Building Succinct Arguments From Ideal Hash Functions

Alessandro Chiesa

EPFL
StarkWare

Succinct Arguments

Succinct Arguments

Cryptographic proofs for computation integrity
that are super short and are super fast to verify.

Succinct Arguments

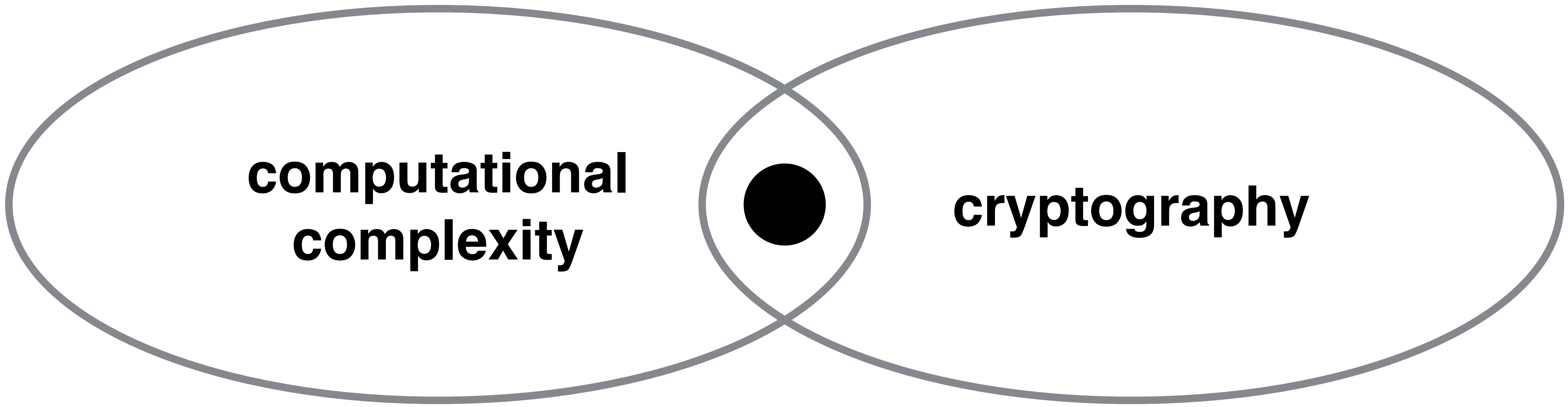
Cryptographic proofs for computation integrity
that are super short and are super fast to verify.

Origins in the 1990s:

Succinct Arguments

Cryptographic proofs for computation integrity
that are super short and are super fast to verify.

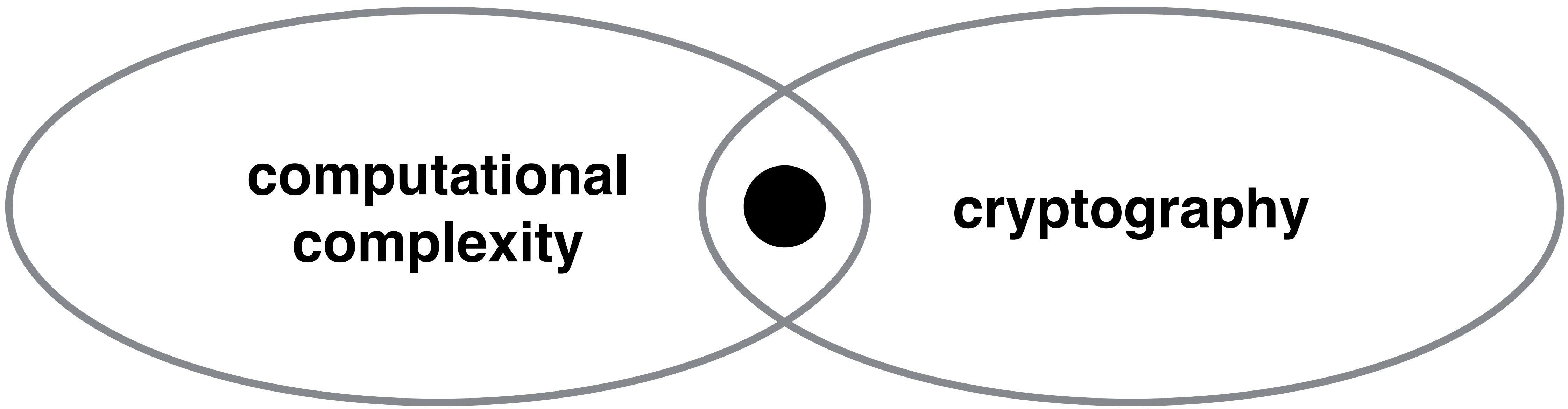
Origins in the 1990s:



Succinct Arguments

Cryptographic proofs for computation integrity
that are super short and are super fast to verify.

Origins in the 1990s:



In last 15 years, extraordinary progress in:

- cryptographic foundations
- efficient constructions
- implementations
- applications
- ...

Recipes

Recipes

Succinct arguments are constructed according to various **recipes**:

Recipes

Succinct arguments are constructed according to various **recipes**:

linear PCP + linear encoding	PCP/IOP + hash function	polynomial PCP/IOP + polynomial commitment	...

Recipes

Succinct arguments are constructed according to various **recipes**:

linear PCP + linear encoding	PCP/IOP + hash function	polynomial PCP/IOP + polynomial commitment	**...**

Recipes follow a template:

Recipes

Succinct arguments are constructed according to various **recipes**:

linear PCP + linear encoding	PCP/IOP + hash function	polynomial PCP/IOP + polynomial commitment	...

Recipes follow a template:

Probabilistic Proof + Cryptographic Commitment → Succinct Argument

Recipes

Succinct arguments are constructed according to various **recipes**:

linear PCP + linear encoding	PCP/IOP + hash function	polynomial PCP/IOP + polynomial commitment	...

Recipes follow a template:

Probabilistic Proof + Cryptographic Commitment → Succinct Argument

determines

↓

type of computation
(e.g., circuit vs machine)

Recipes

Succinct arguments are constructed according to various **recipes**:

linear PCP + linear encoding	PCP/IOP + hash function	polynomial PCP/IOP + polynomial commitment	...

Recipes follow a template:

Probabilistic Proof + Cryptographic Commitment → Succinct Argument

Probabilistic Proof determines ↓ type of computation (e.g., circuit vs machine)

Cryptographic Commitment determines ↓

- cryptographic costs (in prover and in verifier)
- pre-quantum vs. post-quantum
- setup (public or private, specific or universal)

Recipes

Succinct arguments are constructed according to various **recipes**:

linear PCP + linear encoding	PCP/IOP + hash function	polynomial PCP/IOP + polynomial commitment	…

Recipes follow a template:

Probabilistic Proof + Cryptographic Commitment → Succinct Argument

Probabilistic Proof determines ↓ type of computation (e.g., circuit vs machine)

Cryptographic Commitment determines ↓

- cryptographic costs (in prover and in verifier)
- pre-quantum vs. post-quantum
- setup (public or private, specific or universal)

TODAY: What is the security of succinct arguments?

Recipes

Succinct arguments are constructed according to various **recipes**:

linear PCP + linear encoding	PCP/IOP + hash function	polynomial PCP/IOP + polynomial commitment	...

Recipes follow a template:

Probabilistic Proof + Cryptographic Commitment → Succinct Argument

Probabilistic Proof determines ↓ type of computation (e.g., circuit vs machine)

Cryptographic Commitment determines ↓

- cryptographic costs (in prover and in verifier)
- pre-quantum vs. post-quantum
- setup (public or private, specific or universal)

TODAY: What is the security of succinct arguments?

Different Probabilistic Proofs

Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:

Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:

multi-prover
multiple provers that are isolated

interaction
prover and verifier exchange messages

randomness
verifier is probabilistic

oracle access
verifier may query prover's messages

Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:

IP (interactive proofs)

multi-prover
multiple provers that are isolated

interaction
prover and verifier exchange messages

randomness
verifier is probabilistic

oracle access
verifier may query prover's messages

Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:

PCP (probabilistically checkable proofs)

IP (interactive proofs)

multi-prover
multiple provers that are isolated

interaction
prover and verifier exchange messages

randomness
verifier is probabilistic

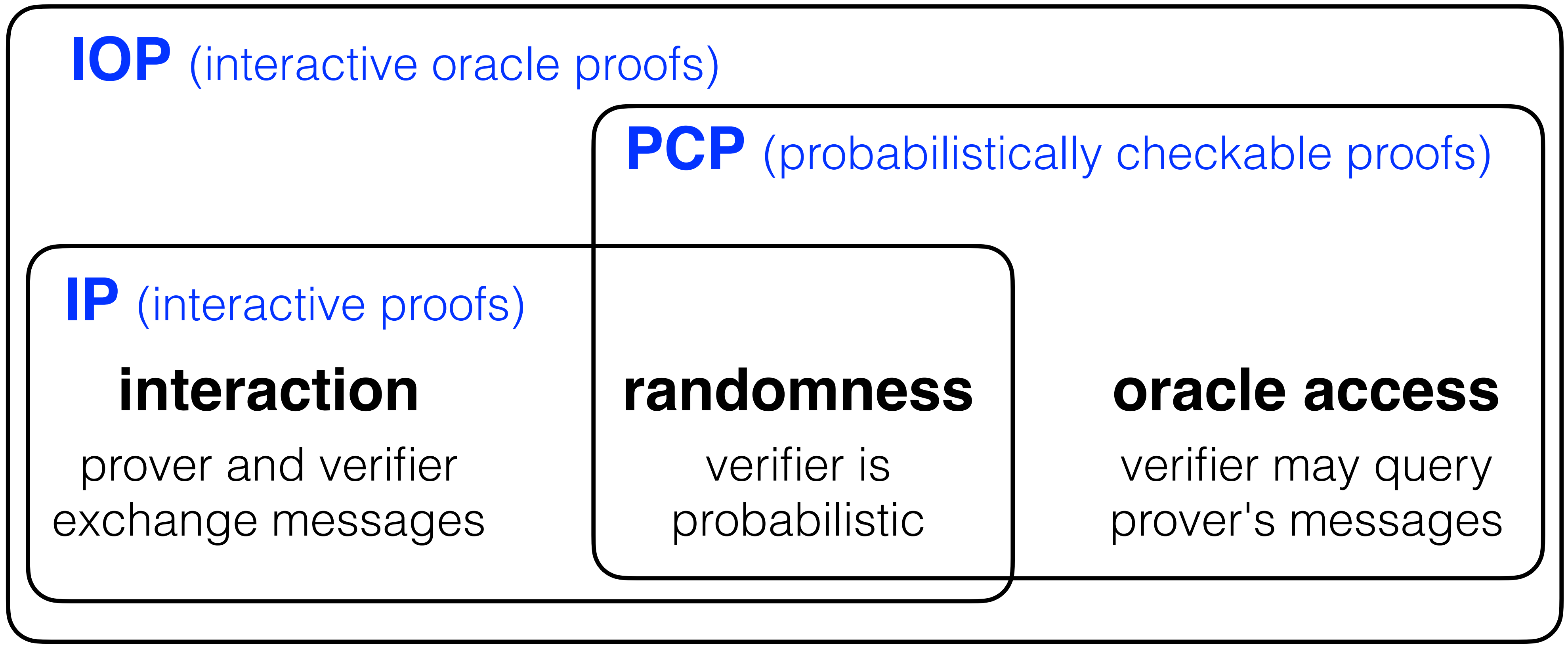
oracle access
verifier may query prover's messages

Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:

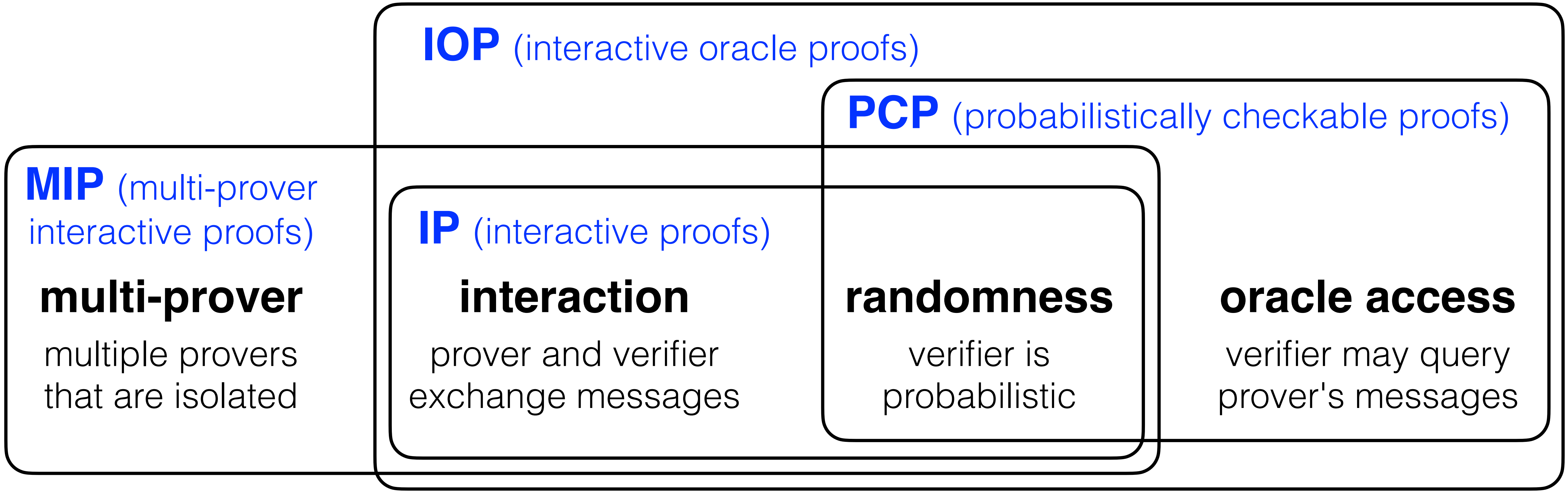
multi-prover

multiple provers
that are isolated



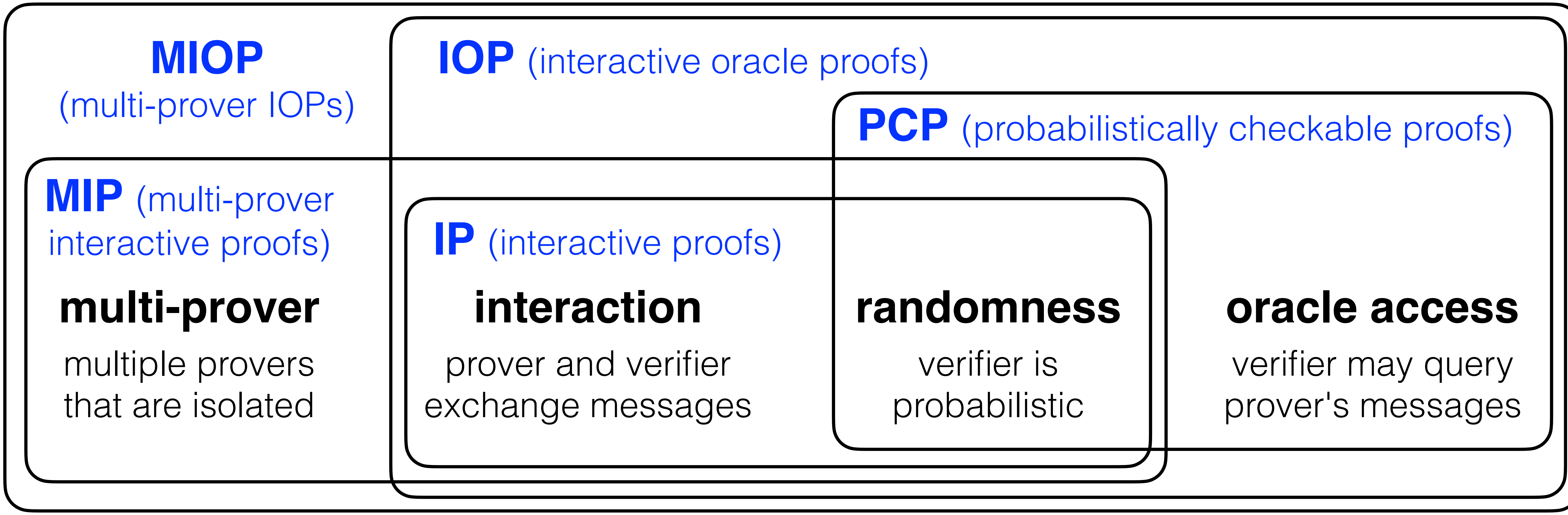
Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:



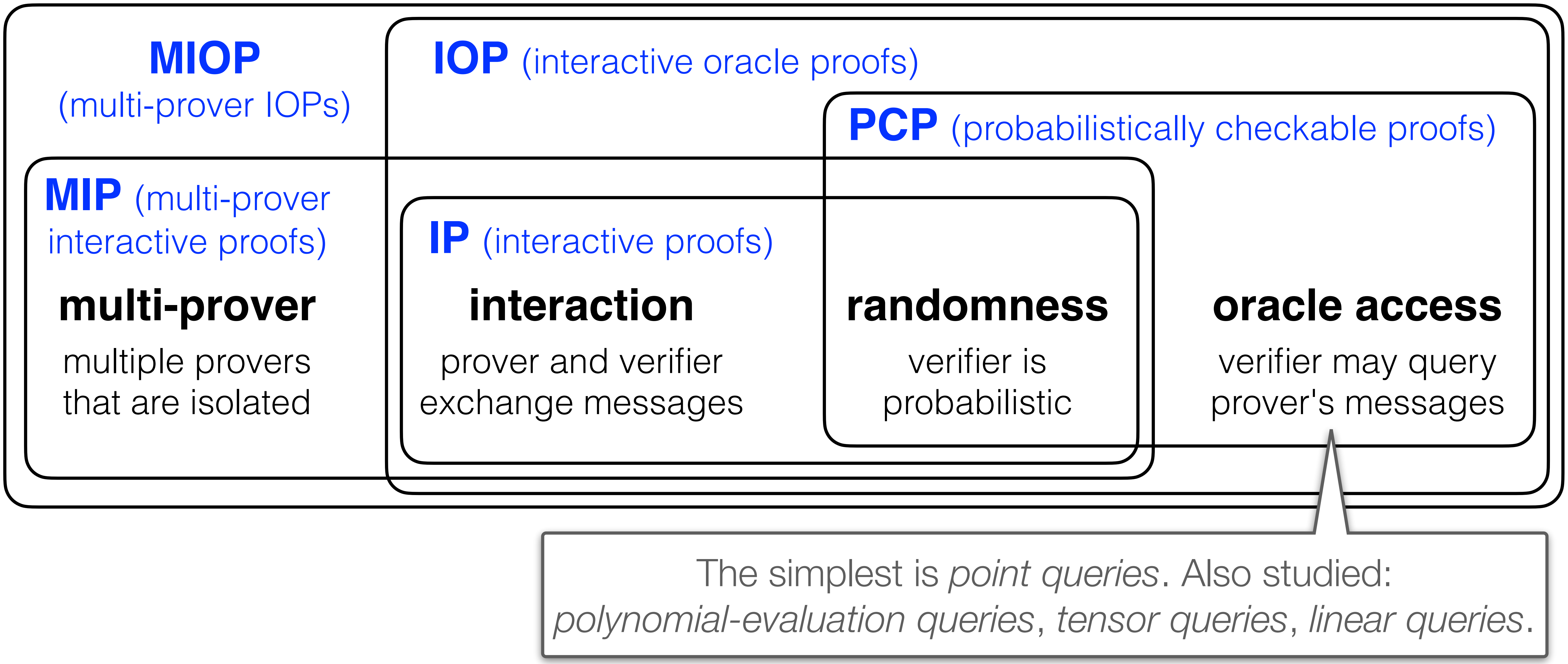
Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:



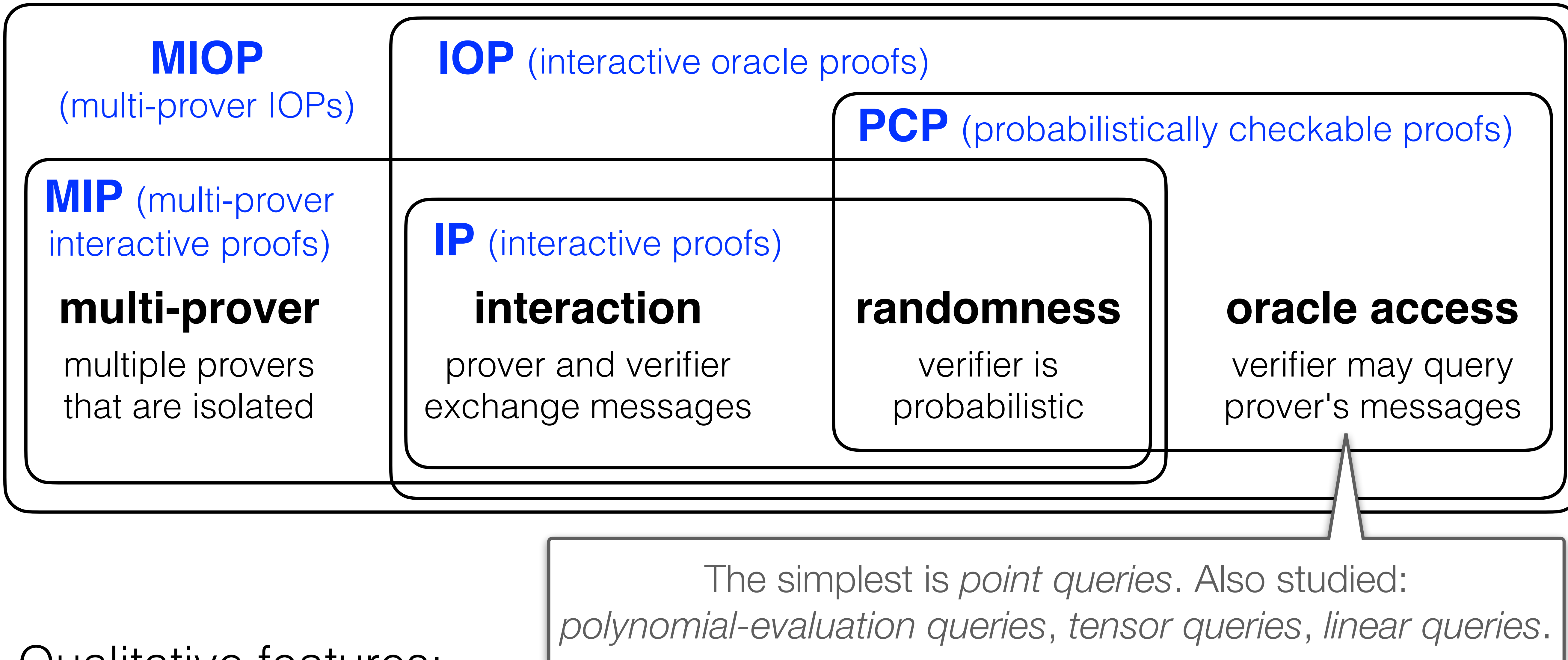
Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:



