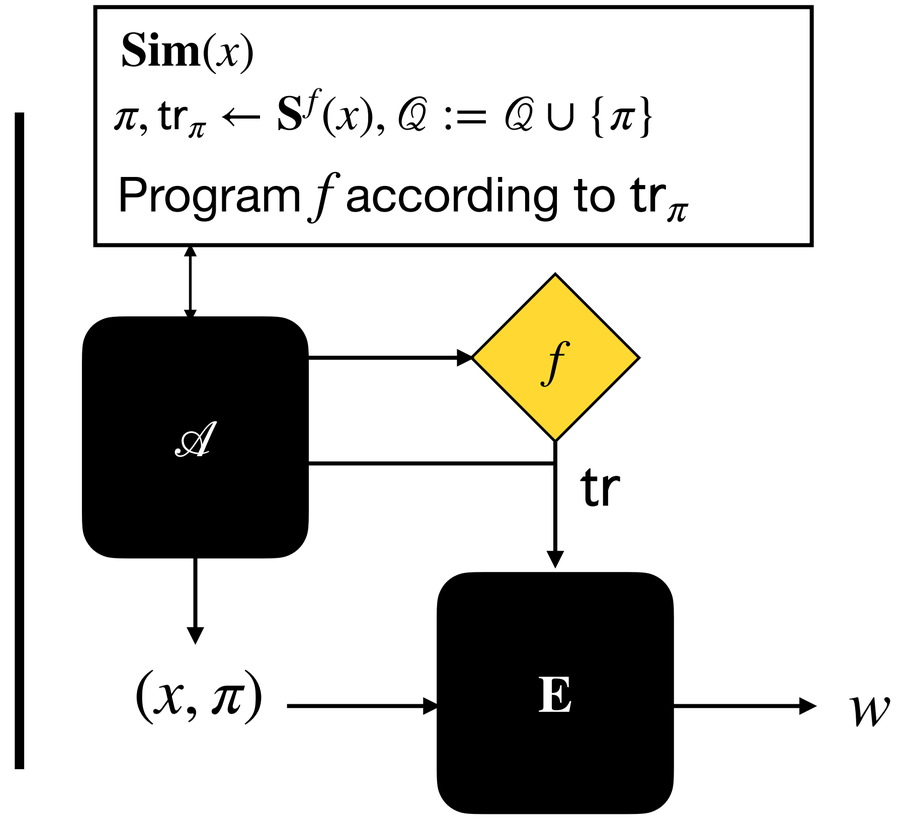
Challenge II

Our $\mathcal{F}_{\text{ARG}}$ gives access to simulated proofs.

Attack: The adversary could use them to “forge” new proofs.

Want: $\exists \mathbf{E}$ straightline s.t. $\forall \mathcal{A}$

$$\Pr\left[\mathbf{V}^f(x,\pi) = 1 \;\wedge\; (x,w) \notin R \;\wedge\; \pi \notin \mathcal{Q}\right] \leq \kappa$$

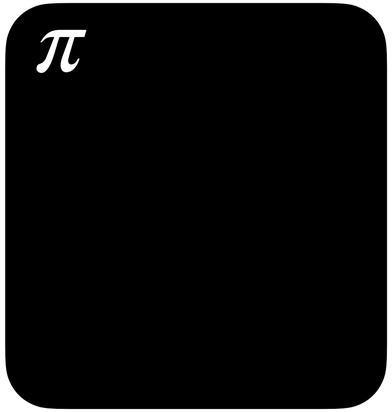


UC with Budgets

Plain UC only models adversaries that are **computationally** bounded

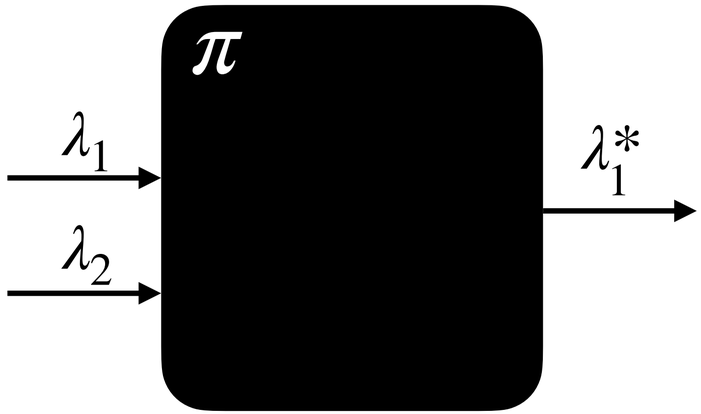
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