Different Probabilistic Proofs

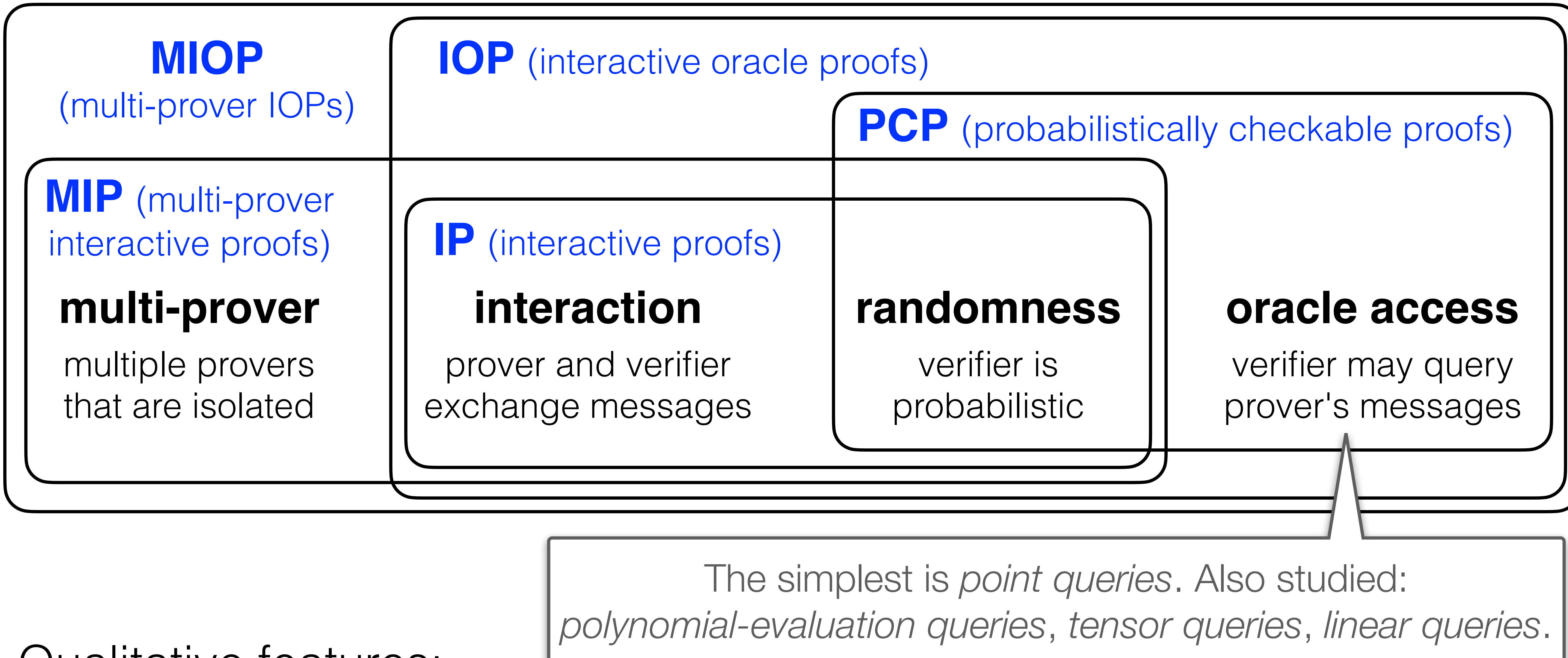
Depending on the "powers" granted to the verifier:



Qualitative features:

Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:

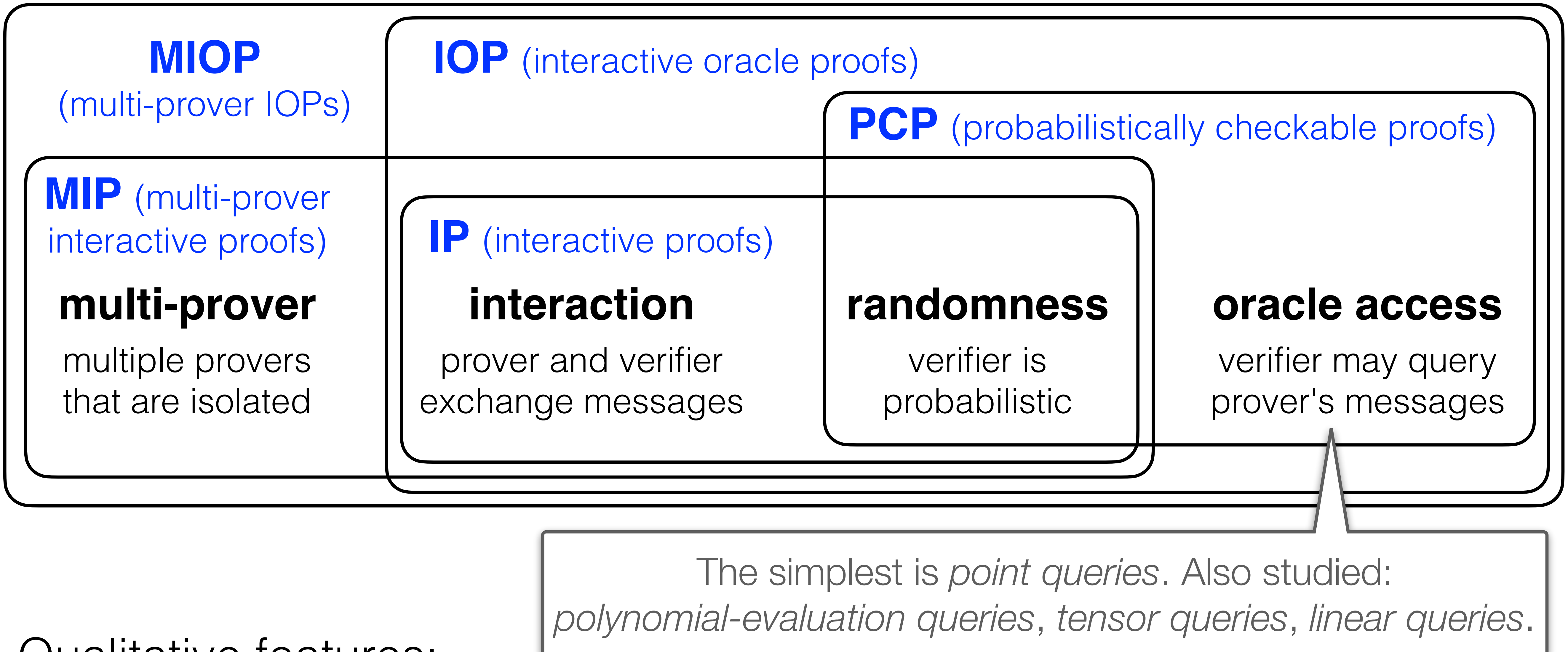


Qualitative features:

- **IP**: primarily sub-routines (e.g. sumcheck) to other probabilistic proofs

Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:

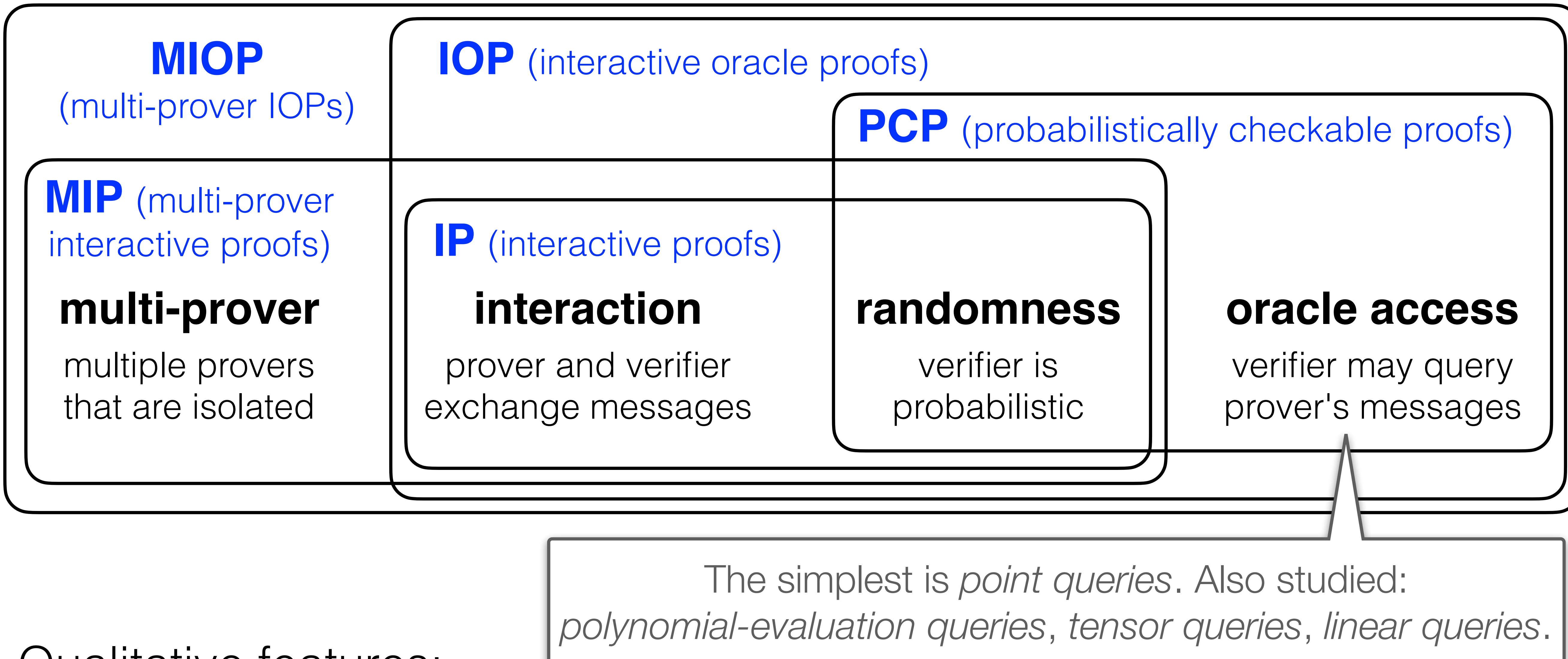


Qualitative features:

- **IP**: primarily sub-routines (e.g. sumcheck) to other probabilistic proofs
- **PCP**: pedagogically useful but inefficient (e.g. with point queries)

Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:

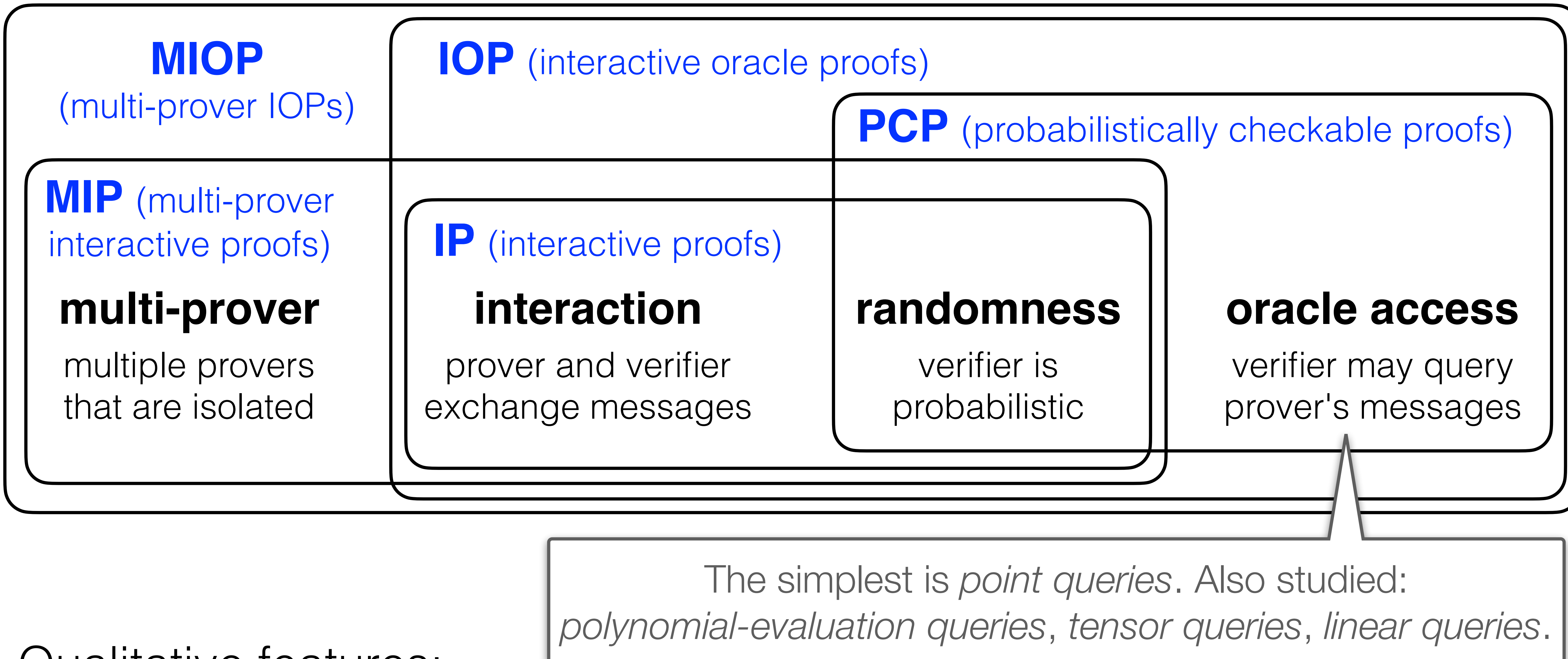


Qualitative features:

- **IP**: primarily sub-routines (e.g. sumcheck) to other probabilistic proofs
- **PCP**: pedagogically useful but inefficient (e.g. with point queries)
- **MIP** (& **MIOP**): attractive features (e.g. space efficiency) but hard to use

Different Probabilistic Proofs

Depending on the "powers" granted to the verifier:



Qualitative features:

- **IP**: primarily sub-routines (e.g. sumcheck) to other probabilistic proofs
- **PCP**: pedagogically useful but inefficient (e.g. with point queries)
- **MIP** (& **MIOP**): attractive features (e.g. space efficiency) but hard to use
- **IOP**: underlie most efficient succinct arguments

(Ideal) Hash Functions

(Ideal) Hash Functions

A **cryptographic hash function** with output size λ is an algorithm that maps an (arbitrary-length) input to a λ-bit output.

(Ideal) Hash Functions

A **cryptographic hash function** with output size λ is an algorithm that maps an (arbitrary-length) input to a λ-bit output.

Hash functions are designed to have no "useful structure".
They are often modeled as **random functions** for analysis.

(Ideal) Hash Functions

A **cryptographic hash function** with output size λ is an algorithm that maps an (arbitrary-length) input to a λ-bit output.

Hash functions are designed to have no "useful structure".
They are often modeled as **random functions** for analysis.

Random Oracle Model (ROM): every party (honest or malicious) has query access to a randomly sampled function $f: \{0,1\}^* \to \{0,1\}^\lambda$.

(Ideal) Hash Functions

A **cryptographic hash function** with output size λ is an algorithm that maps an (arbitrary-length) input to a λ-bit output.

Hash functions are designed to have no "useful structure".
They are often modeled as **random functions** for analysis.

Random Oracle Model (ROM): every party (honest or malicious) has query access to a randomly sampled function $f: \{0,1\}^* \to \{0,1\}^\lambda$.

Template security theorem for a cryptographic scheme S in the ROM:

$\forall$ t-query adversary $\mathcal{A}$

Pr[$\mathcal{A}$ breaks scheme S] $\leq \varepsilon(\lambda,t)$

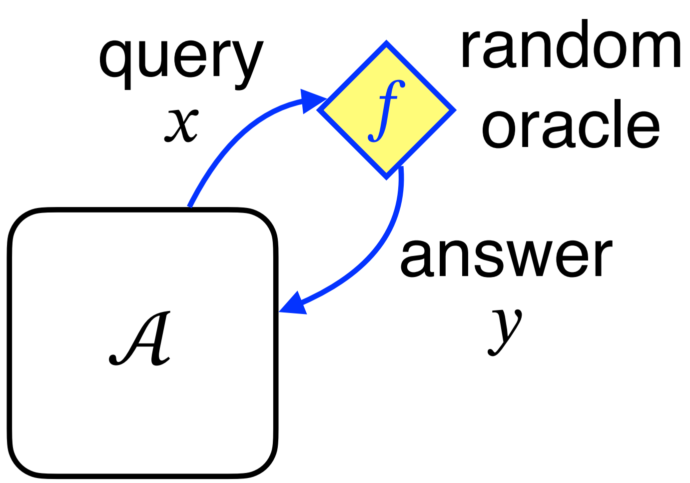
(Ideal) Hash Functions

A **cryptographic hash function** with output size λ is an algorithm that maps an (arbitrary-length) input to a λ-bit output.

Hash functions are designed to have no "useful structure".
They are often modeled as **random functions** for analysis.

Random Oracle Model (ROM): every party (honest or malicious) has query access to a randomly sampled function $f: \{0,1\}^* \to \{0,1\}^\lambda$.

Template security theorem for a cryptographic scheme S in the ROM:
$\forall$ t-query adversary $\mathcal{A}$
Pr[$\mathcal{A}$ breaks scheme S] $\leq \varepsilon(\lambda,t)$



Warm-Up:
Succinct Arguments from PCPs

Probabilistically Checkable Proofs

Probabilistically Checkable Proofs

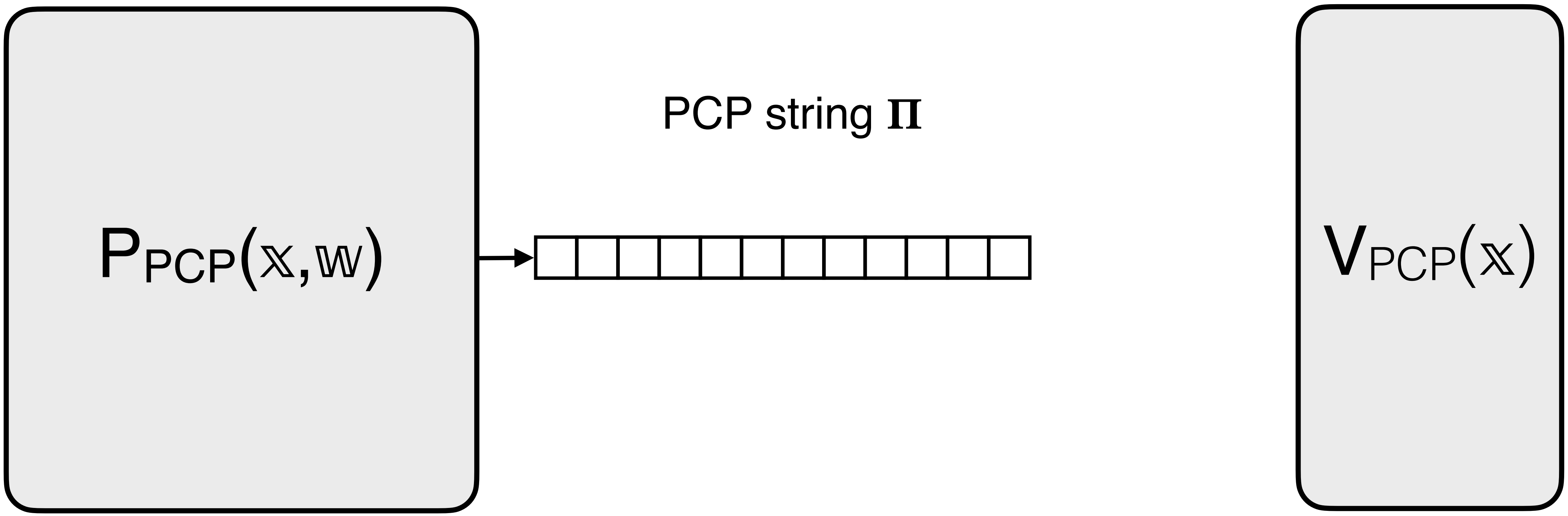
A **PCP** is a model of proof system where the verifier
is probabilistic and has oracle access to 1 prover message.

Probabilistically Checkable Proofs

A **PCP** is a model of proof system where the verifier
is probabilistic and has oracle access to 1 prover message.

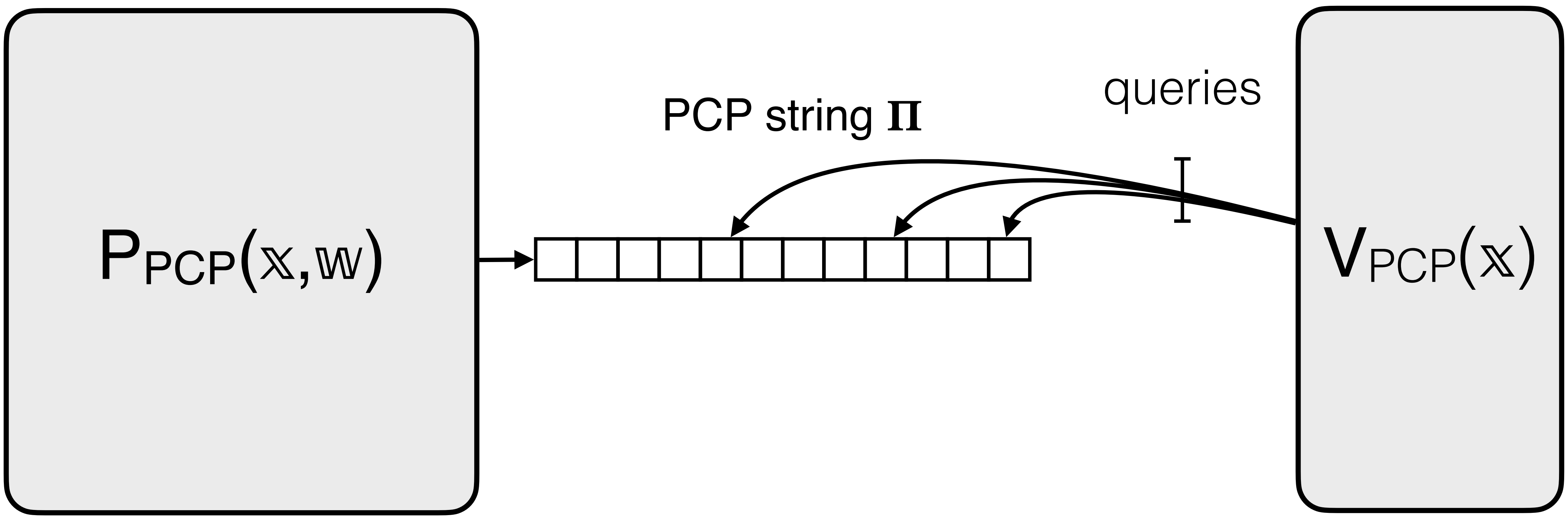
Probabilistically Checkable Proofs

A **PCP** is a model of proof system where the verifier
is probabilistic and has oracle access to 1 prover message.