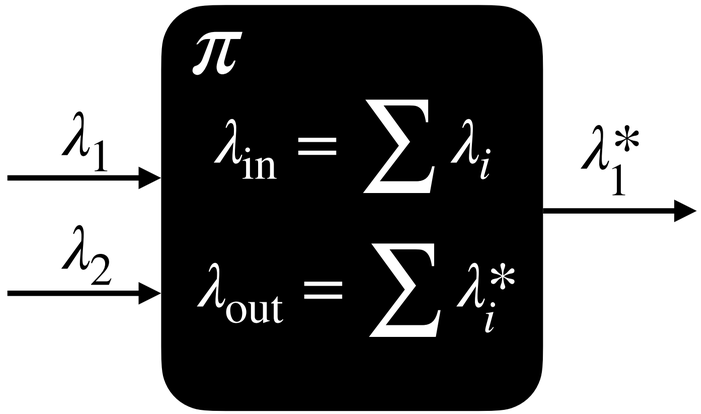
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



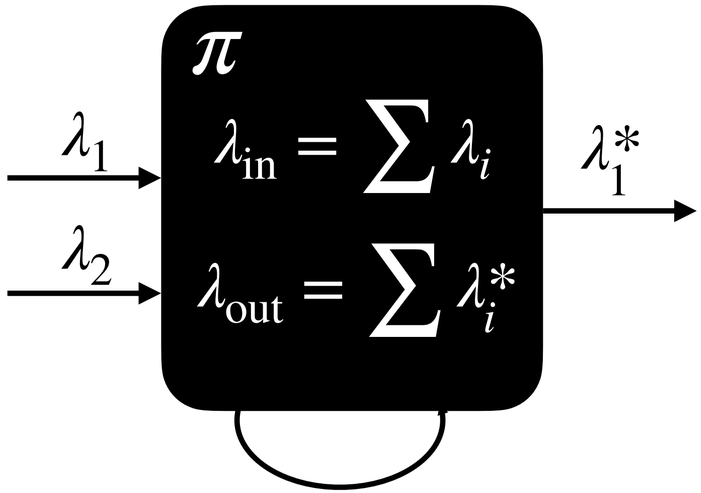
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



UC with Budgets

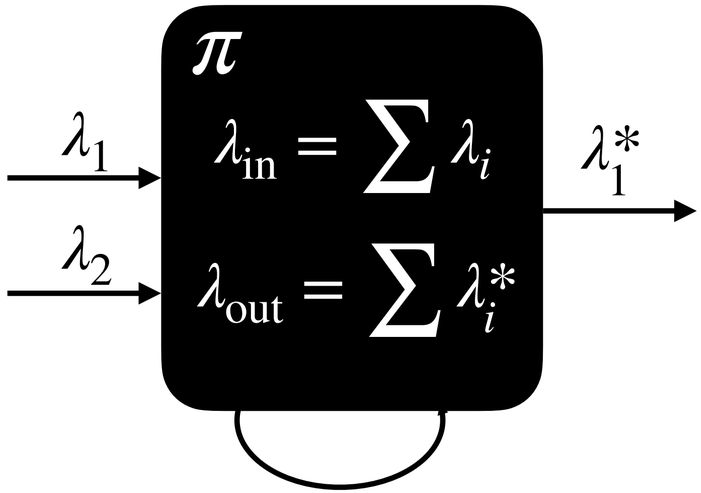
Plain UC only models adversaries that are **computationally** bounded