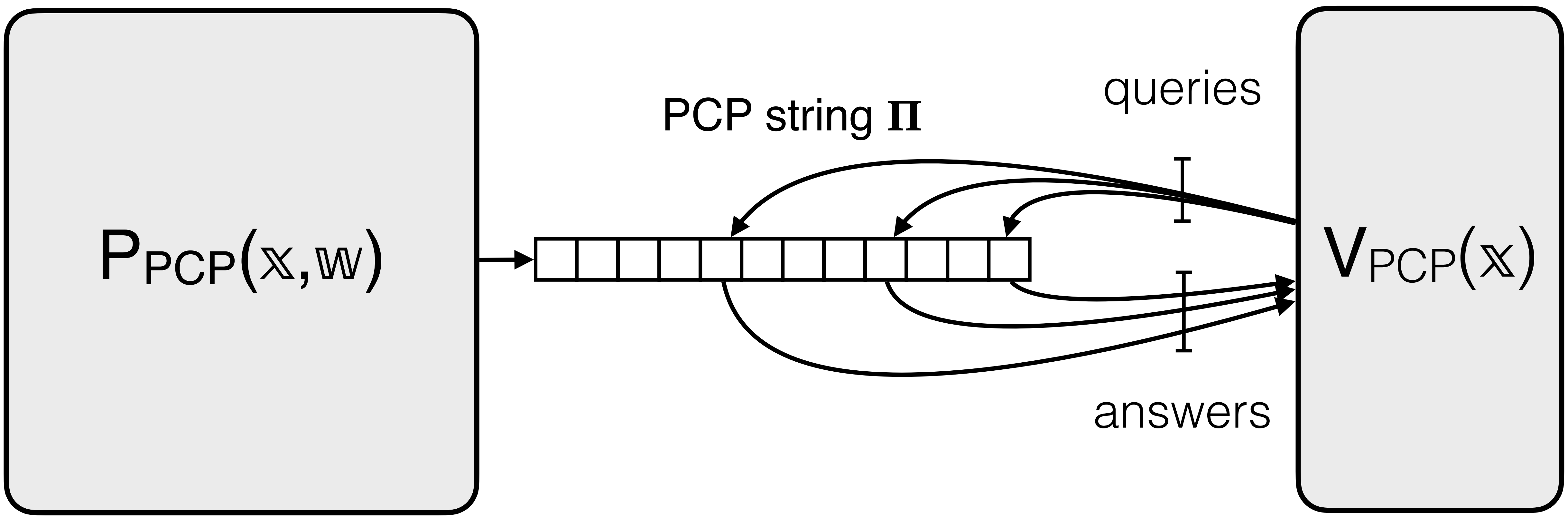
Probabilistically Checkable Proofs

A **PCP** is a model of proof system where the verifier
is probabilistic and has oracle access to 1 prover message.



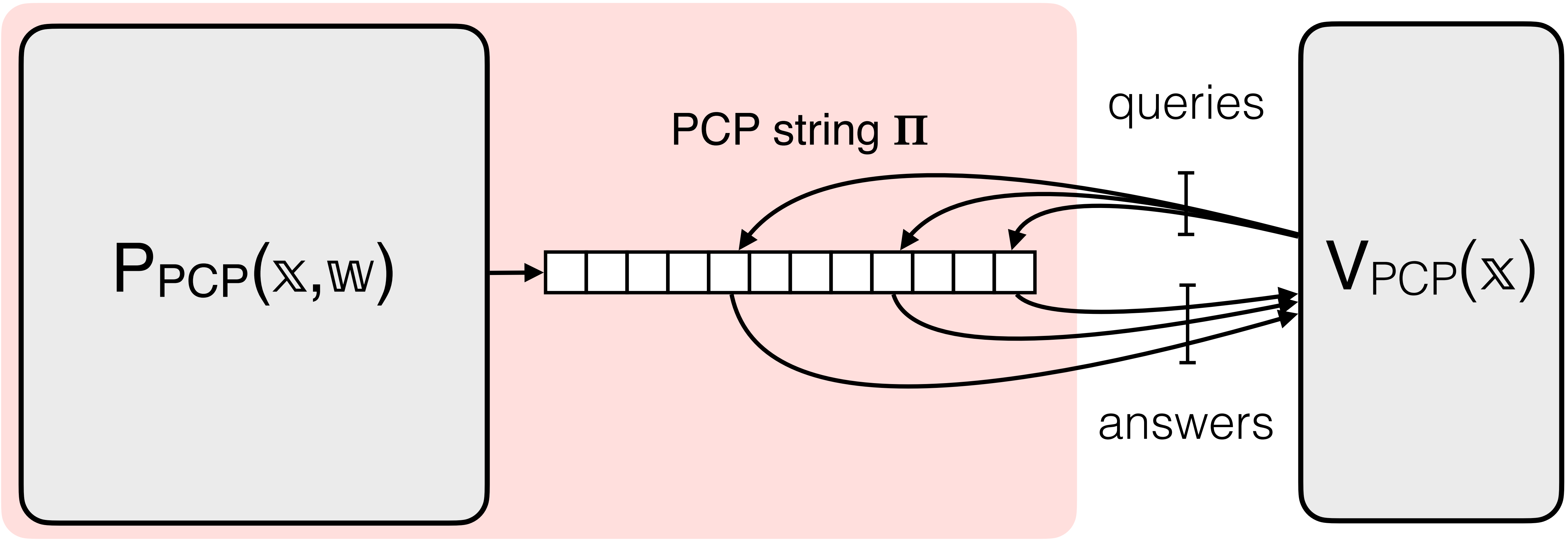
Probabilistically Checkable Proofs

A **PCP** is a model of proof system where the verifier
is probabilistic and has oracle access to 1 prover message.



Probabilistically Checkable Proofs

A **PCP** is a model of proof system where the verifier
is probabilistic and has oracle access to 1 prover message.



Note: PCP ≠ Succinct Argument!
It is insecure for the verifier to ask the prover to answer queries.

Kilian Protocol

Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.

Kilian Protocol

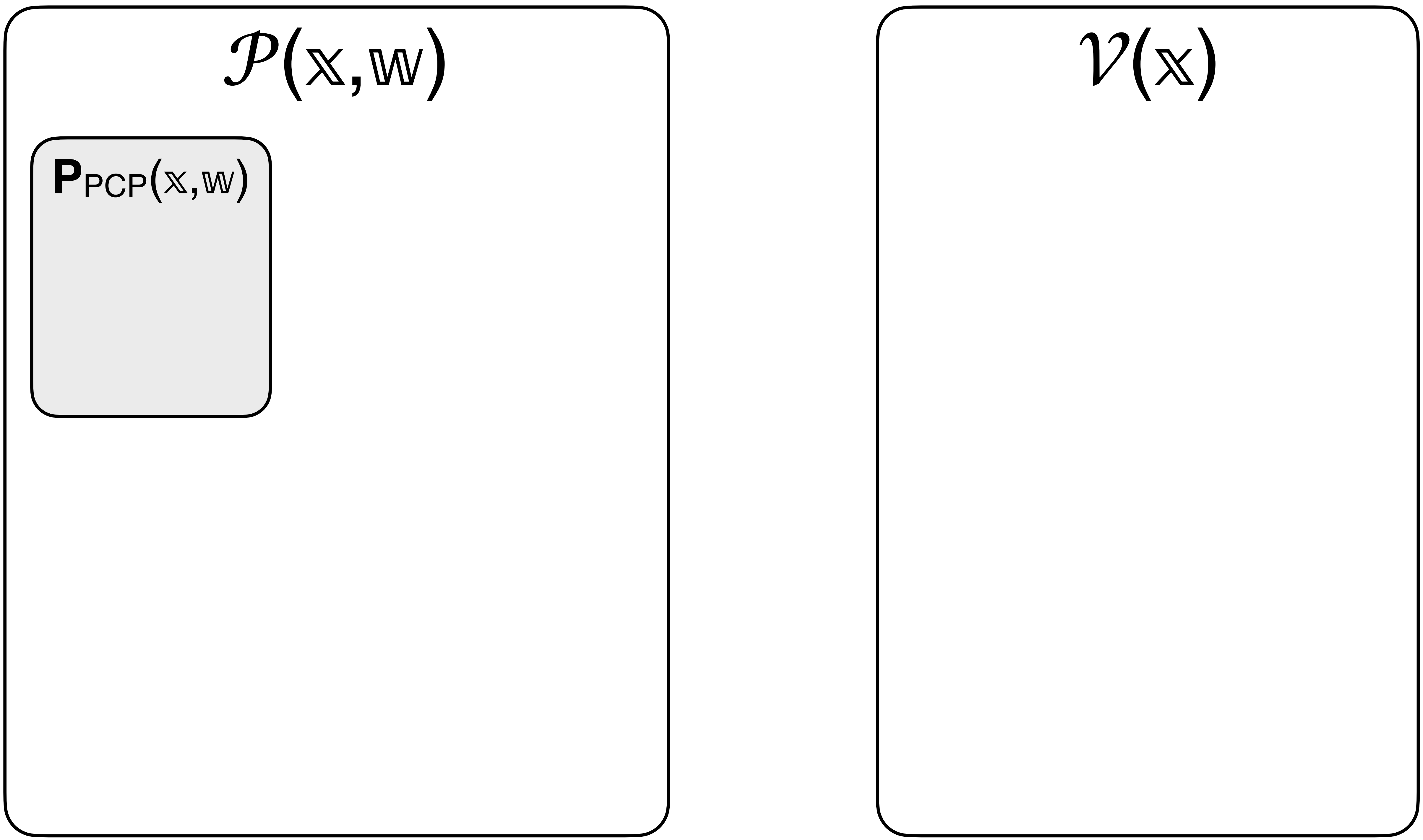
Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.

$\mathcal{P}(\mathbb{x},\mathbb{w})$

$\mathcal{V}(\mathbb{x})$

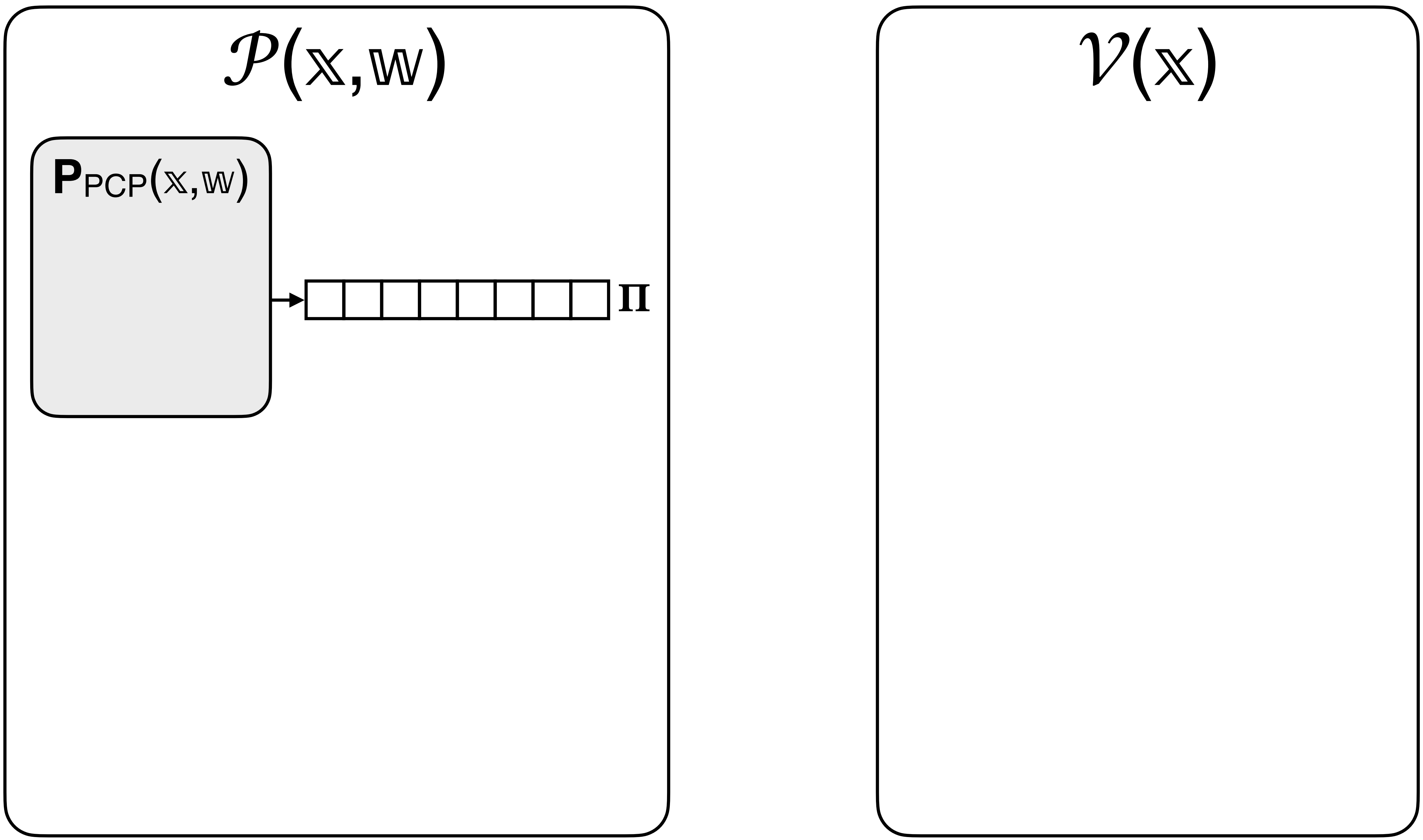
Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



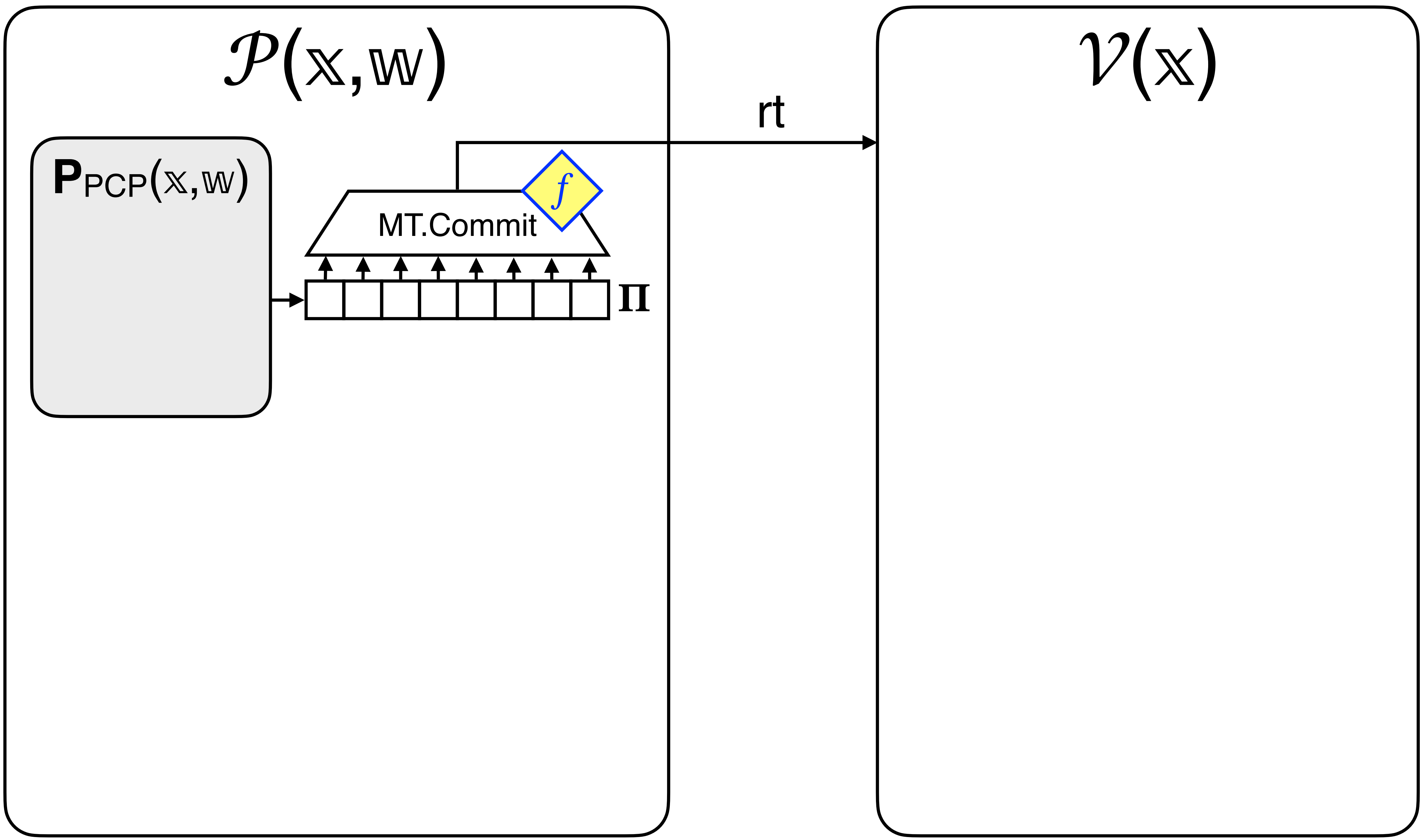
Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



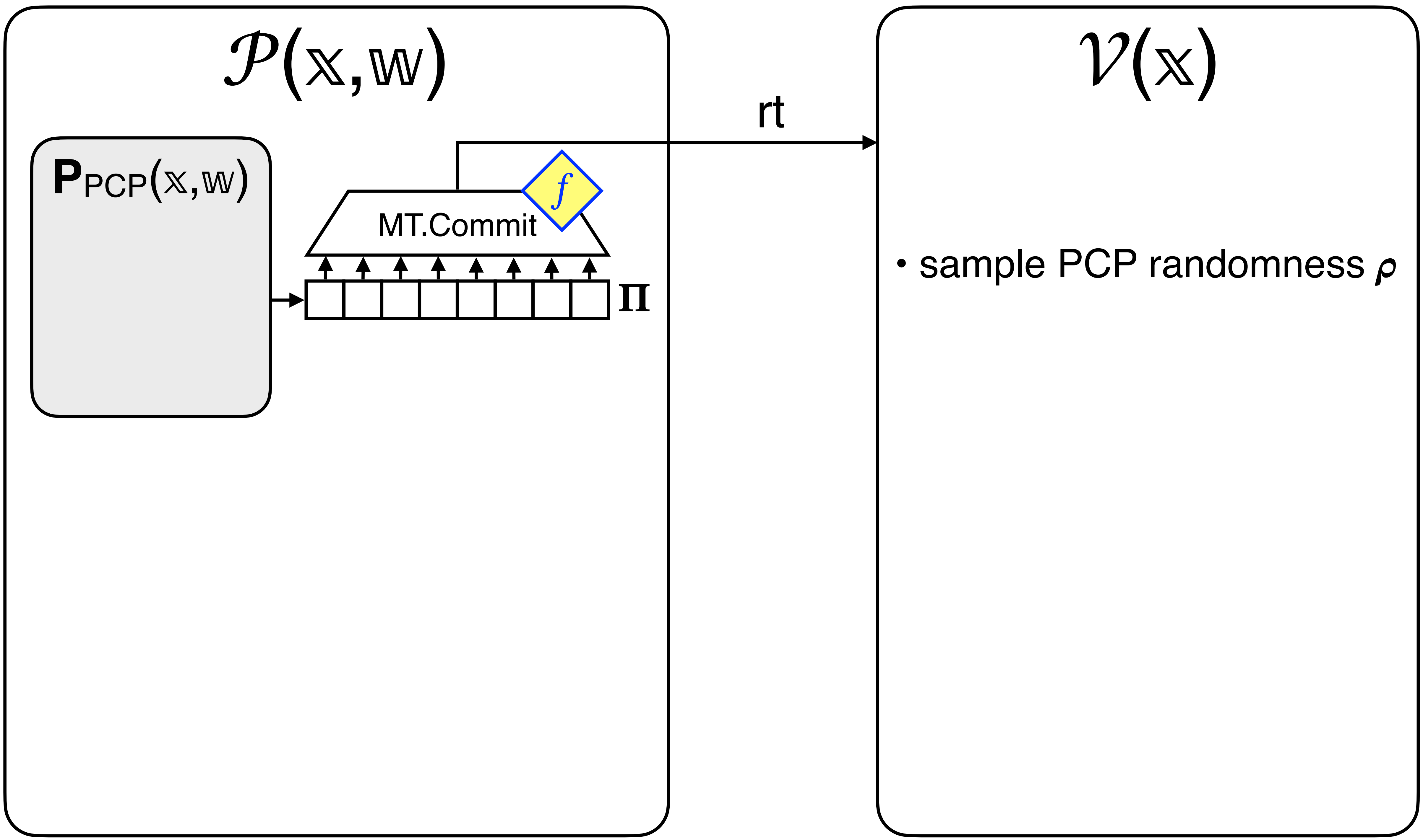
Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



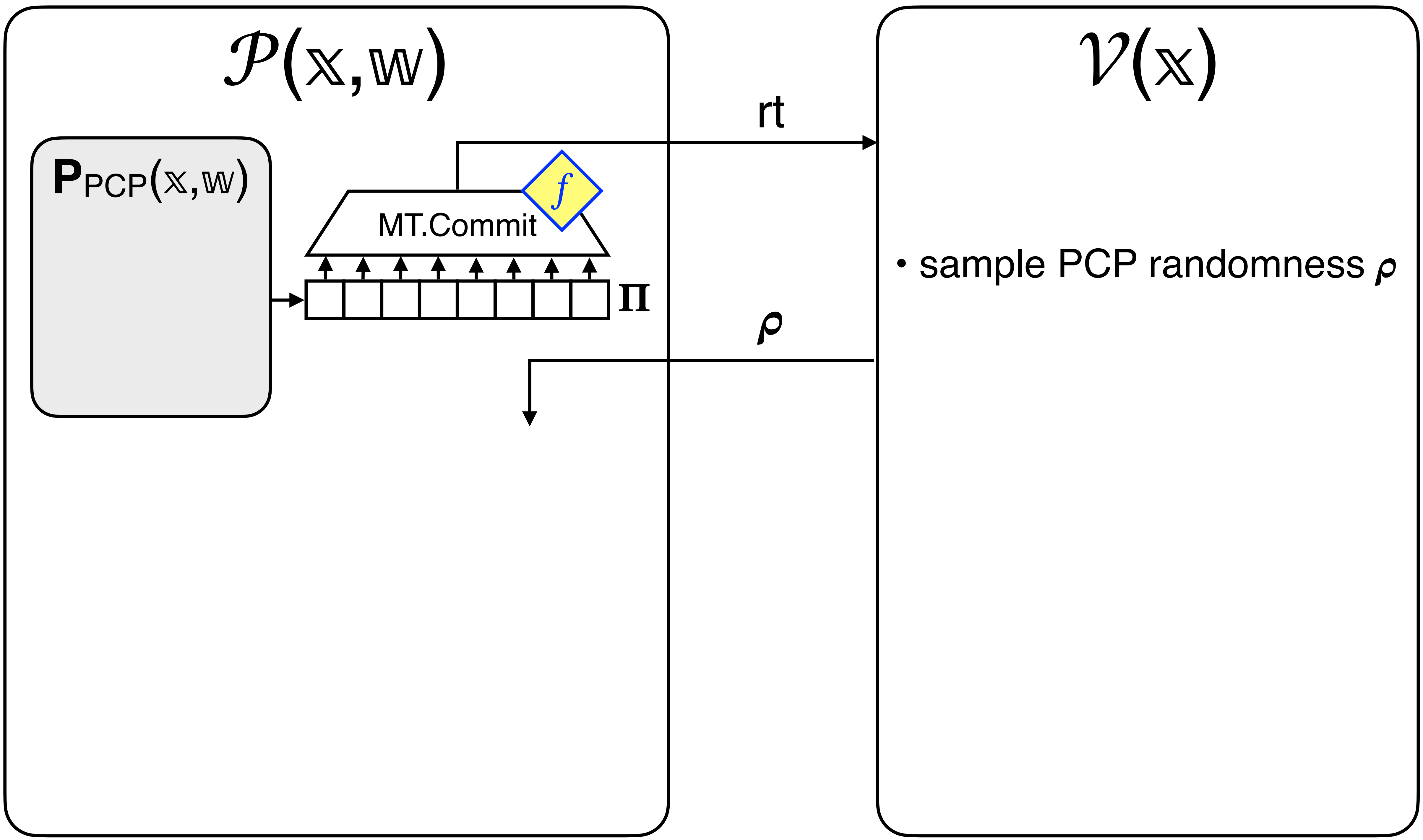
Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