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded

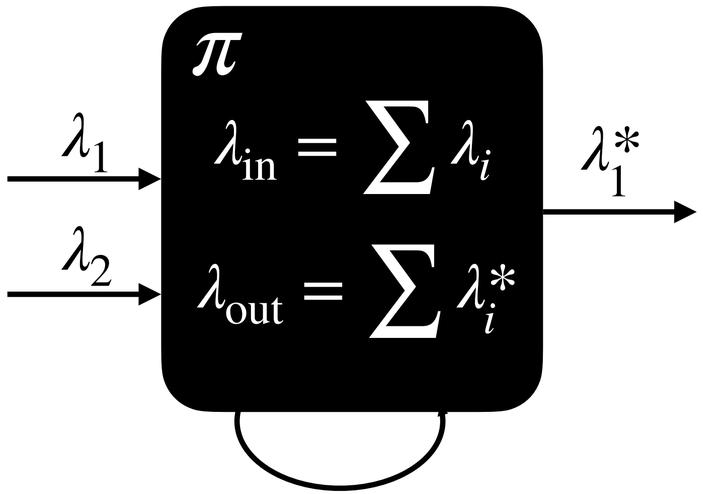


$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**

UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**

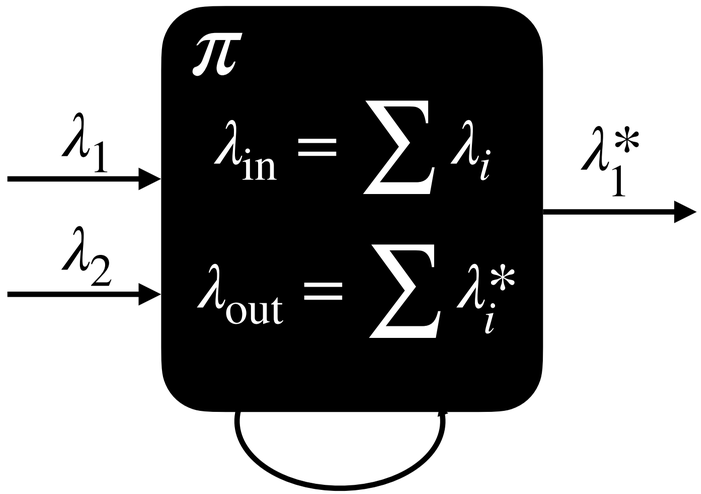
π

$$\mathscr{B} = \mathscr{B}_{\text{start}} + \sum \mathscr{B}_{\text{in}}$$

$$\mathscr{B} = (t_{\text{q}}, t_{\text{p}}, \ell_{\text{p}}, \ell_{\text{v}})$$

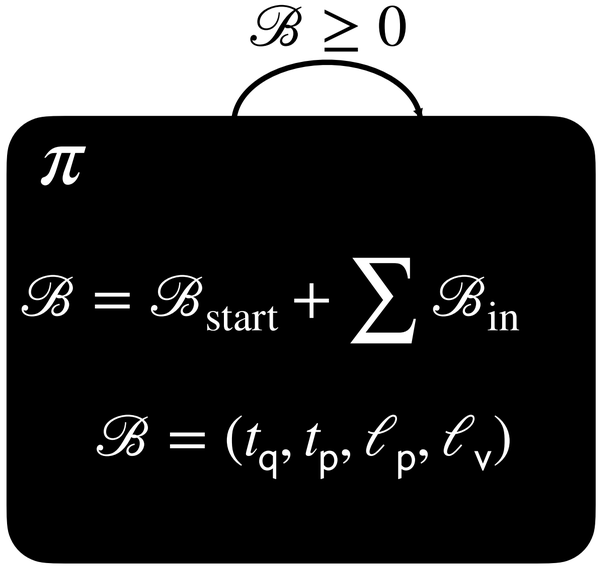
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



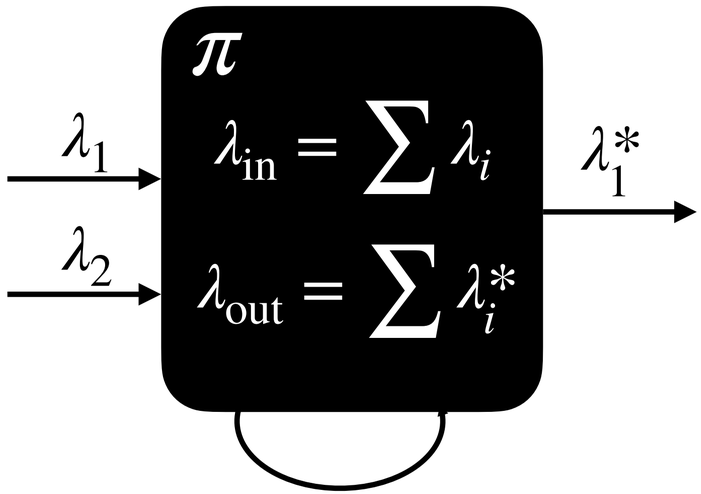
$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**