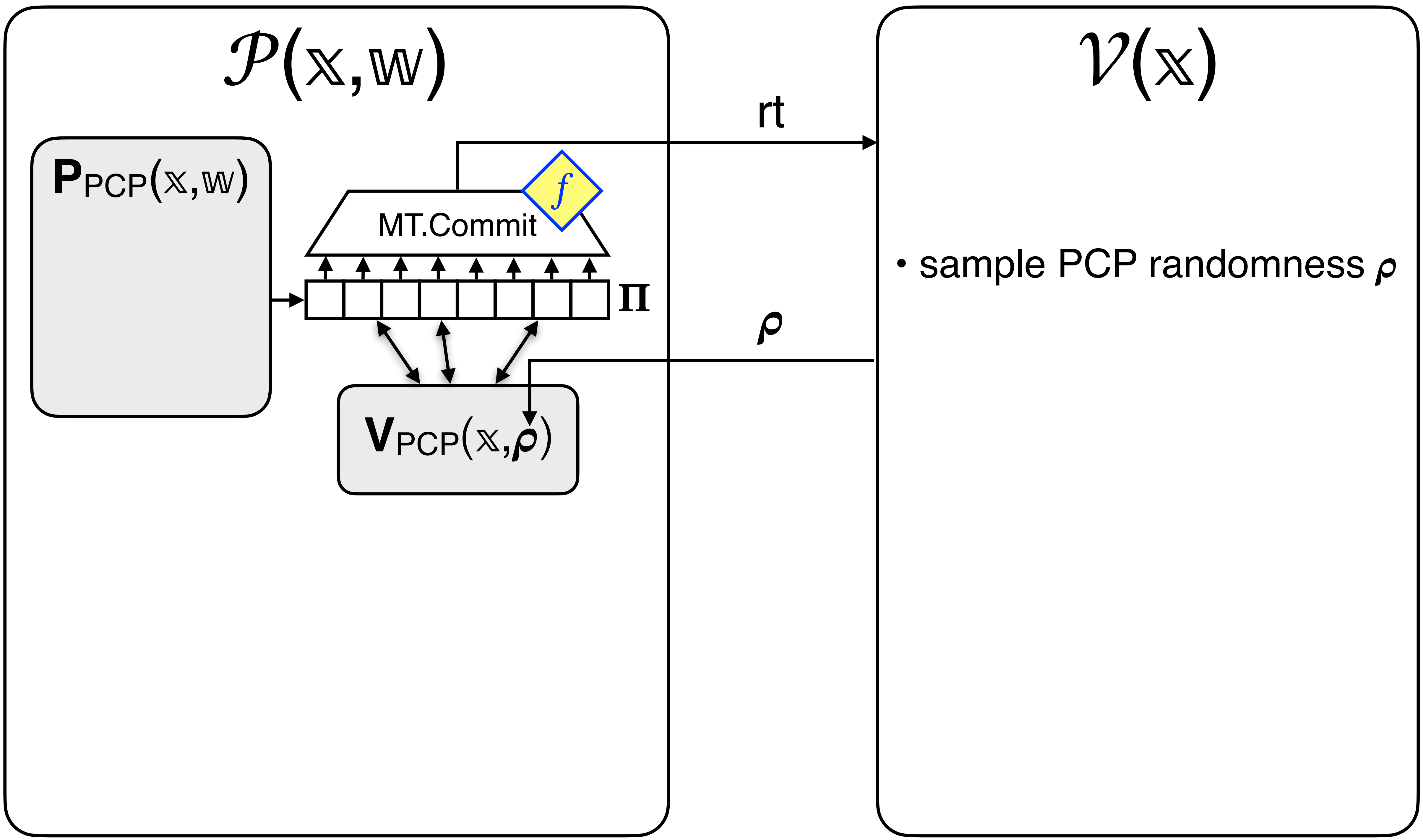
Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



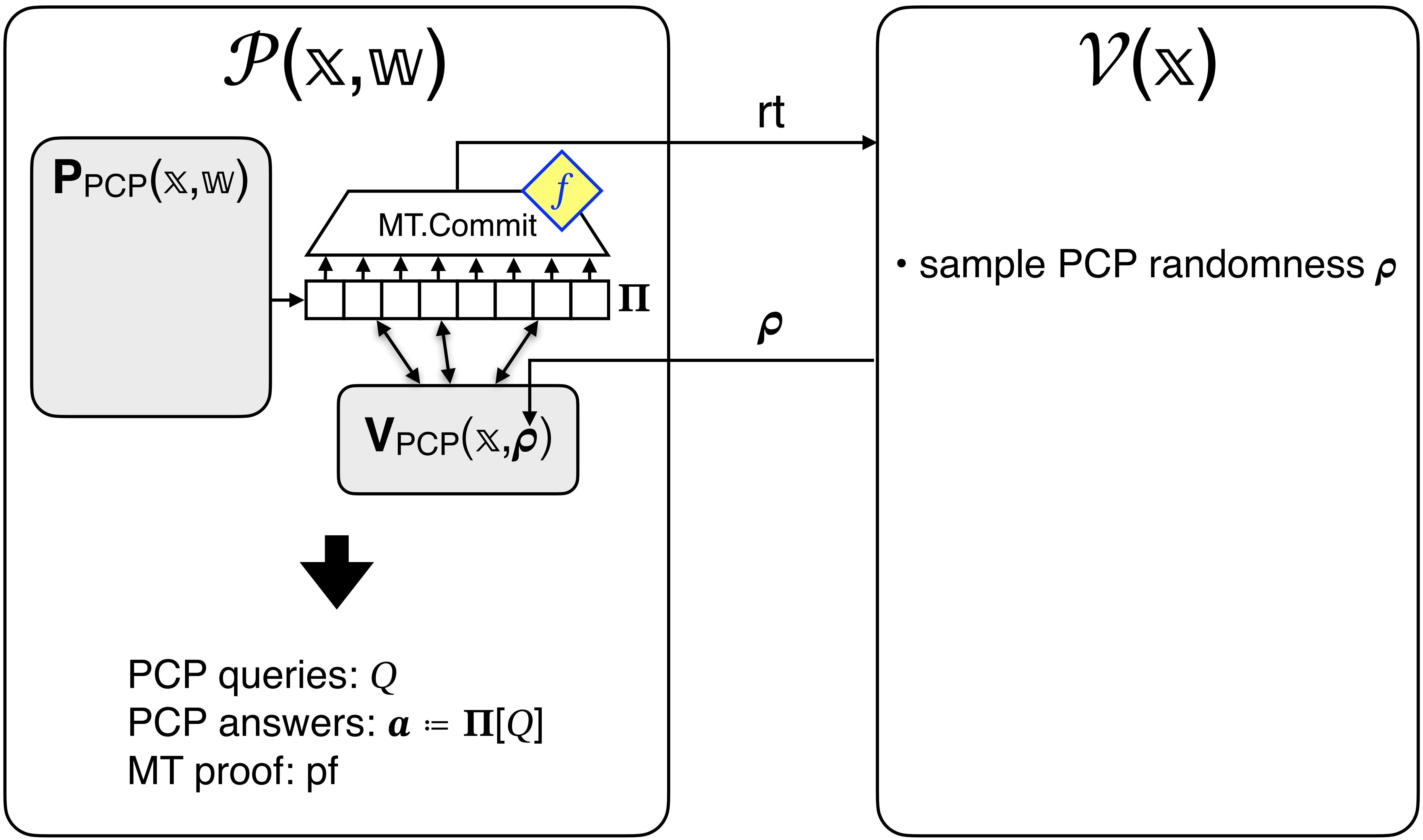
Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



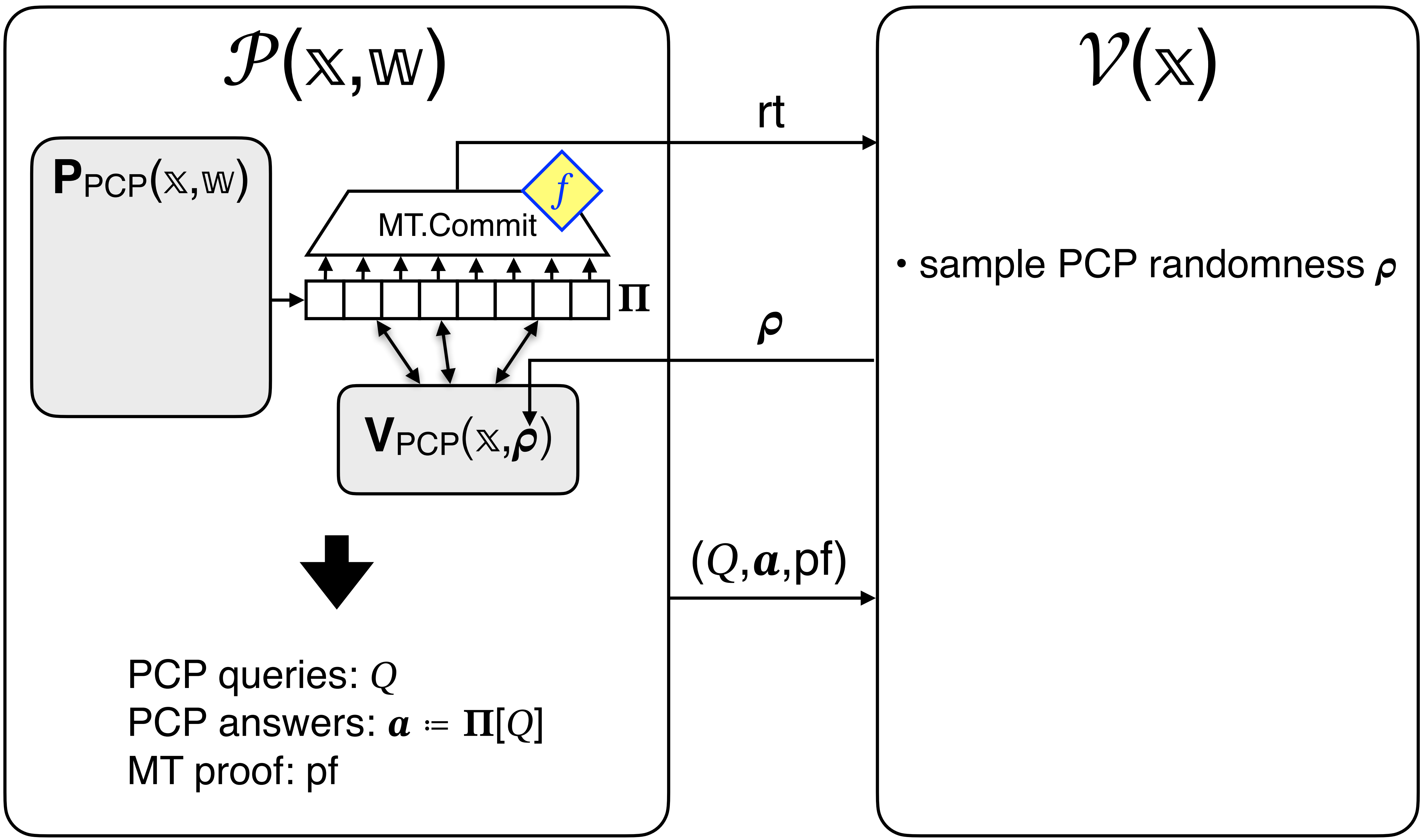
Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



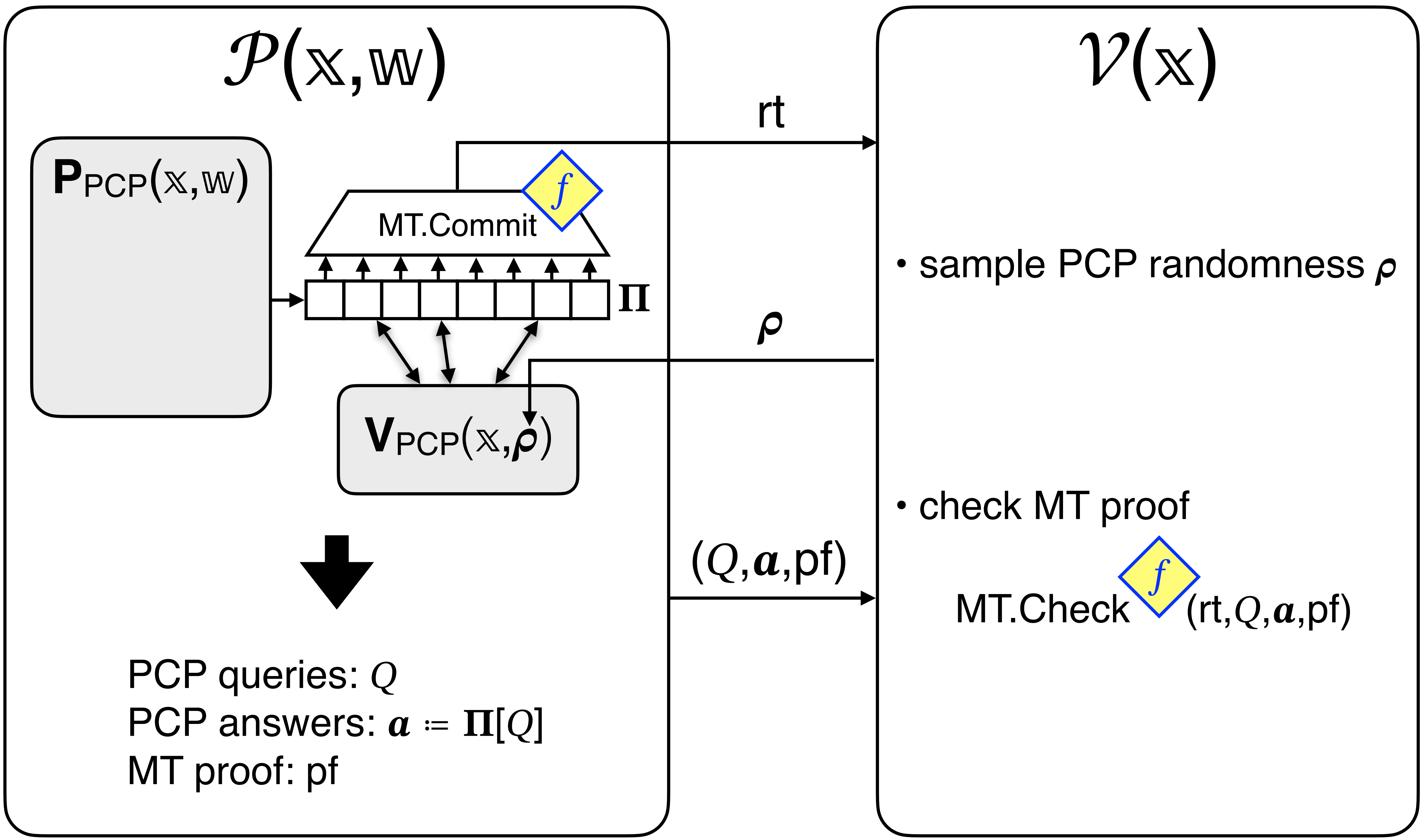
Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



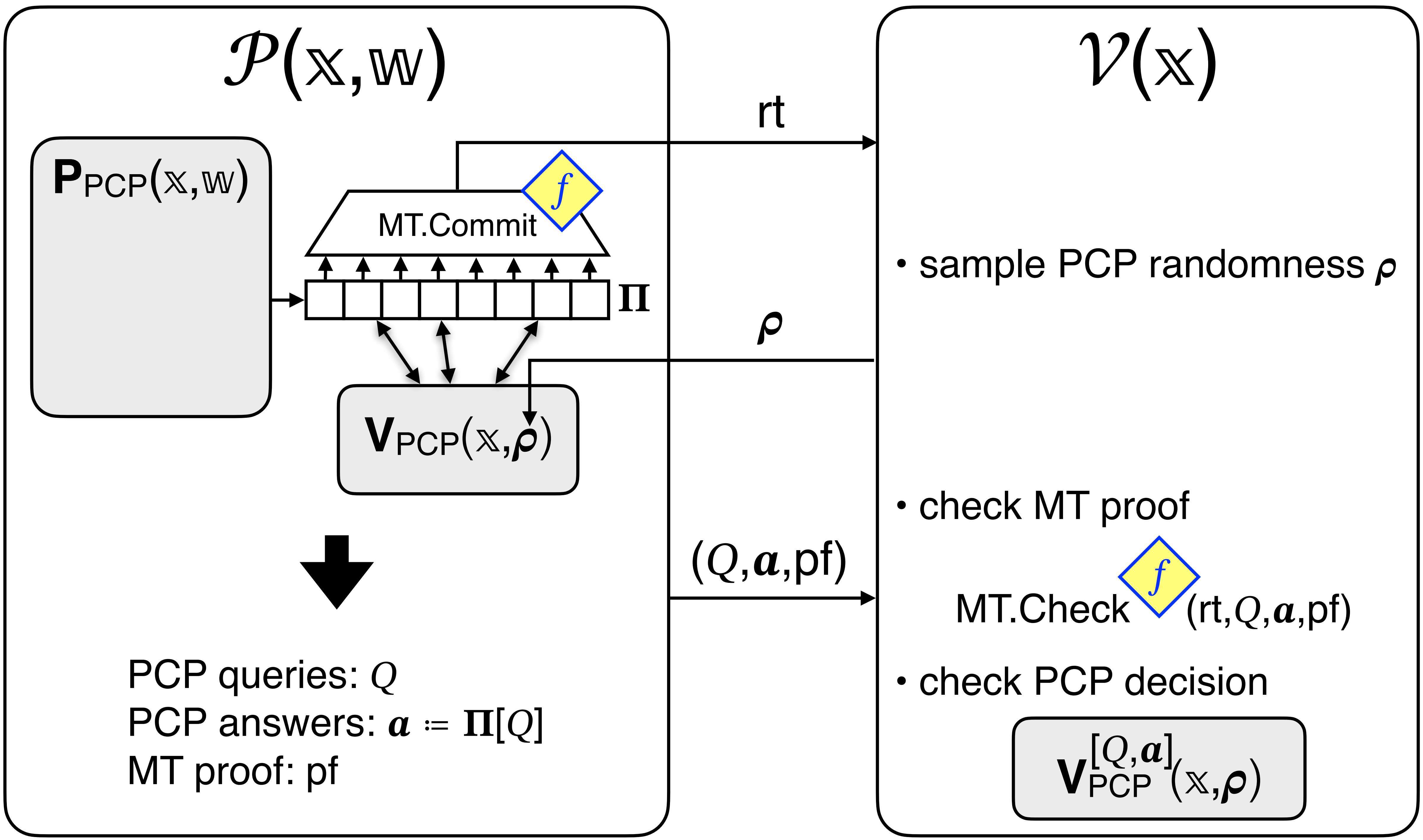
Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



Kilian Protocol

Idea: Use a Merkle Tree to commit to the PCP string.
Then reveal the queried locations of the PCP string.



Kilian Protocol

Kilian Protocol

PCP parameters:
- ε_{PCP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Kilian Protocol

PCP parameters:
- ε_{PCP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

Kilian Protocol

PCP parameters:
- $\varepsilon_{\mathrm{PCP}}$ is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathbf{P}_{\mathrm{PCP}}) + \ell \cdot \mathrm{time}(f)$$

Kilian Protocol

PCP parameters:
- ε_{PCP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\text{time}(\mathcal{P}) = \text{time}(\mathbf{P}_{\text{PCP}}) + \ell \cdot \text{time}(f)$$

$$\text{time}(\mathcal{V}) = \text{time}(\mathbf{V}_{\text{PCP}}) + q \cdot \log\ell \cdot \text{time}(f)$$

Kilian Protocol

PCP parameters:
- ε_{PCP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\text{time}(\mathcal{P}) = \text{time}(\mathbf{P}_{\text{PCP}}) + \boxed{\ell \cdot \text{time}(f)}$$
$$\text{time}(\mathcal{V}) = \text{time}(\mathbf{V}_{\text{PCP}}) + \boxed{q \cdot \log \ell \cdot \text{time}(f)}$$

small overheads

Kilian Protocol

PCP parameters:
- $\varepsilon_{\mathrm{PCP}}$ is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathbf{P}_{\mathrm{PCP}}) + \ell \cdot \mathrm{time}(f)$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathbf{V}_{\mathrm{PCP}}) + q \cdot \log \ell \cdot \mathrm{time}(f)$$

small overheads

Communication ≈ query complexity (rather than proof length):

Kilian Protocol

PCP parameters:
- $\varepsilon_{\mathrm{PCP}}$ is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathrm{P}_{\mathrm{PCP}}) + \ell \cdot \mathrm{time}(f)$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathrm{V}_{\mathrm{PCP}}) + q \cdot \log\ell \cdot \mathrm{time}(f)$$

small overheads

Communication ≈ query complexity (rather than proof length):

$$|\mathrm{rt}| + |(Q,\boldsymbol{a},\mathrm{pf})| \approx q \cdot (\log|\Sigma| + 2\lambda \log\ell)$$

Kilian Protocol

PCP parameters:
- ε_{PCP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathrm{P_{PCP}}) + \boxed{\ell \cdot \mathrm{time}(f)}$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathrm{V_{PCP}}) + \boxed{q \cdot \log\ell \cdot \mathrm{time}(f)}$$

small overheads

Communication ≈ query complexity (rather than proof length):

$$|\mathrm{rt}| + |(Q,\boldsymbol{a},\mathrm{pf})| \approx q \cdot (\log|\Sigma| + 2\lambda \log\ell)$$

Security:

Kilian Protocol

PCP parameters:
- ε_{PCP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathbf{P}_{PCP}) + \ell \cdot \mathrm{time}(f)$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathbf{V}_{PCP}) + q \cdot \log\ell \cdot \mathrm{time}(f)$$

small overheads

Communication ≈ query complexity (rather than proof length):

$$|\mathrm{rt}| + |(Q,\boldsymbol{a},\mathrm{pf})| \approx q \cdot (\log|\Sigma| + 2\lambda \log\ell)$$

Security:

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks Kilian}] \leq \epsilon_{\mathrm{PCP}} + \frac{t^2}{2^\lambda}$$

Kilian Protocol

PCP parameters:
- ε_{PCP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathrm{P}_{\mathrm{PCP}}) + \ell \cdot \mathrm{time}(f)$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathrm{V}_{\mathrm{PCP}}) + q \cdot \log \ell \cdot \mathrm{time}(f)$$

small overheads

Communication ≈ query complexity (rather than proof length):

$$|\mathrm{rt}| + |(Q, \boldsymbol{a}, \mathrm{pf})| \approx q \cdot (\log|\Sigma| + 2\lambda \log \ell)$$

Security:

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks Kilian}] \leq \epsilon_{\mathrm{PCP}} + \frac{t^2}{2^\lambda}$$

Example:
Set ε_{PCP} =1/4 and λ=256+2=258.

Every 2^{128}-query adversary breaks Kilian w.p. ≤1/4+1/4=1/2.

Micali Protocol

Micali Protocol

Idea: Use the random oracle to derive the PCP randomness.

(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)

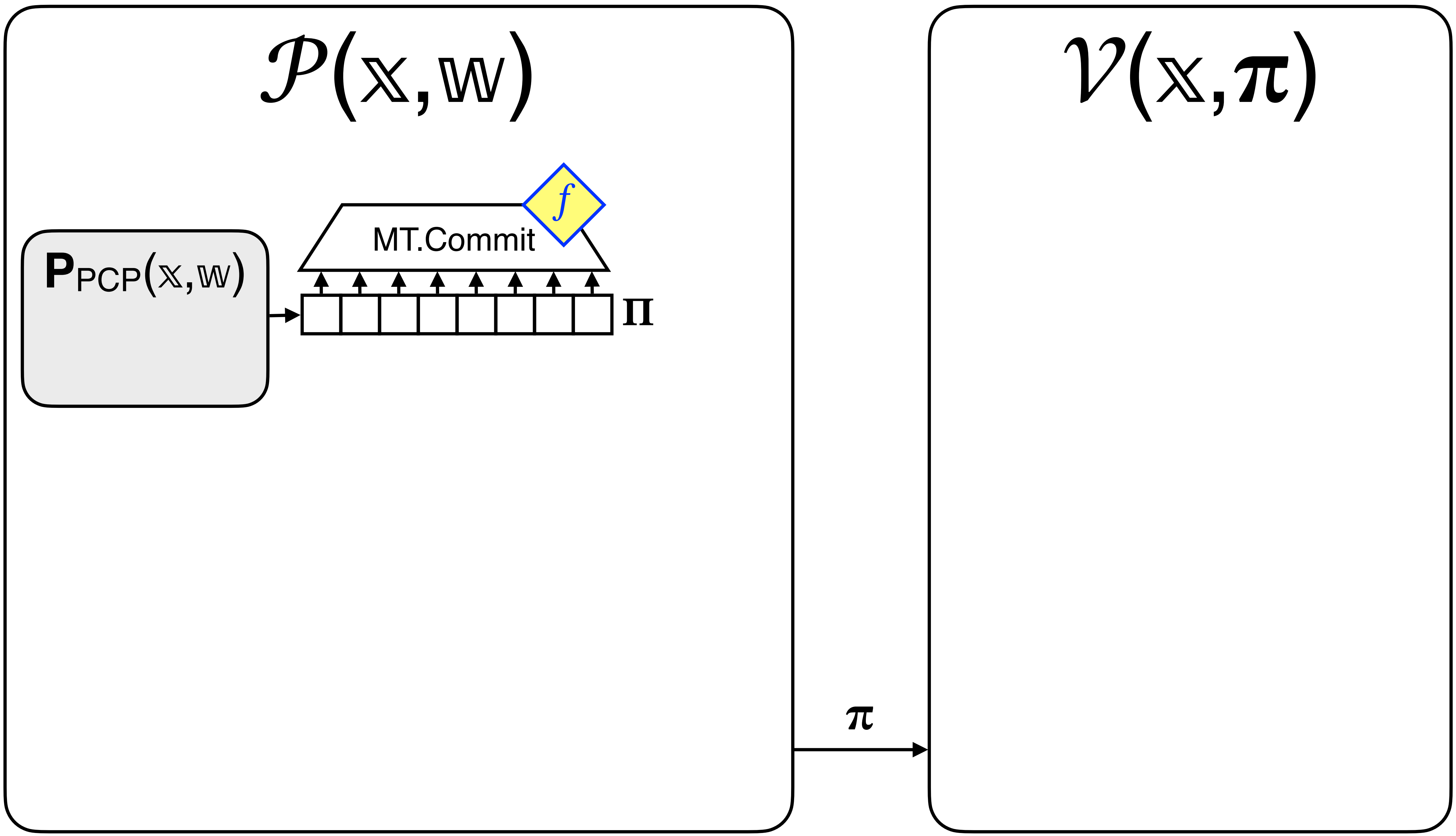
Micali Protocol

Idea: Use the random oracle to derive the PCP randomness.
(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)

$\mathcal{P}(\mathbb{x},\mathbb{w})$ $\xrightarrow{\boldsymbol{\pi}}$ $\mathcal{V}(\mathbb{x},\boldsymbol{\pi})$

Micali Protocol

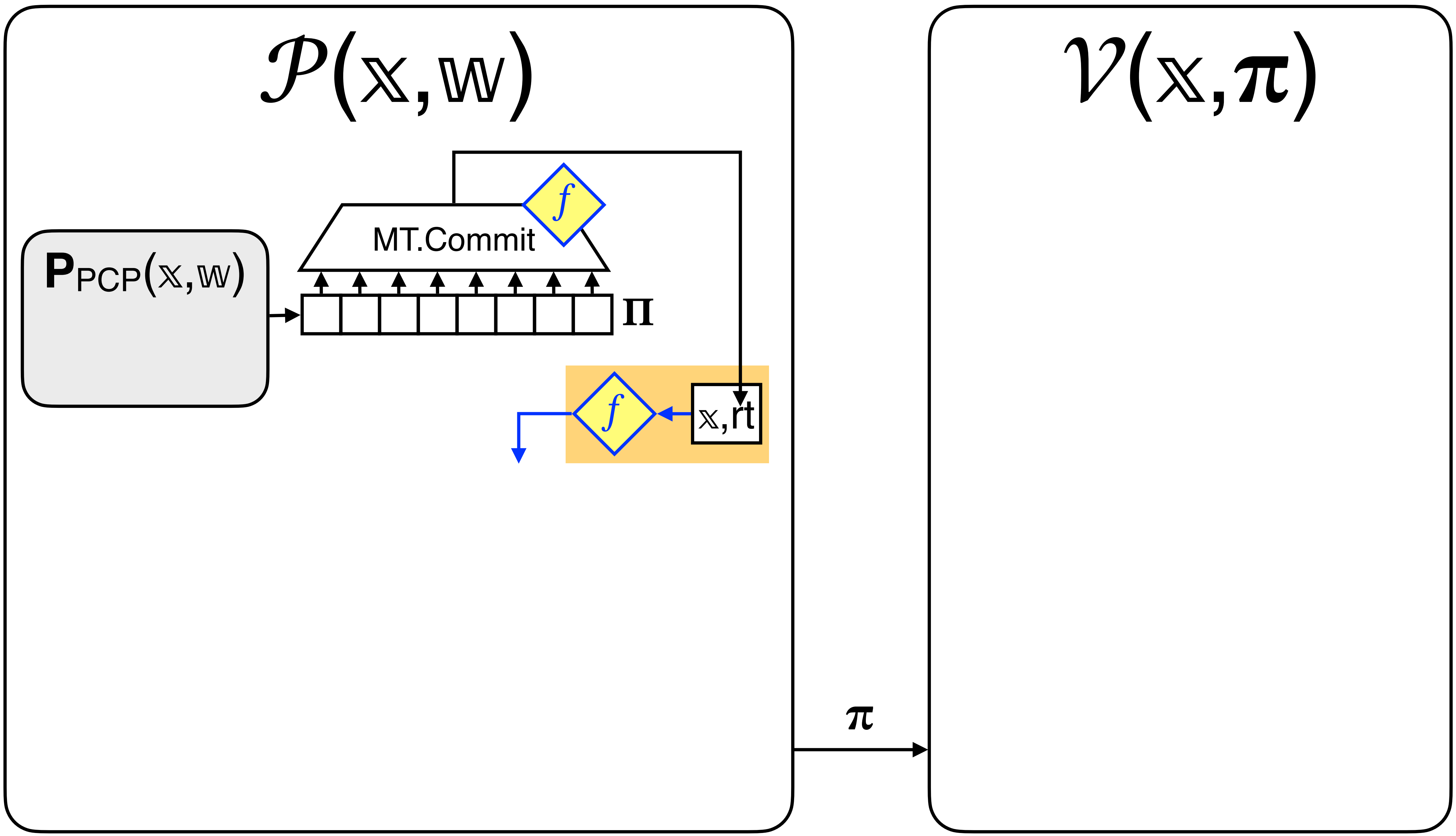
Idea: Use the random oracle to derive the PCP randomness.
(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)



Micali Protocol

Idea: Use the random oracle to derive the PCP randomness.

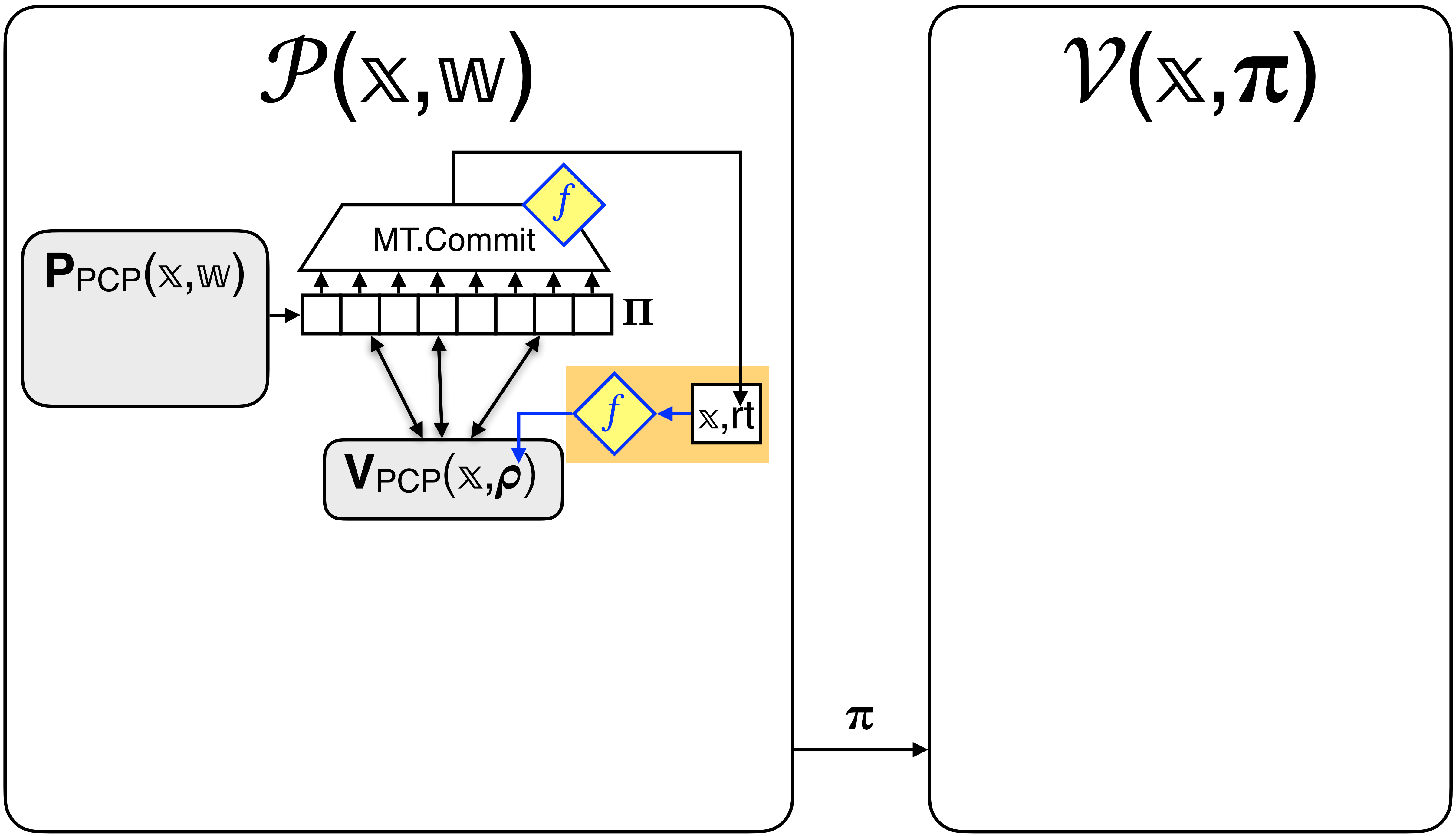
(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)



Micali Protocol

Idea: Use the random oracle to derive the PCP randomness.

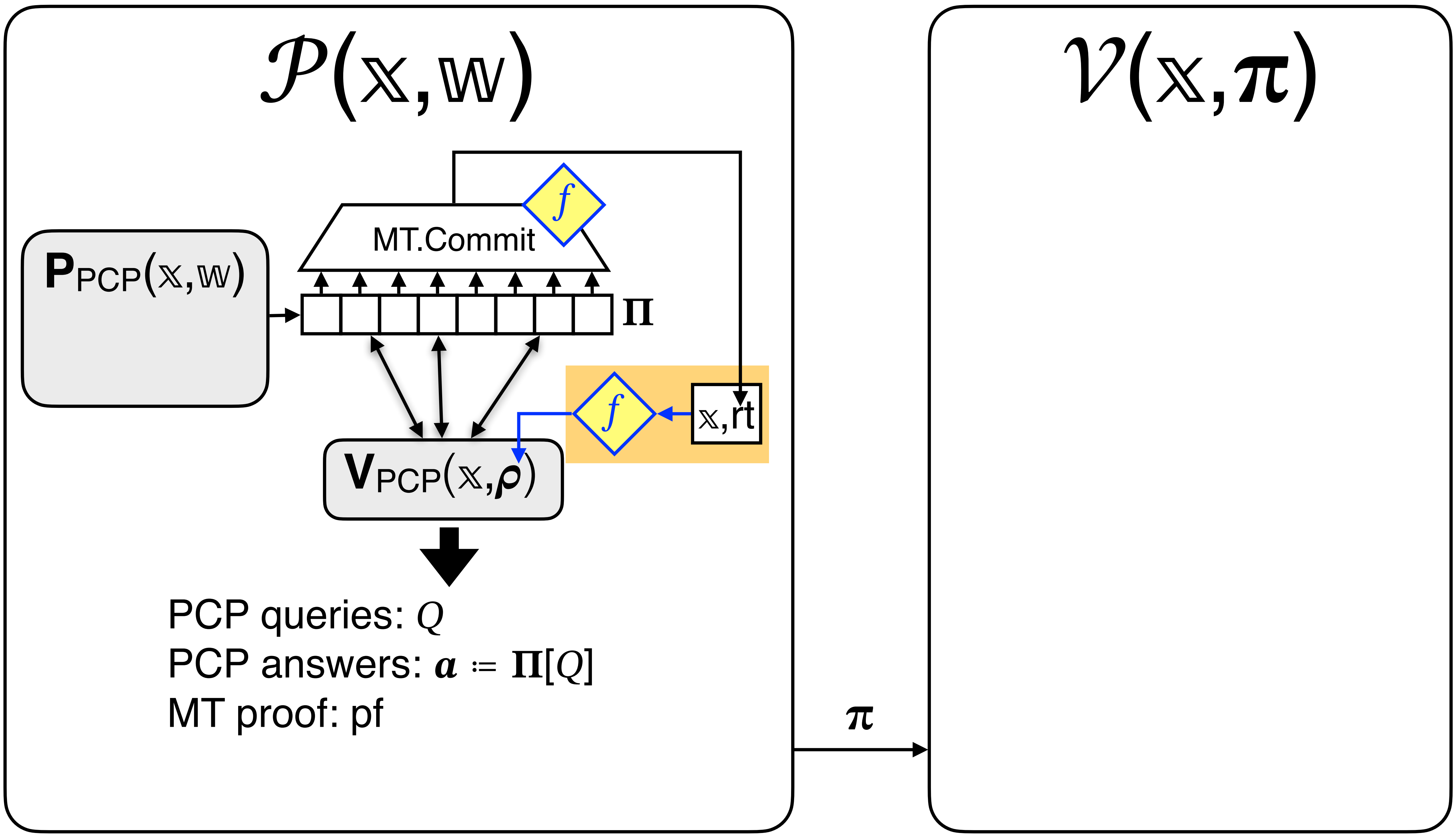
(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)



Micali Protocol

Idea: Use the random oracle to derive the PCP randomness.

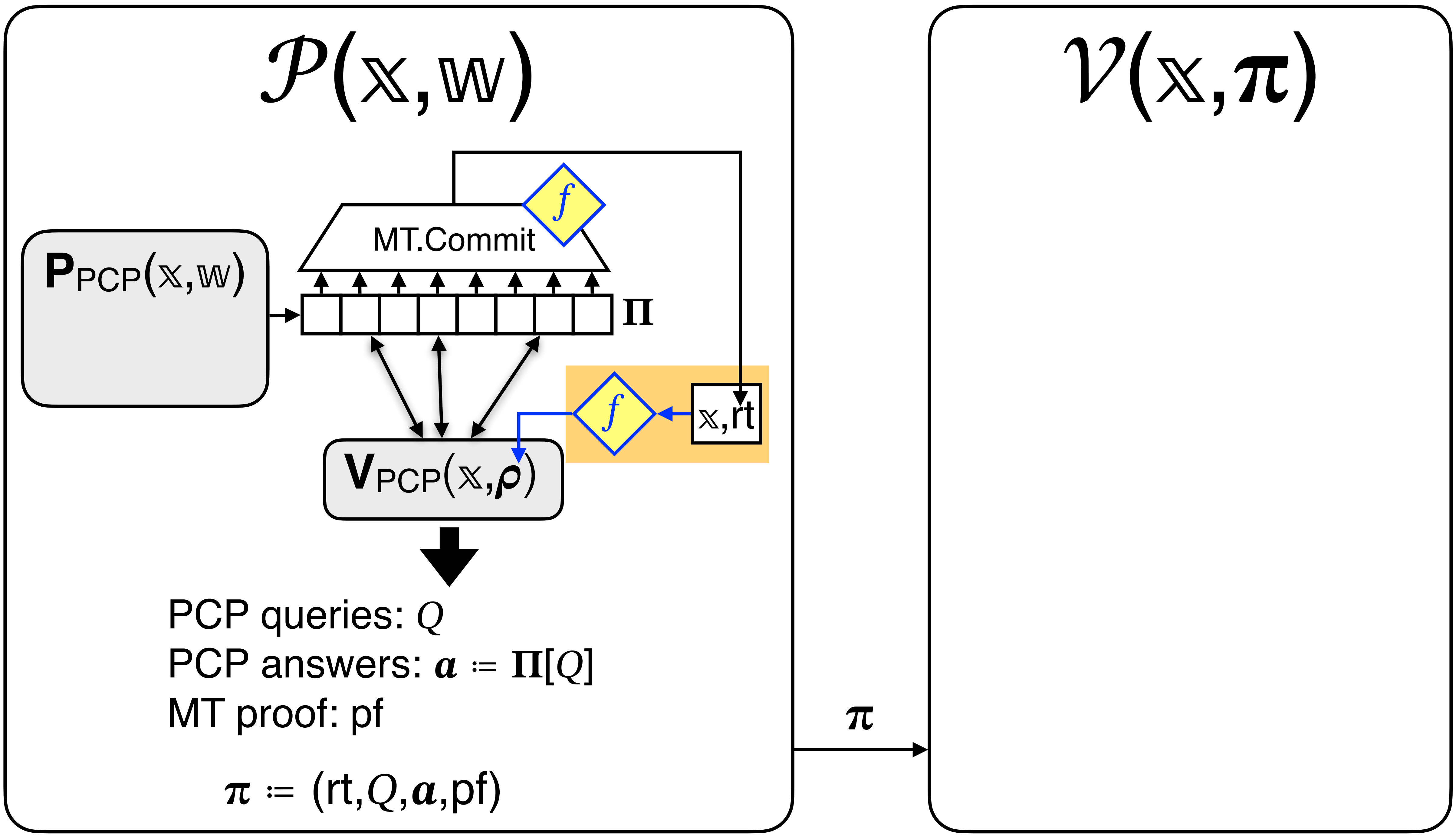
(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)



Micali Protocol

Idea: Use the random oracle to derive the PCP randomness.

(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)

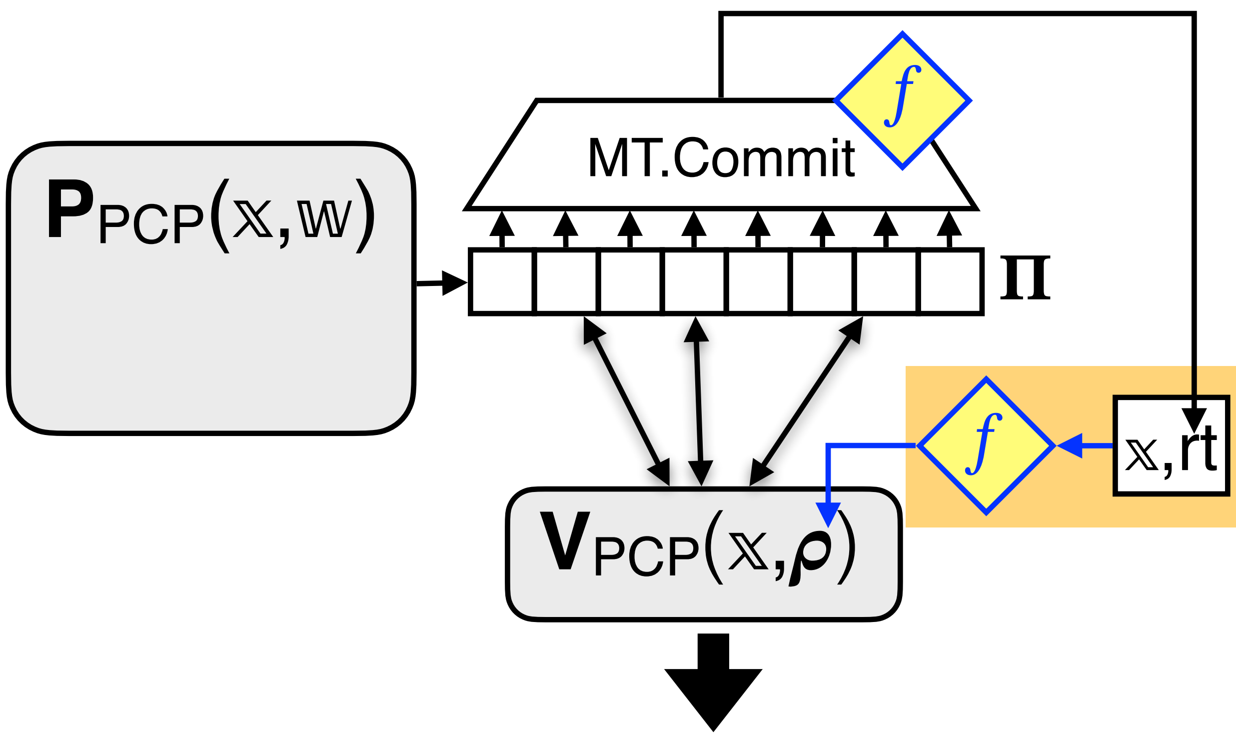


Micali Protocol

Idea: Use the random oracle to derive the PCP randomness.