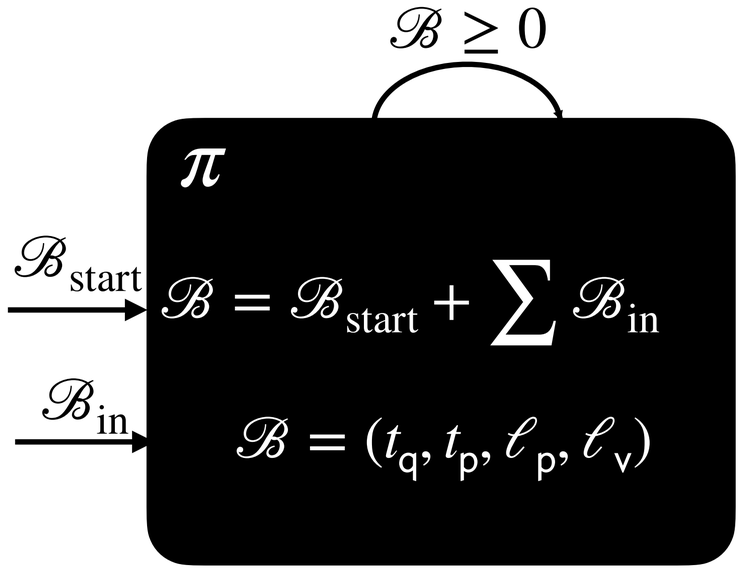
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



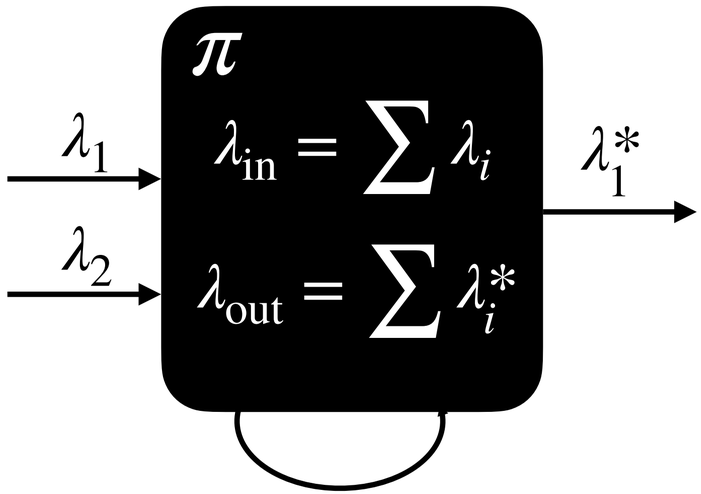
$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**



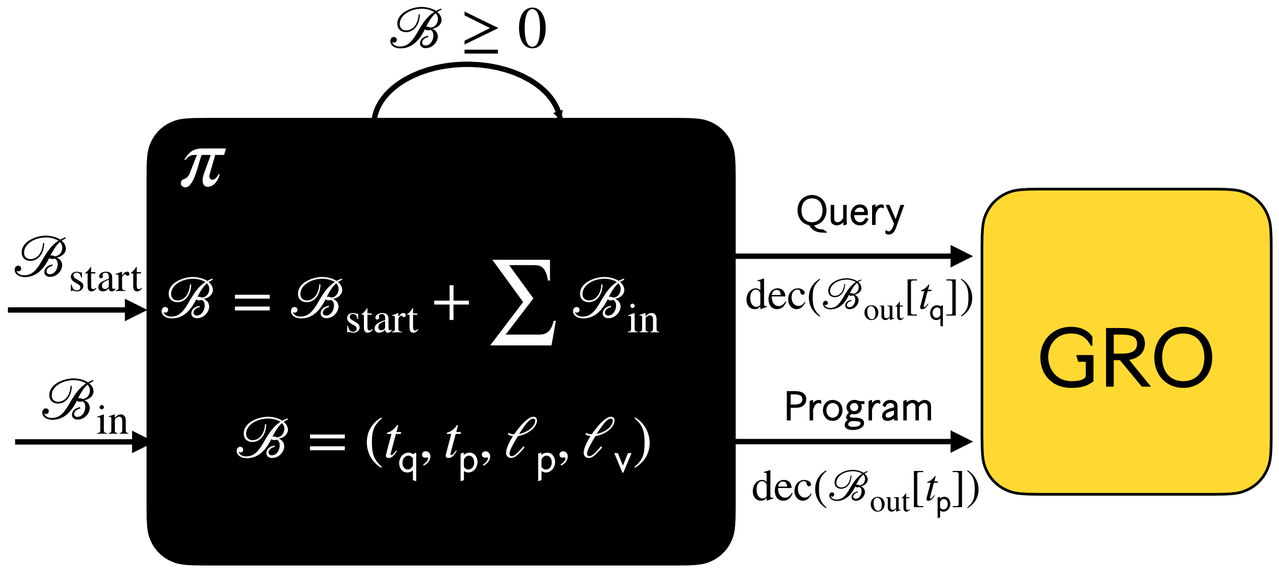
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