(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)

$\mathcal{P}(\mathbb{x},\mathbb{w})$



PCP queries: Q

PCP answers: $\boldsymbol{a} \coloneqq \boldsymbol{\Pi}[Q]$

MT proof: pf

$\boldsymbol{\pi} \coloneqq (\mathrm{rt}, Q, \boldsymbol{a}, \mathrm{pf})$

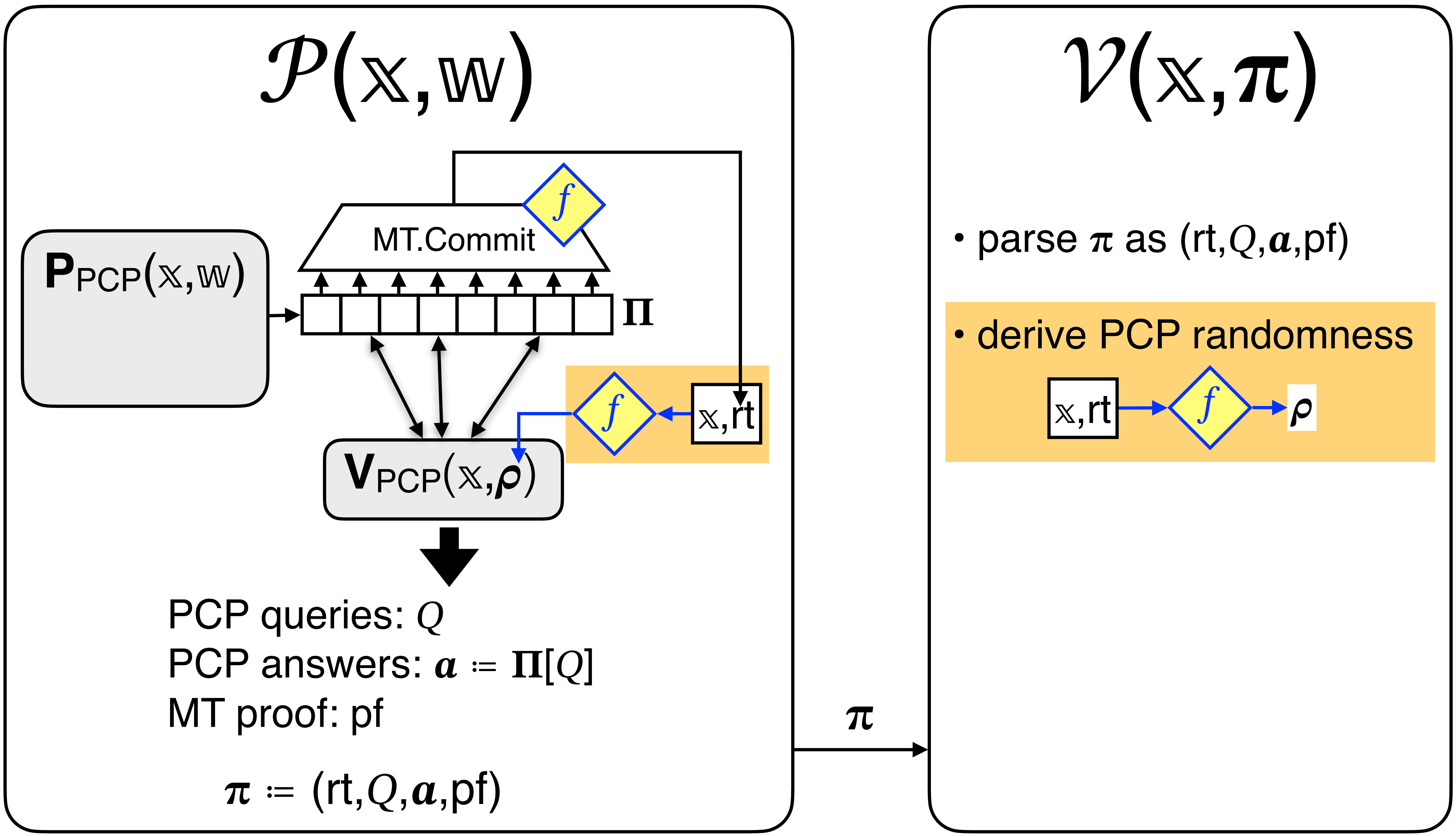
$\boldsymbol{\pi}$

$\mathcal{V}(\mathbb{x},\boldsymbol{\pi})$

- parse $\boldsymbol{\pi}$ as $(\mathrm{rt}, Q, \boldsymbol{a}, \mathrm{pf})$

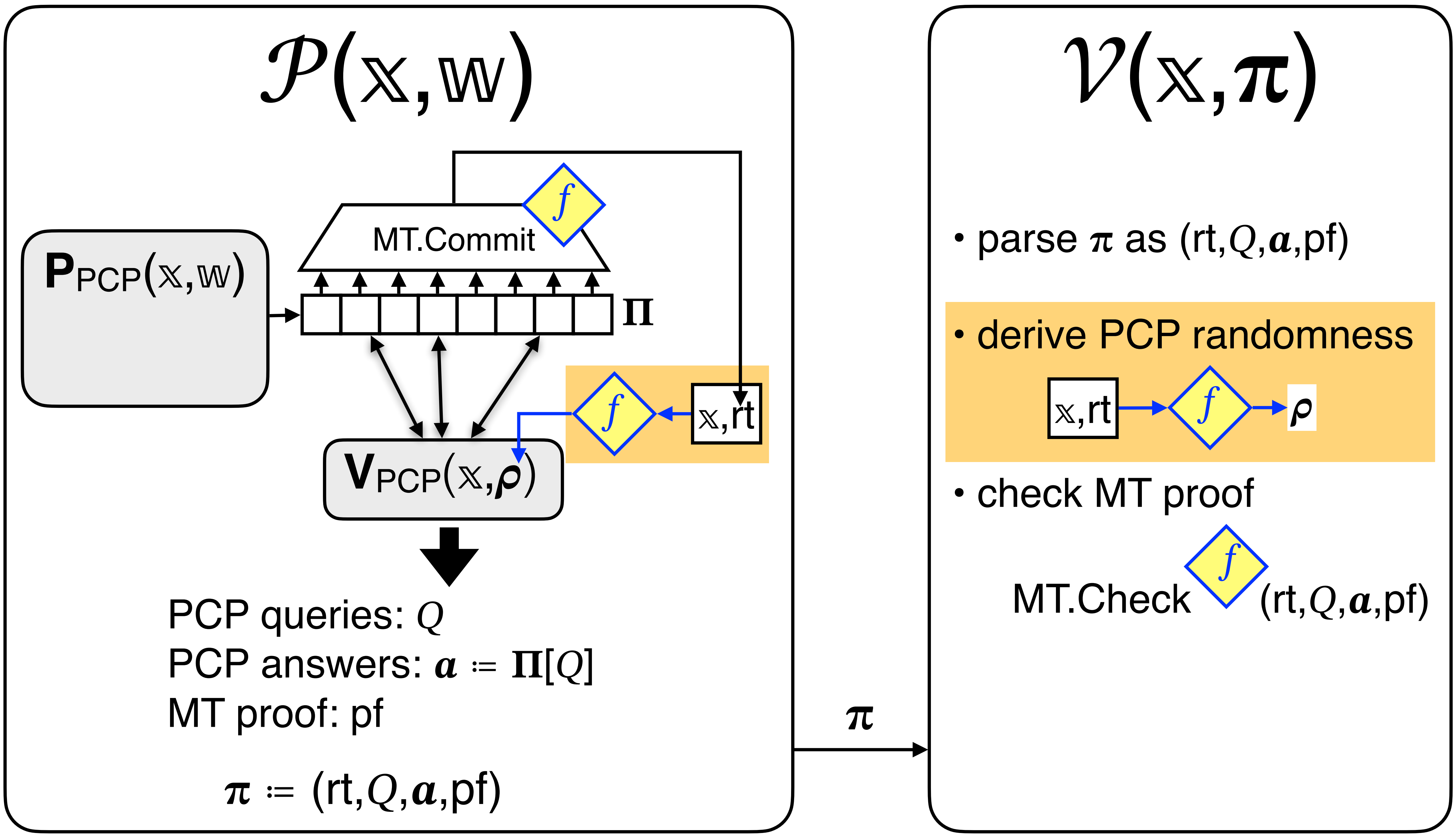
Micali Protocol

Idea: Use the random oracle to derive the PCP randomness.
(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)



Micali Protocol

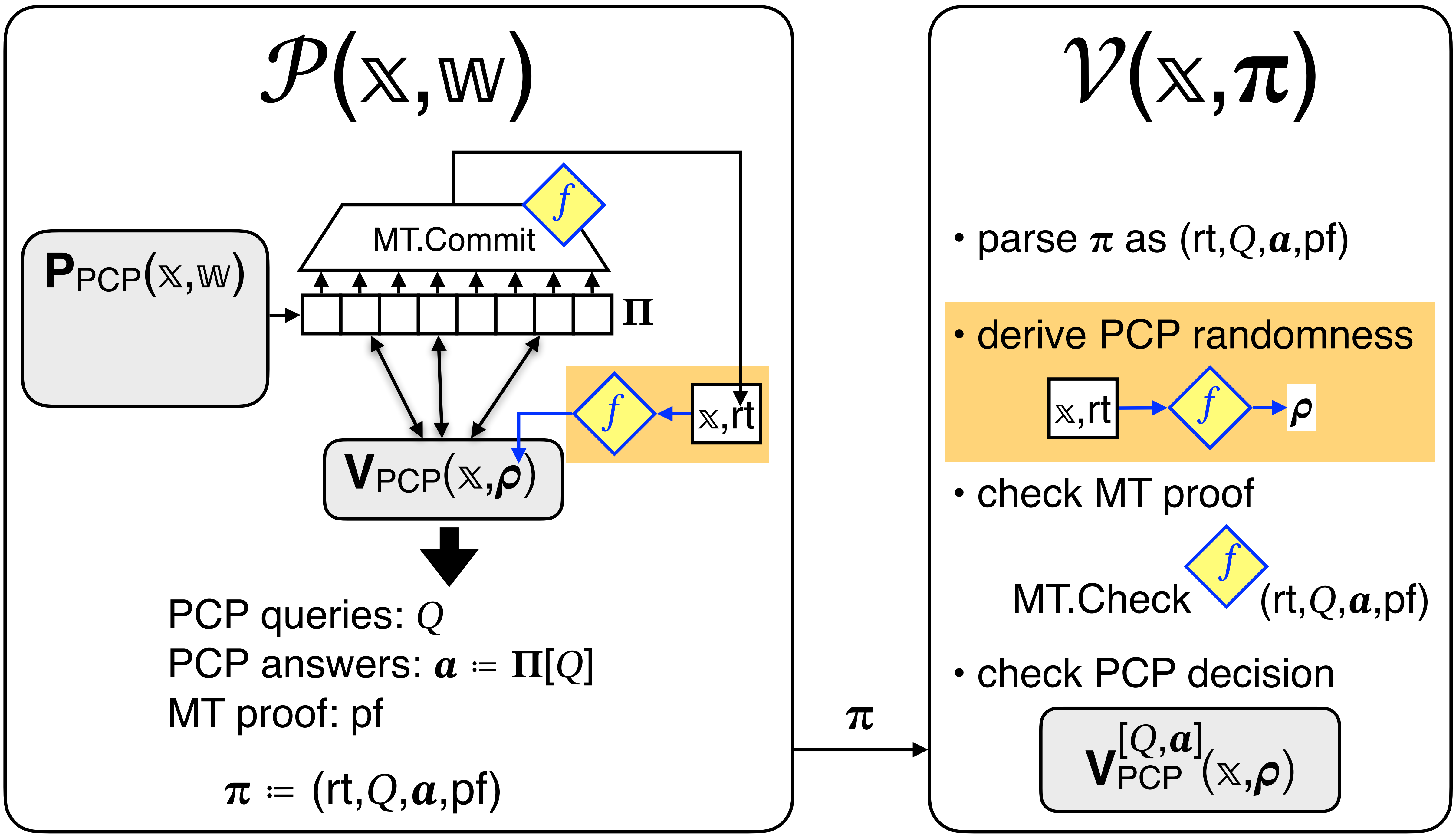
Idea: Use the random oracle to derive the PCP randomness.
(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)



Micali Protocol

Idea: Use the random oracle to derive the PCP randomness.

(For cryptographers: apply the Fiat–Shamir transformation to the Kilian protocol.)



Micali Protocol

Micali Protocol

Time efficiency and communication are similar to the Kilian protocol.

Micali Protocol

Time efficiency and communication are similar to the Kilian protocol.

Security of the Micali protocol is DIFFERENT.

Micali Protocol

Time efficiency and communication are similar to the Kilian protocol.

Security of the Micali protocol is DIFFERENT.

An adversary can attack the PCP multiple times,
by trying different Merkle roots to obtain different PCP randomness.

Micali Protocol

Time efficiency and communication are similar to the Kilian protocol.

Security of the Micali protocol is DIFFERENT.

An adversary can attack the PCP multiple times,
by trying different Merkle roots to obtain different PCP randomness.

> **Theorem:** $\forall$ t-query adversary $\mathcal{A}$,
>
> $$\Pr[\mathcal{A} \text{ breaks Micali}] \leq t \cdot \epsilon_{\mathrm{PCP}} + \frac{t^2}{2^\lambda}$$

Micali Protocol

Time efficiency and communication are similar to the Kilian protocol.

Security of the Micali protocol is DIFFERENT.

An adversary can attack the PCP multiple times,
by trying different Merkle roots to obtain different PCP randomness.

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks Micali}] \leq t \cdot \epsilon_{\mathrm{PCP}} + \frac{t^2}{2^\lambda}$$

➔ The PCP must be far more secure in Micali (vs in Kilian).

Micali Protocol

Time efficiency and communication are similar to the Kilian protocol.

Security of the Micali protocol is DIFFERENT.

An adversary can attack the PCP multiple times,
by trying different Merkle roots to obtain different PCP randomness.

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks Micali}] \leq t \cdot \epsilon_{\mathrm{PCP}} + \frac{t^2}{2^\lambda}$$

➔ The PCP must be far more secure in Micali (vs in Kilian).

Example:

Set $\varepsilon_{\mathrm{PCP}} = 2^{-128}/4$ and $\lambda = 256+2 = 258$.

Every 2^{128}-query adversary breaks Micali w.p. $\leq 1/4 + 1/4 = 1/2$.

Micali Protocol

Time efficiency and communication are similar to the Kilian protocol.

Security of the Micali protocol is DIFFERENT.

An adversary can attack the PCP multiple times,
by trying different Merkle roots to obtain different PCP randomness.

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks Micali}] \le t \cdot \epsilon_{\mathrm{PCP}} + \frac{t^2}{2^\lambda}$$

➔ The PCP must be far more secure in Micali (vs in Kilian).

Example:

Set $\varepsilon_{\mathrm{PCP}} = 2^{-128}/4$ and $\lambda = 256+2 = 258$.

Every 2^{128}-query adversary breaks Micali w.p. $\le 1/4 + 1/4 = 1/2$.

Known PCPs are inefficient, so the Micali protocol is not used.

In Practice:
Succinct Arguments from IOPs

Interactive Oracle Proofs

Interactive Oracle Proofs

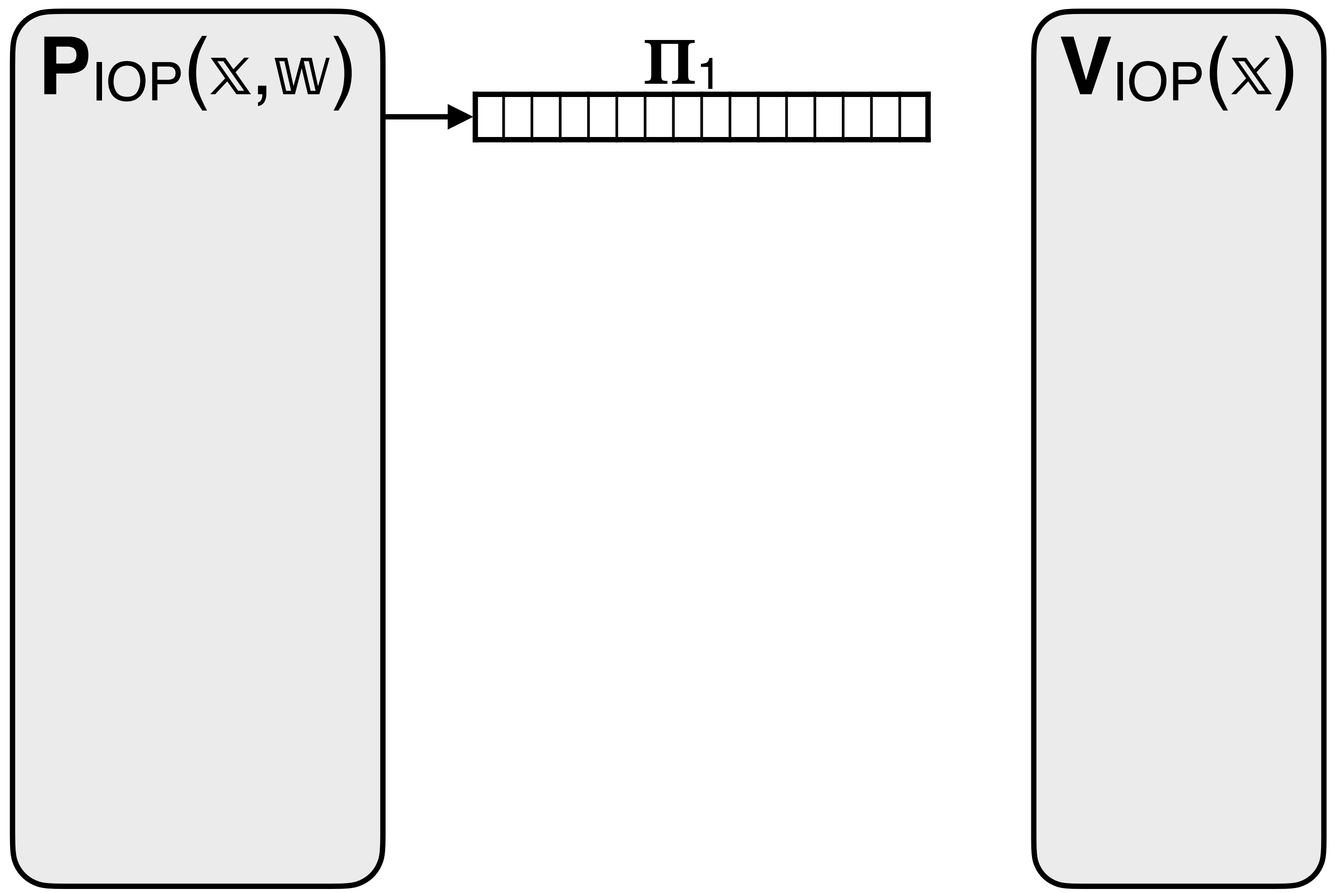
An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.

Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.

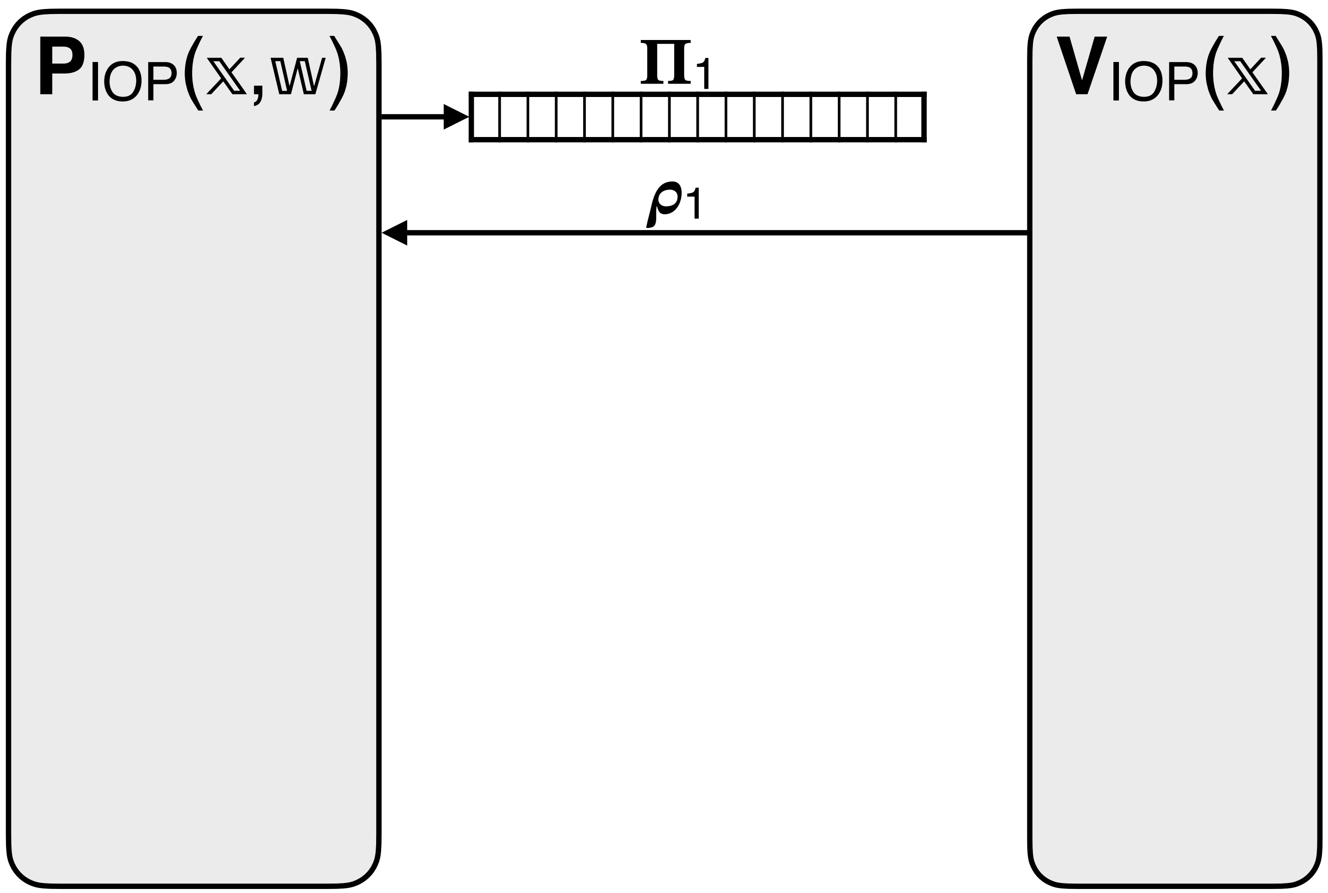
Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



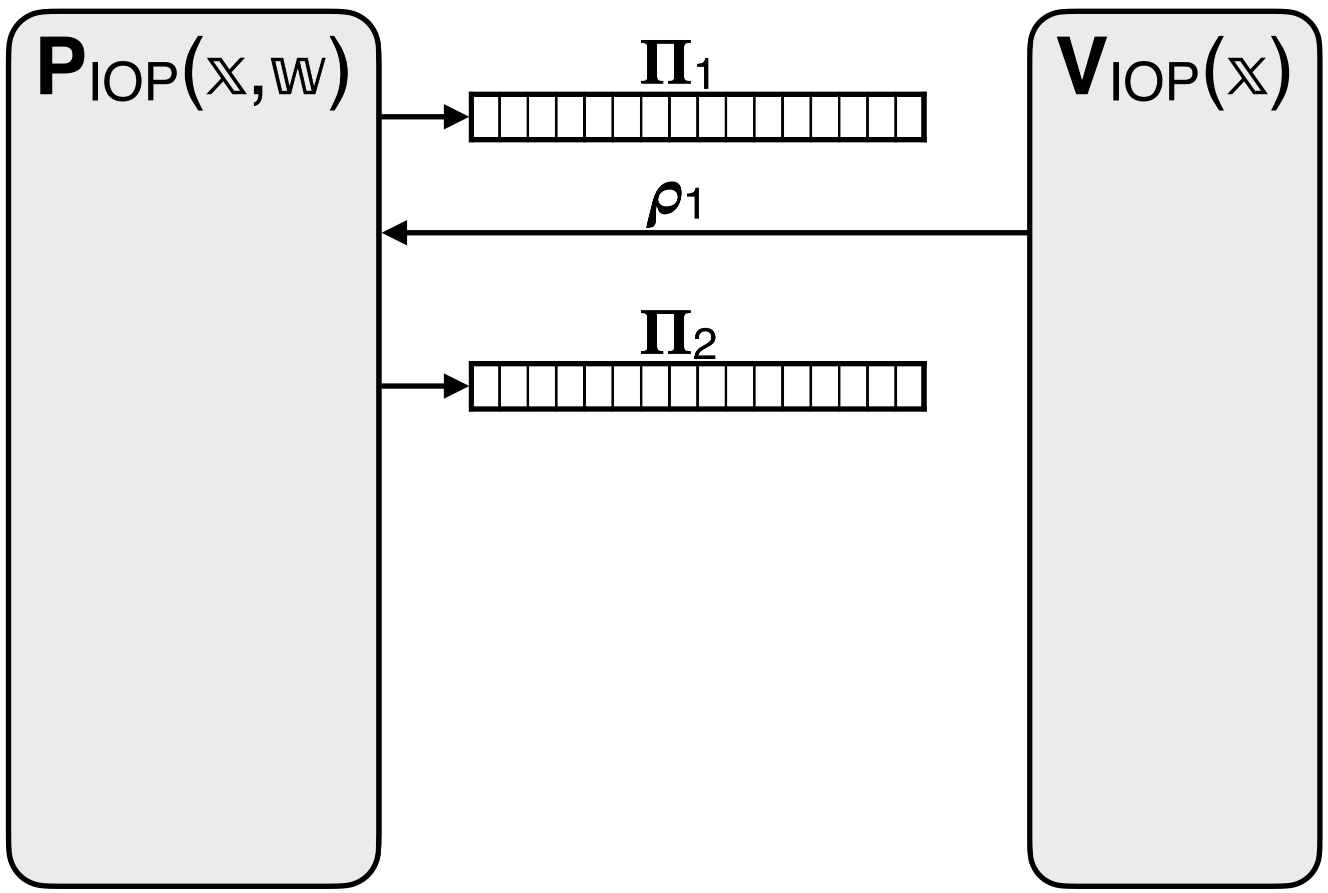
Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



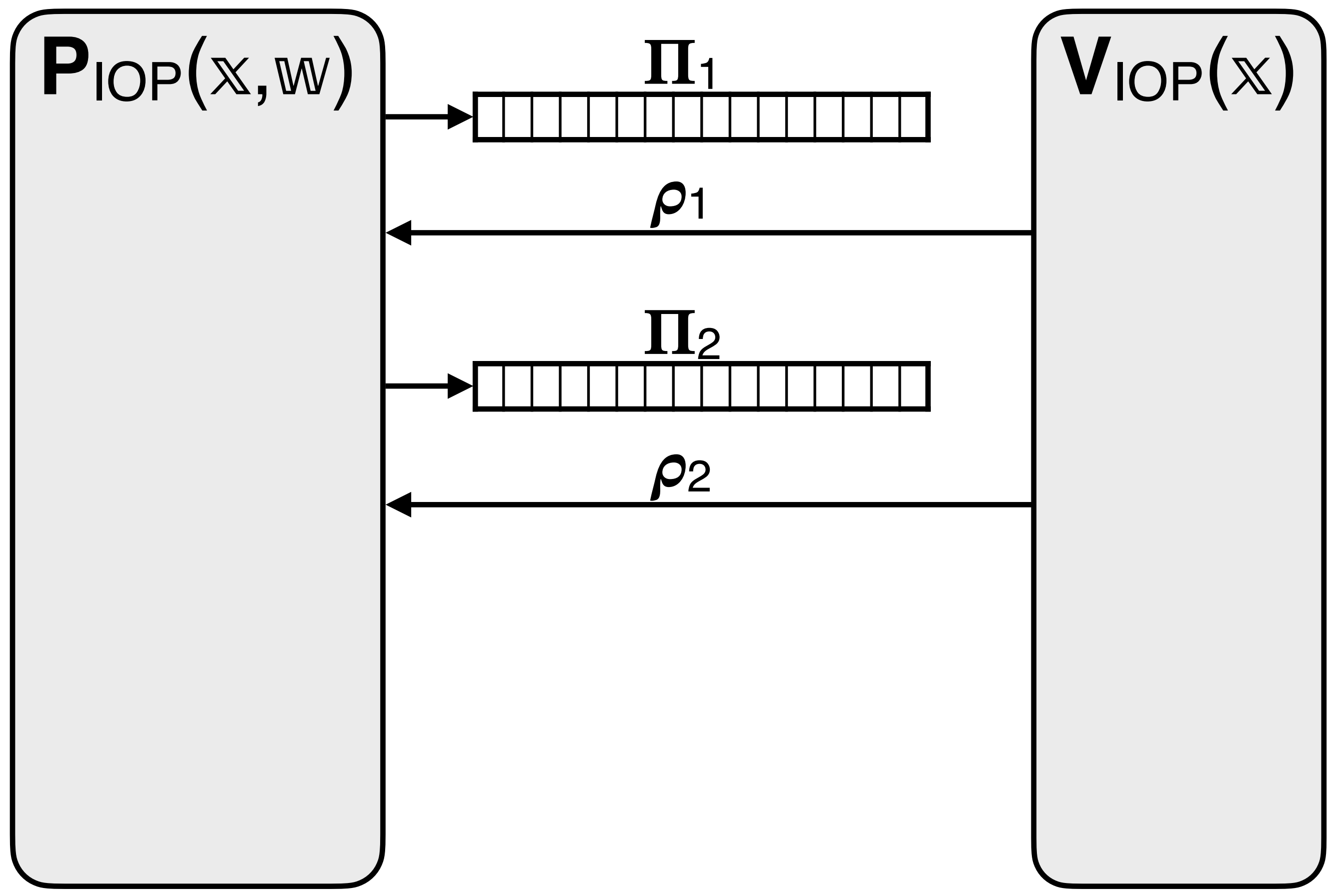
Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



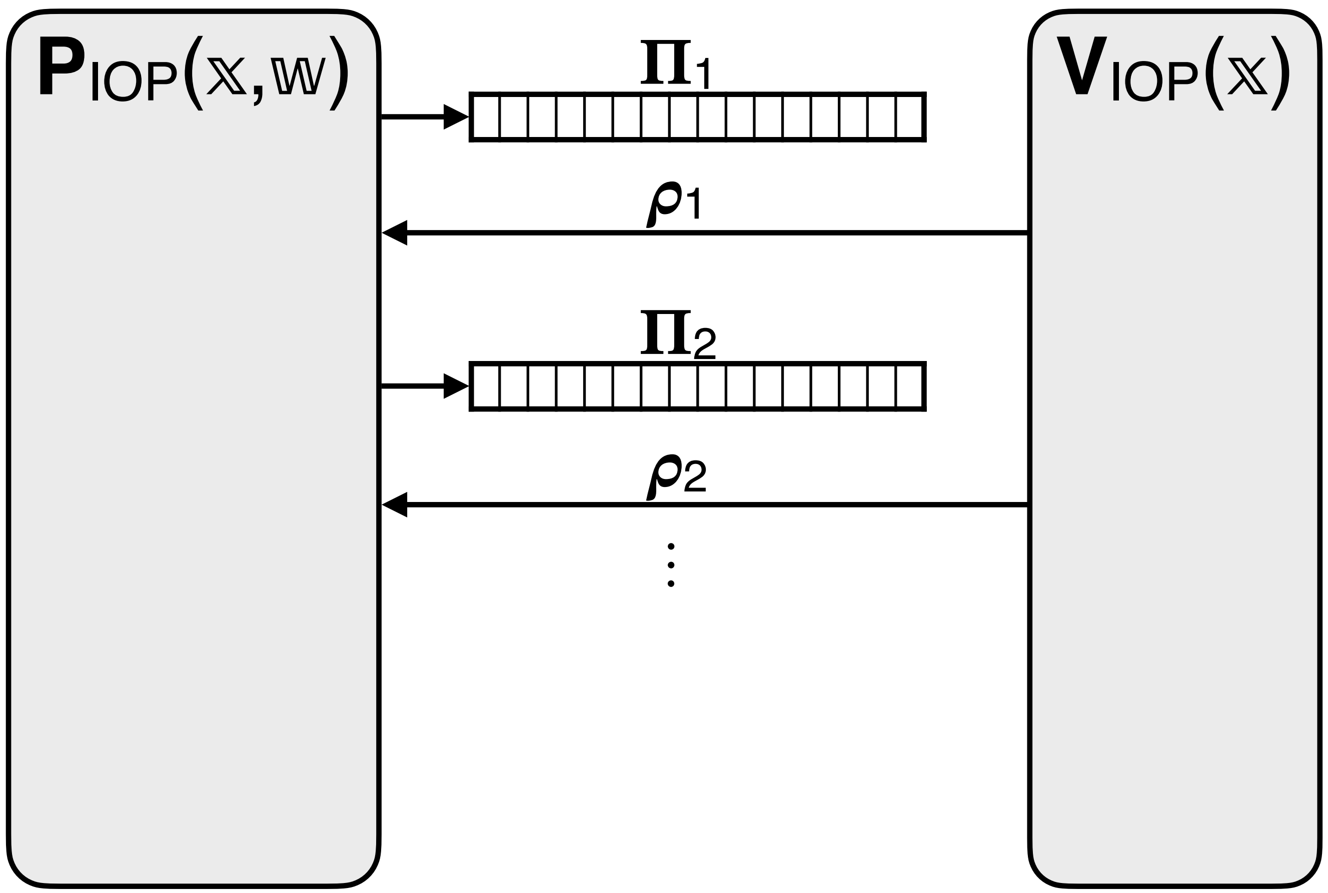
Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



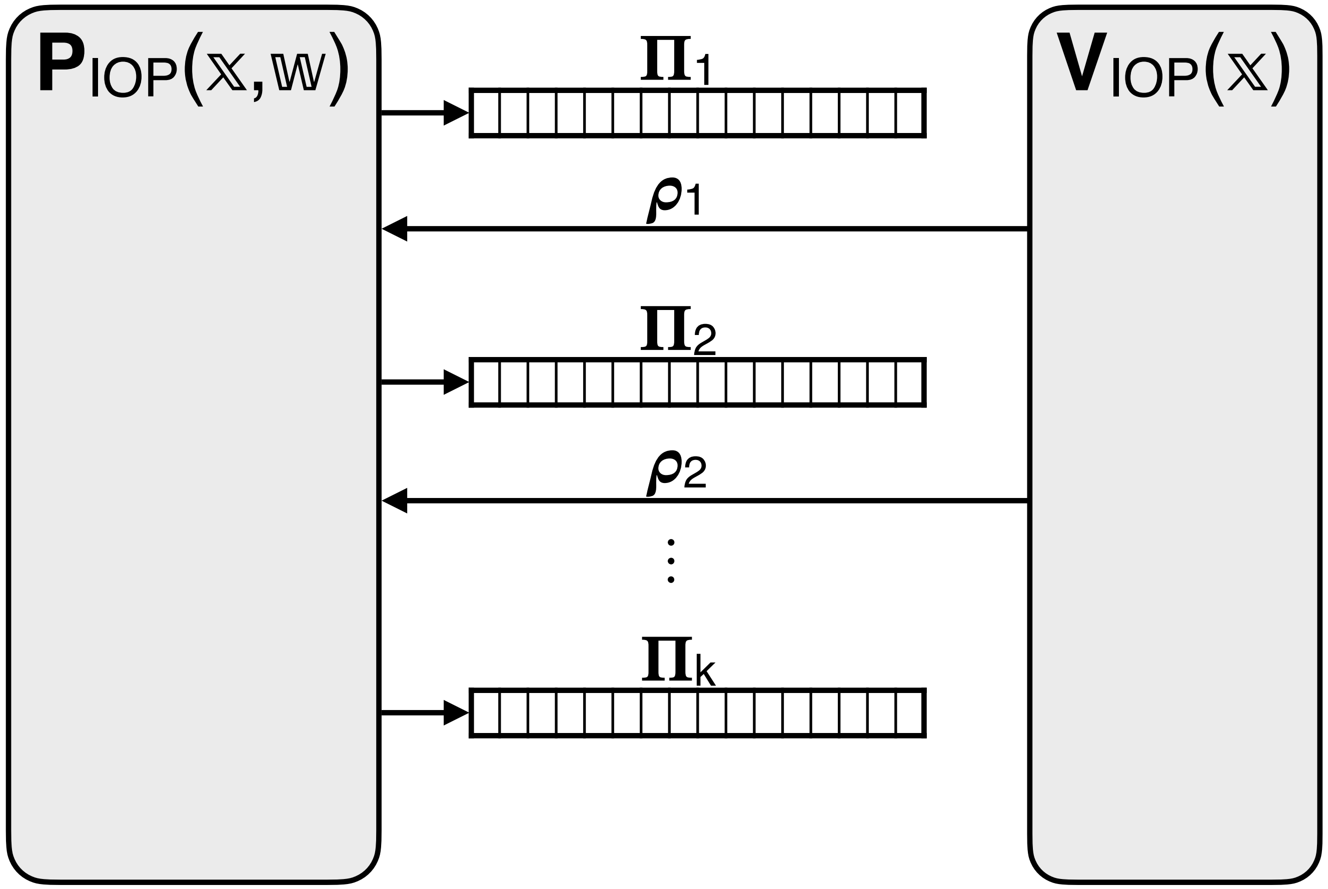
Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



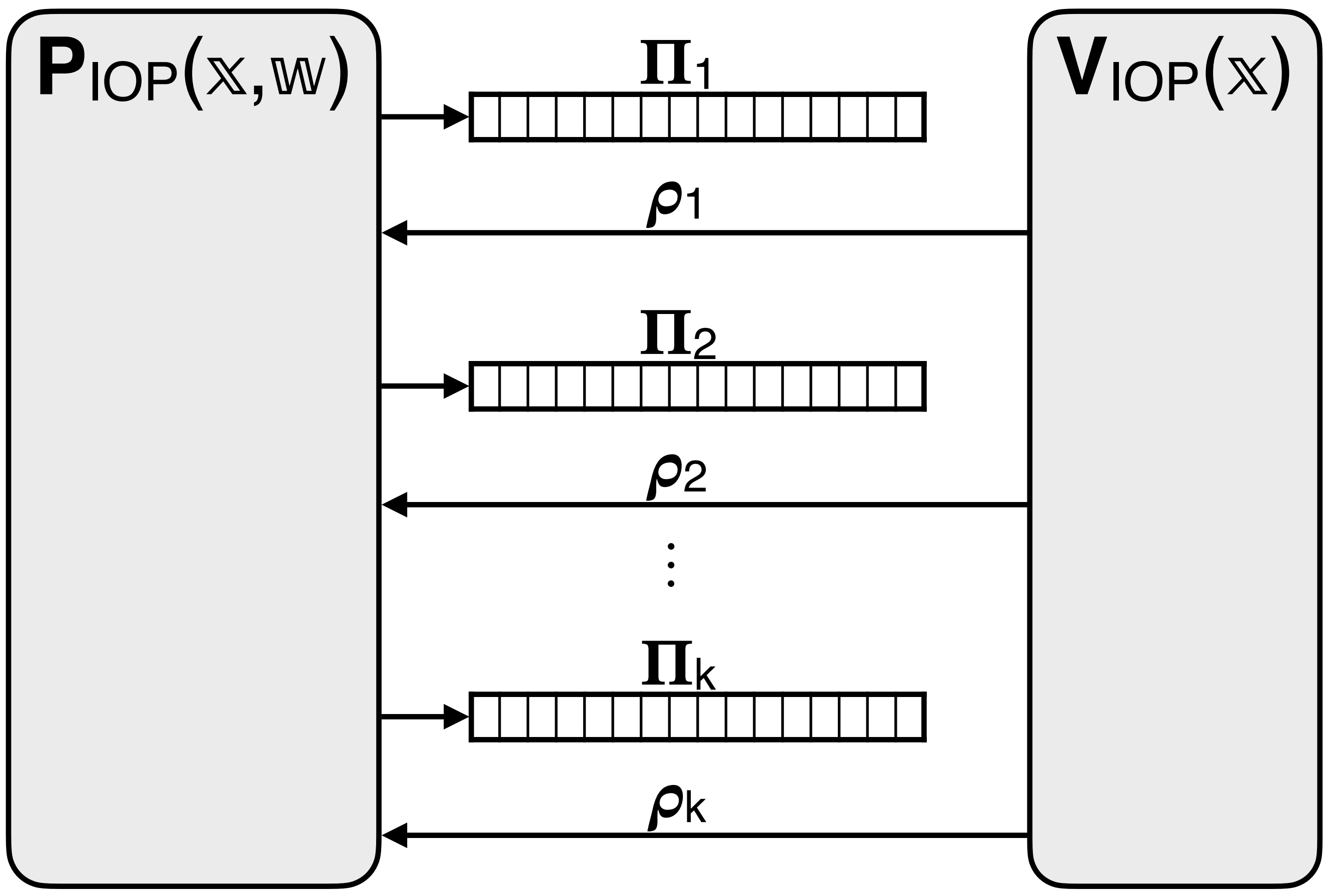
Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



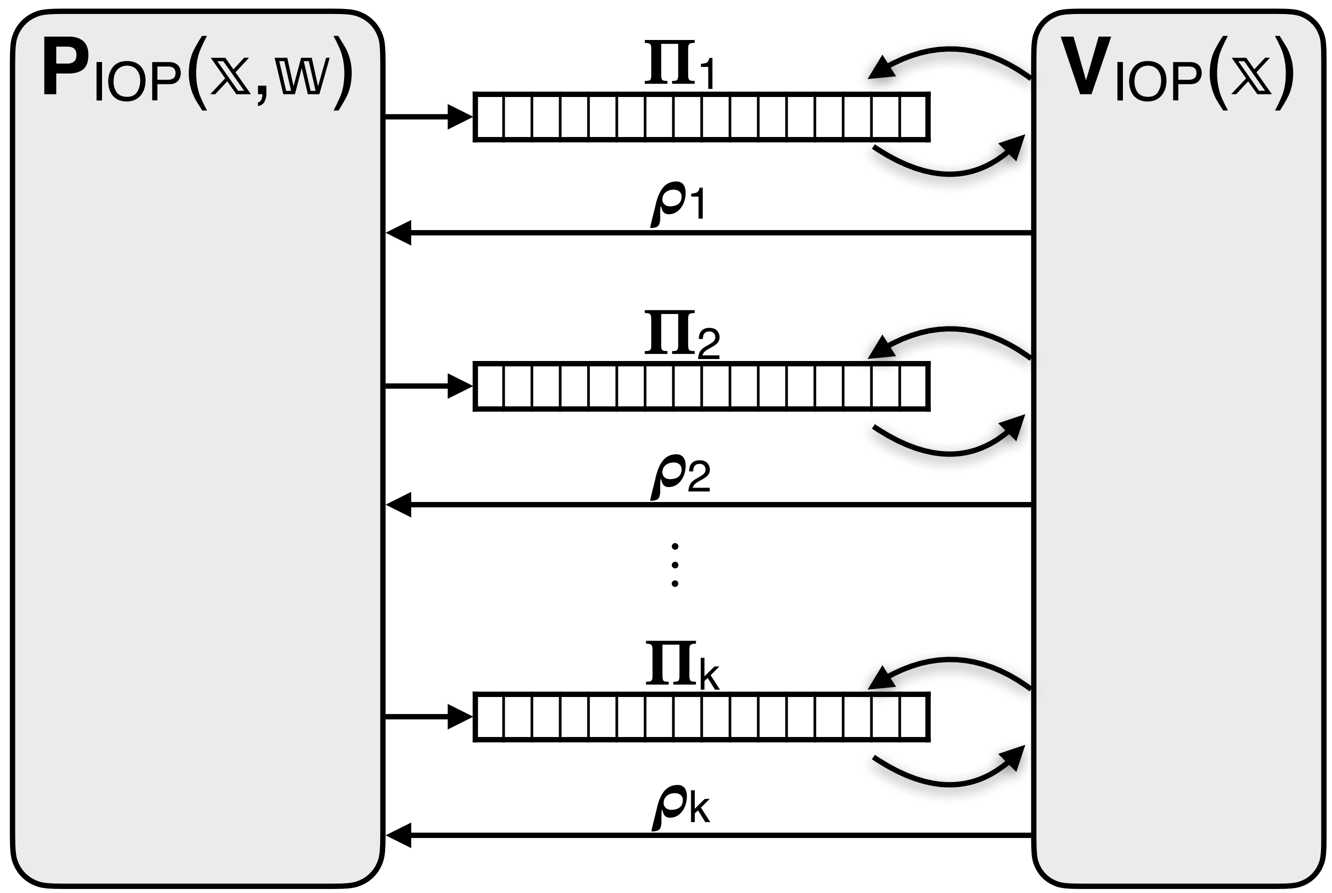
Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



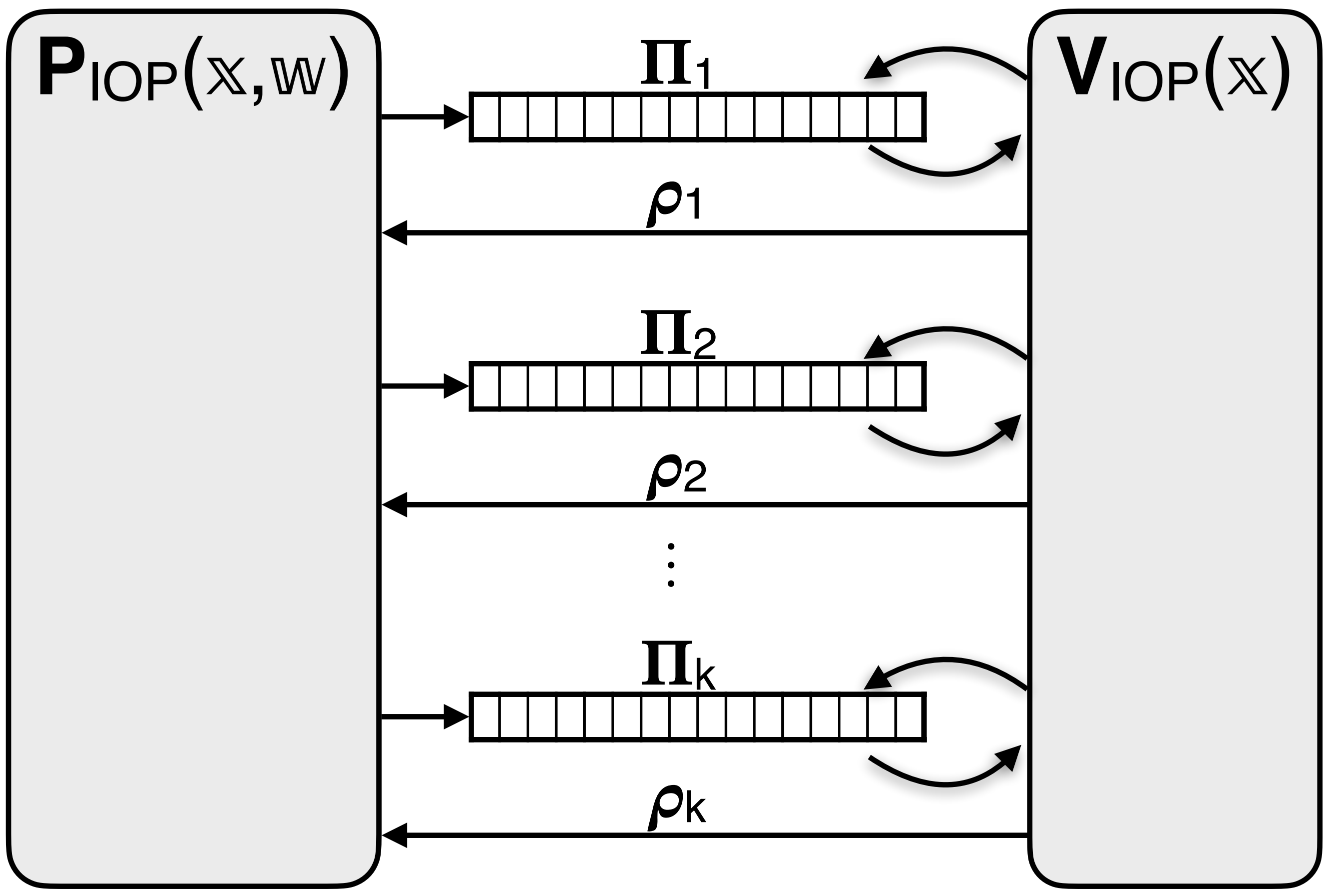
Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