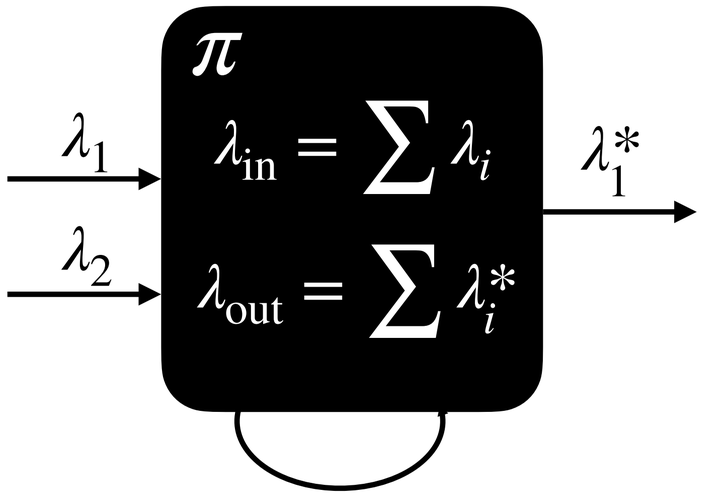
We consider adversaries that are **resource** bounded and computationally **unbounded**



$\text{dec}(x) := x \leftarrow x - 1$

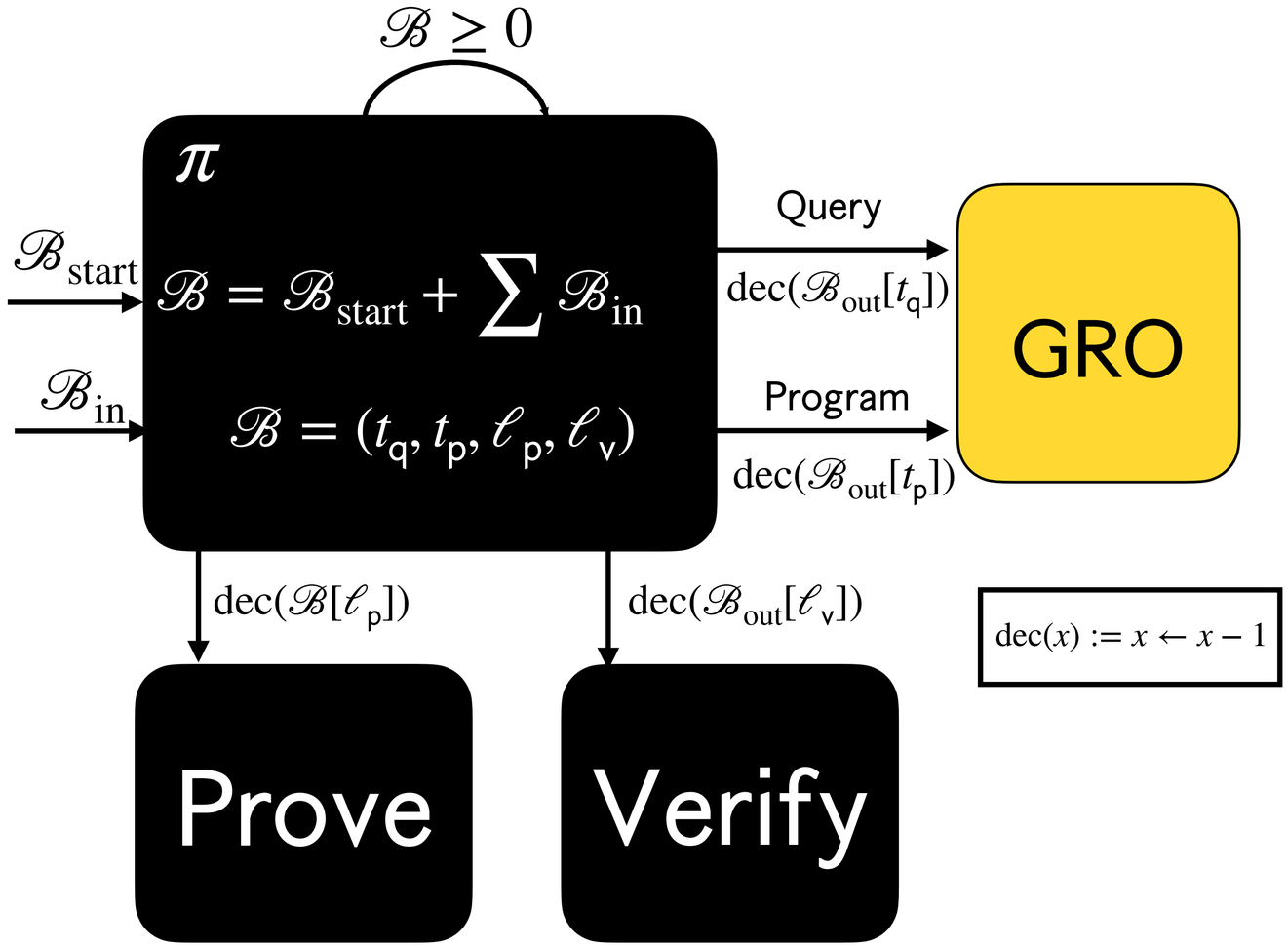
UC with Budgets

Plain UC only models adversaries that are **computationally** bounded



$$\text{time}(\pi) \leq p(\lambda_{\text{in}} - \lambda_{\text{out}})$$

We consider adversaries that are **resource** bounded and computationally **unbounded**



Recap:

What we talked about

Recap:

What we talked about

- UC with budgets

Recap:

What we talked about

- UC with budgets
- UC with Global Subroutines

Recap:

What we talked about

- UC with budgets
- UC with Global Subroutines
- UC-friendly security properties imply UC-security

Recap:

What we talked about

- UC with budgets
- UC with Global Subroutines
- UC-friendly security properties imply UC-security
- Micali has:

Recap:

What we talked about

- UC with budgets
- UC with Global Subroutines
- UC-friendly security properties imply UC-security
- Micali has:
  - UC-friendly completeness

Recap:

What we talked about

- UC with budgets
- UC with Global Subroutines
- UC-friendly security properties imply UC-security
- Micali has:
  - UC-friendly completeness
  - UC-friendly zero knowledge

Recap:

What we talked about

- UC with budgets
- UC with Global Subroutines
- UC-friendly security properties imply UC-security
- Micali has:
  - UC-friendly completeness
  - UC-friendly zero knowledge
  - UC-friendly knowledge soundness

There is more!

What we did not talk about

There is more!

What we did not talk about

- Concrete security bounds

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of BCS (leads to UC-security of **deployed** zkSNARKs)

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of BCS (leads to UC-security of **deployed** zkSNARKs)
- UC-friendly properties are **necessary**

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of BCS (leads to UC-security of **deployed** zkSNARKs)
- UC-friendly properties are **necessary**
- We can handle adaptive corruptions with **strong** UC-friendly properties

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of BCS (leads to UC-security of **deployed** zkSNARKs)
- UC-friendly properties are **necessary**
- We can handle adaptive corruptions with **strong** UC-friendly properties
- Merkle trees have (strong) UC-friendly hiding

There is more!

What we did not talk about

- Concrete security bounds
- UC-security of BCS (leads to UC-security of **deployed** zkSNARKs)
- UC-friendly properties are **necessary**
- We can handle adaptive corruptions with **strong** UC-friendly properties
- Merkle trees have (strong) UC-friendly hiding
- Merkle trees have (strong) UC-friendly extraction