Interactive Oracle Proofs

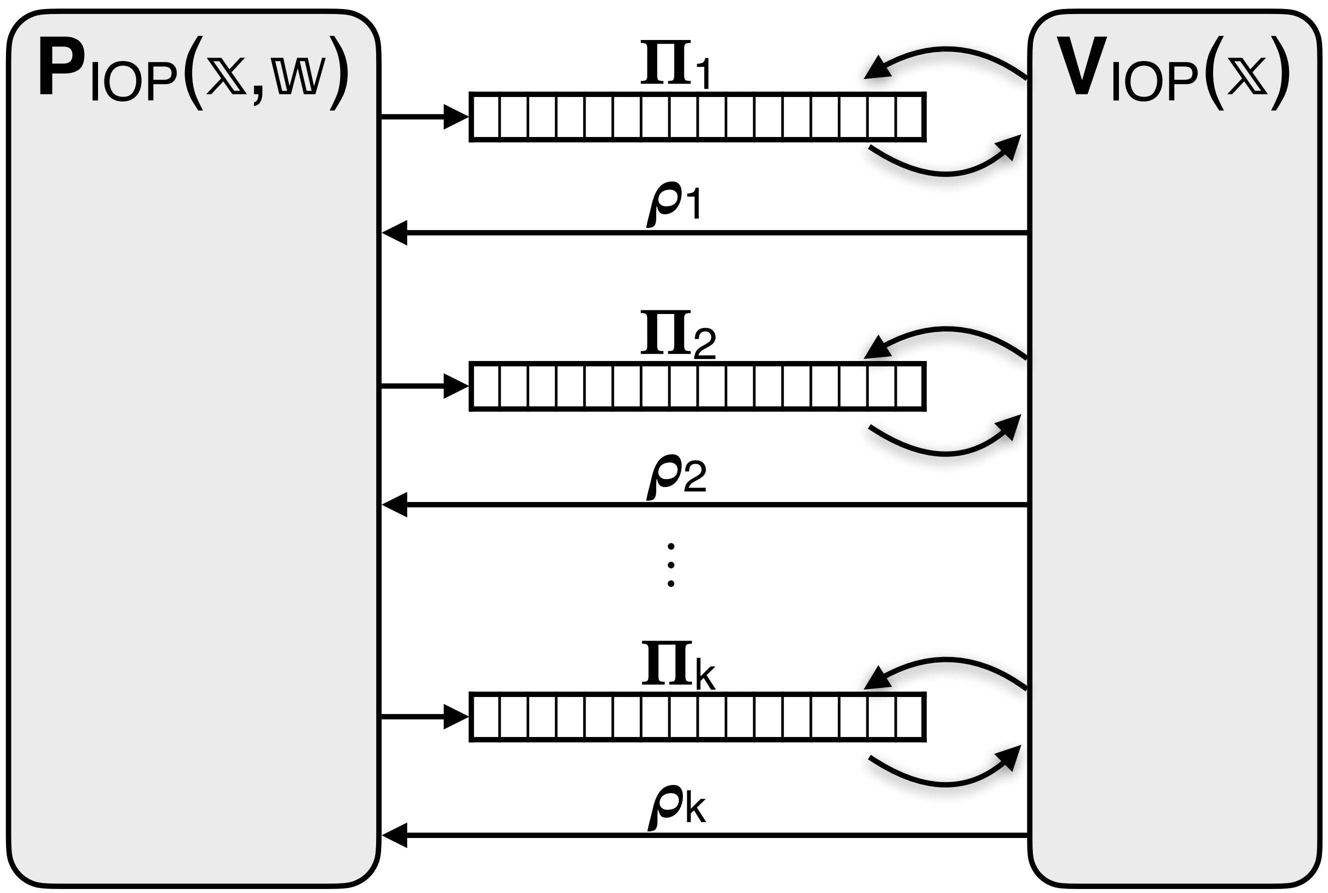
An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



Known IOPs are vastly more efficient than known PCPs.

Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.

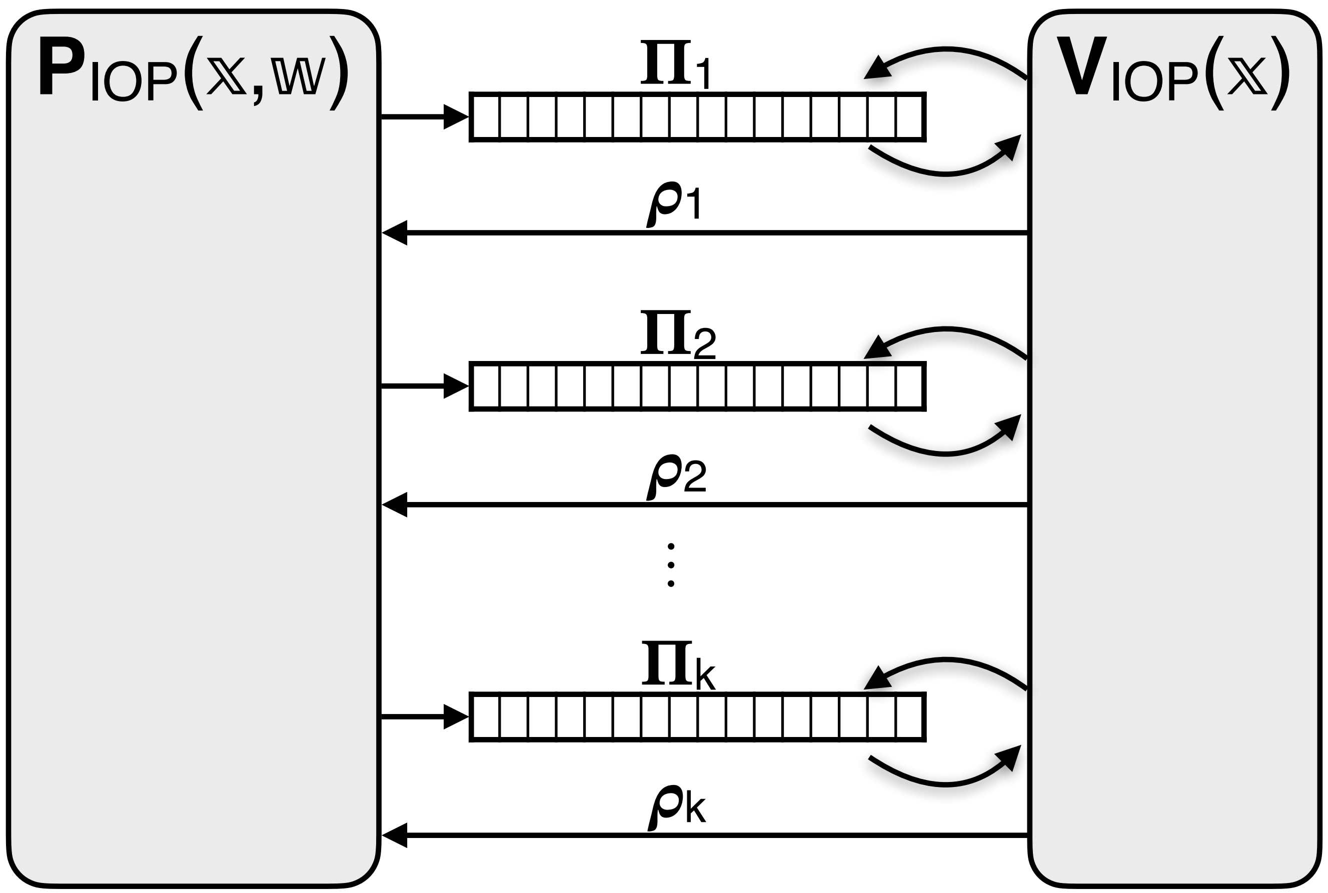


Known IOPs are vastly more efficient than known PCPs.

BCS showed how to construct **succinct arguments from IOPs**.

Interactive Oracle Proofs

An **IOP** is a model of proof system where the verifier can leverage **randomness**, **interaction**, and **oracle access**.



Known IOPs are vastly more efficient than known PCPs.

BCS showed how to construct **succinct arguments from IOPs**.

IOPs underlie essentially all modern succinct arguments.

Intuition for BCS Protocol

Intuition for BCS Protocol

Recap on succinct arguments from PCPs:

Intuition for BCS Protocol

Recap on succinct arguments from PCPs:

- **Kilian** (interactive) **protocol:**
  commit to PCP string via a Merkle Tree then open

Intuition for BCS Protocol

Recap on succinct arguments from PCPs:

- **Kilian** (interactive) **protocol:**
  commit to PCP string via a Merkle Tree then open
- **Micali** (non-interactive) **protocol:**
  derive PCP randomness using a random oracle

Intuition for BCS Protocol

Recap on succinct arguments from PCPs:

- **Kilian** (interactive) **protocol:**
  commit to PCP string via a Merkle Tree then open
- **Micali** (non-interactive) **protocol:**
  derive PCP randomness using a random oracle

These ideas can be adapted to work with IOPs (multi-round PCPs):

Intuition for BCS Protocol

Recap on succinct arguments from PCPs:

- **Kilian** (interactive) **protocol:**
  commit to PCP string via a Merkle Tree then open
- **Micali** (non-interactive) **protocol:**
  derive PCP randomness using a random oracle

These ideas can be adapted to work with IOPs (multi-round PCPs):

- **iBCS** (interactive) **protocol:**
  commit to each IOP string via a Merkle Tree then open

Intuition for BCS Protocol

Recap on succinct arguments from PCPs:

- **Kilian** (interactive) **protocol:**
  commit to PCP string via a Merkle Tree then open
- **Micali** (non-interactive) **protocol:**
  derive PCP randomness using a random oracle

These ideas can be adapted to work with IOPs (multi-round PCPs):

- **iBCS** (interactive) **protocol:**
  commit to each IOP string via a Merkle Tree then open
- **BCS** (non-interactive) **protocol:**
  derive each IOP randomness using a random oracle
  (and enforce their order via a hash chain)

Intuition for BCS Protocol

Recap on succinct arguments from PCPs:

- **Kilian** (interactive) **protocol:**
  commit to PCP string via a Merkle Tree then open
- **Micali** (non-interactive) **protocol:**
  derive PCP randomness using a random oracle

These ideas can be adapted to work with IOPs (multi-round PCPs):

- **iBCS** (interactive) **protocol:**
  commit to each IOP string via a Merkle Tree then open
- **BCS** (non-interactive) **protocol:**
  derive each IOP randomness using a random oracle
  (and enforce their order via a hash chain)

...

iBCS Protocol

iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.

iBCS Protocol

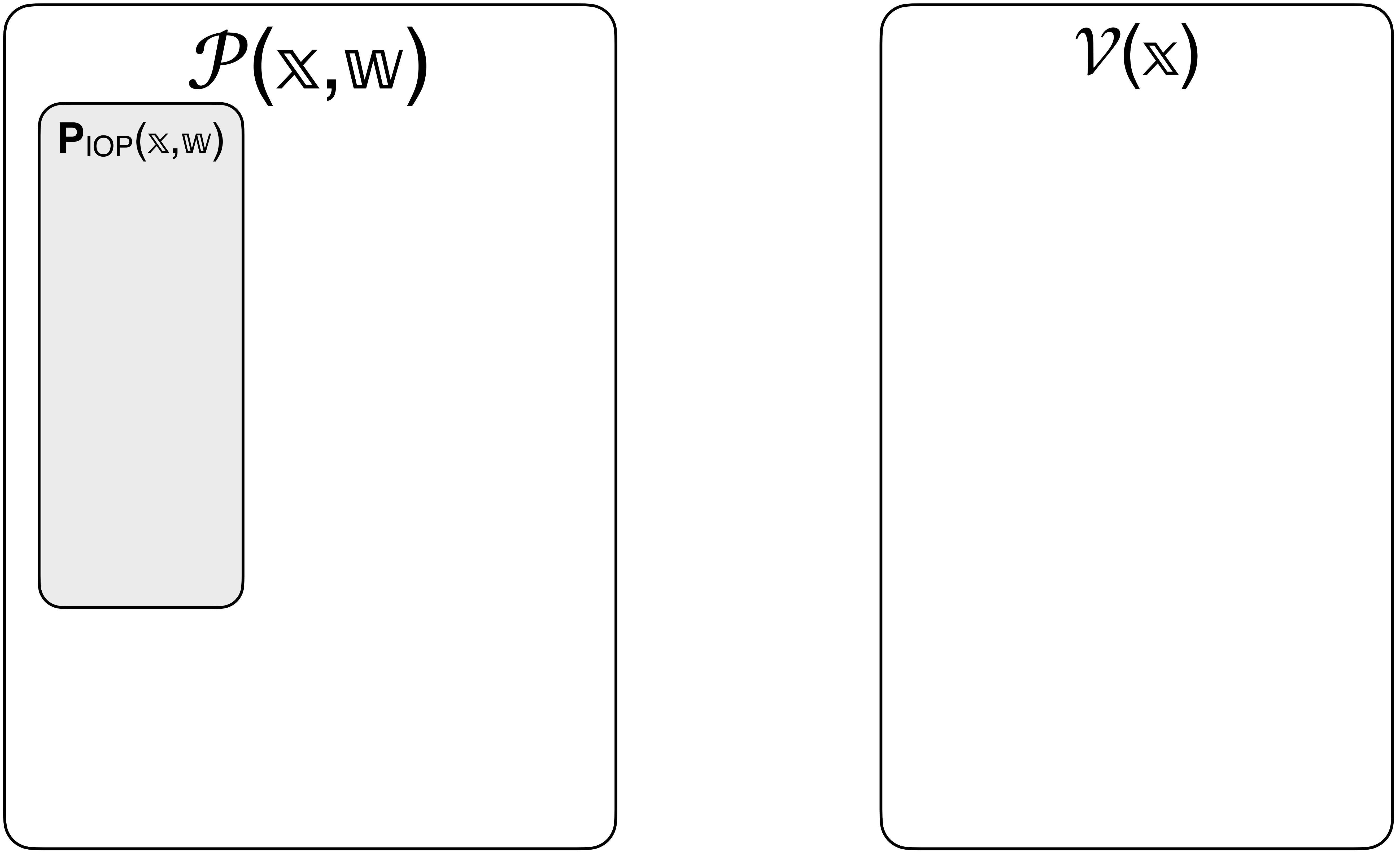
Idea: Use a Merkle Tree to commit to each IOP string.

$\mathcal{P}(\mathbb{x},\mathbb{w})$

$\mathcal{V}(\mathbb{x})$

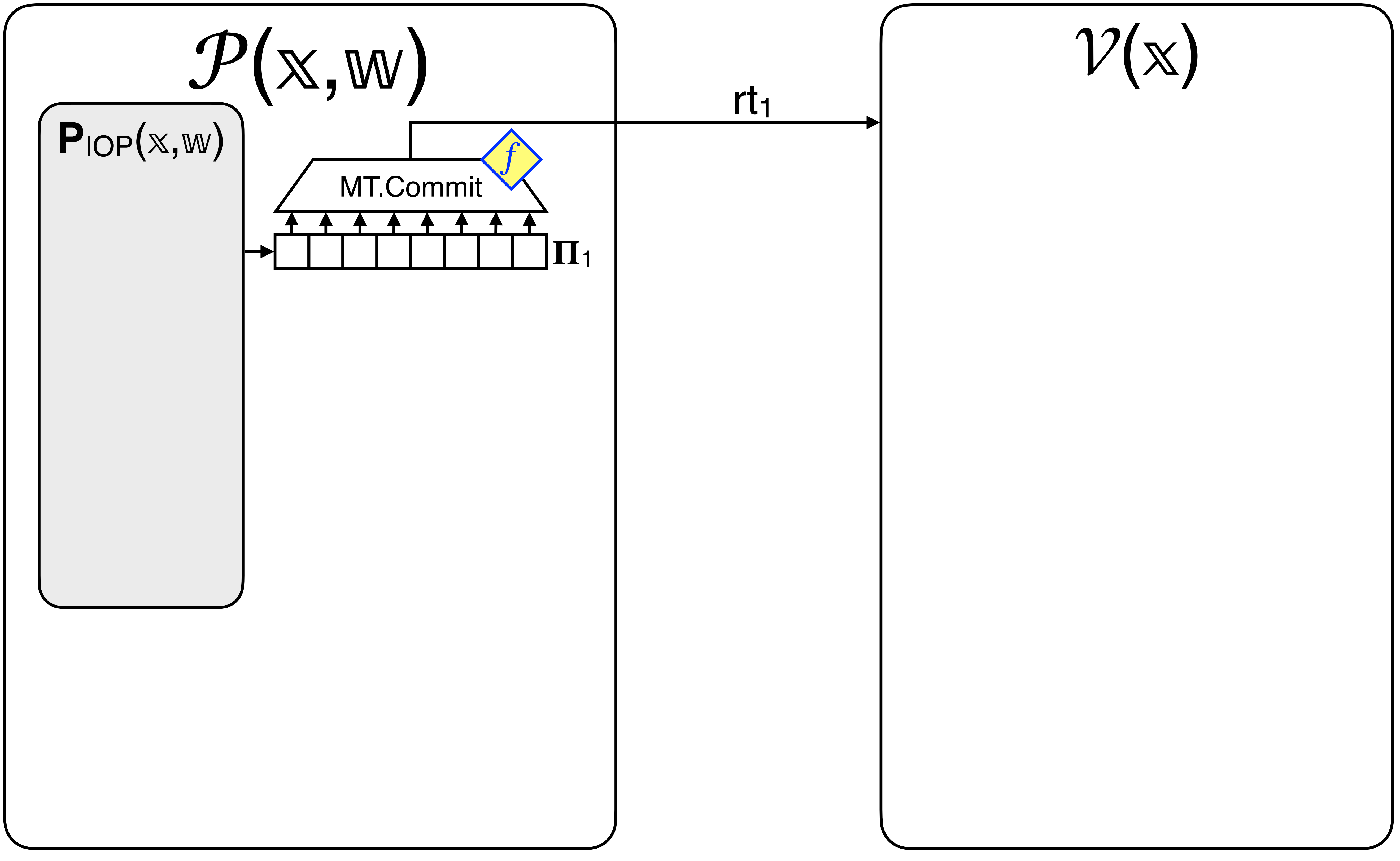
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



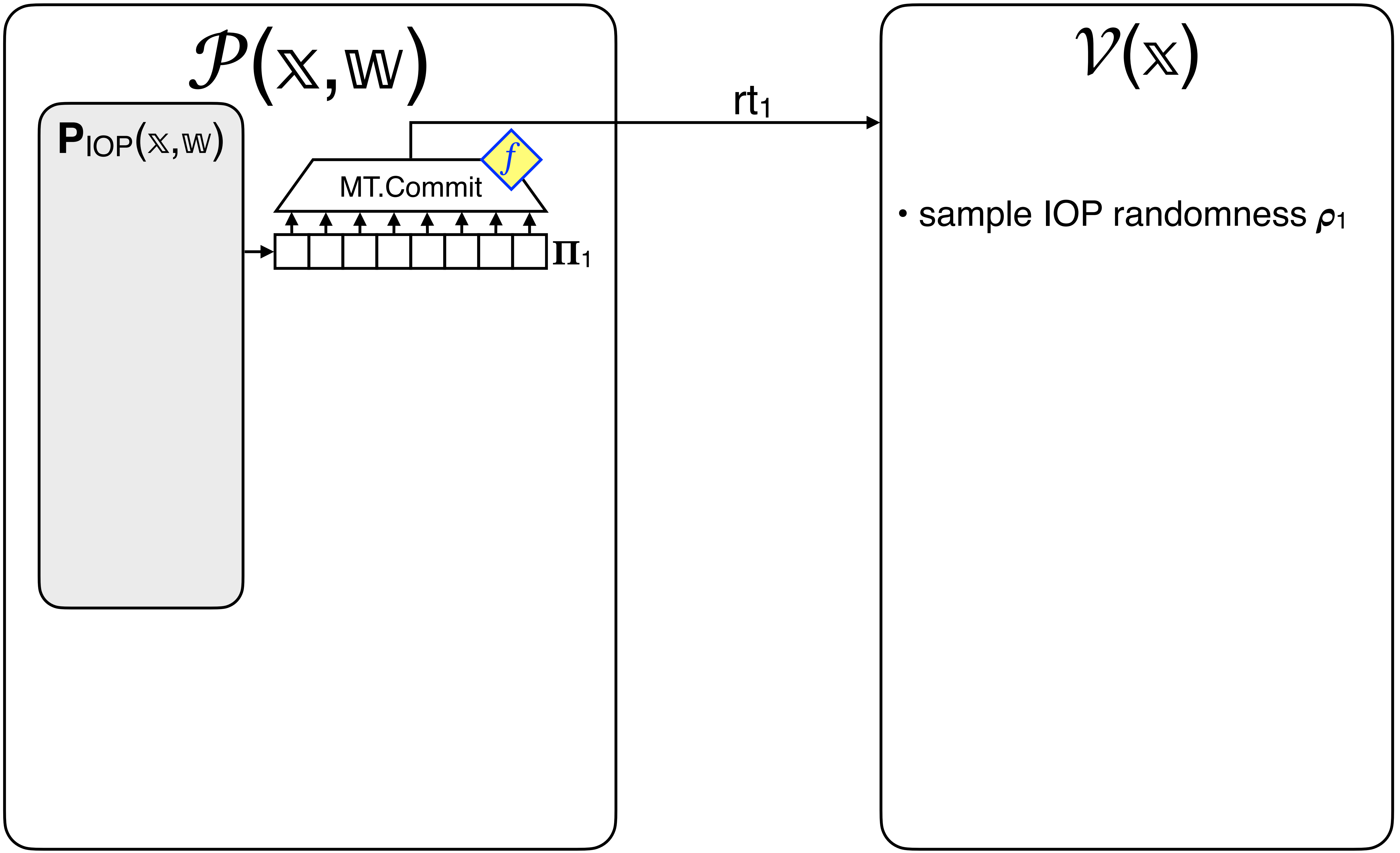
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



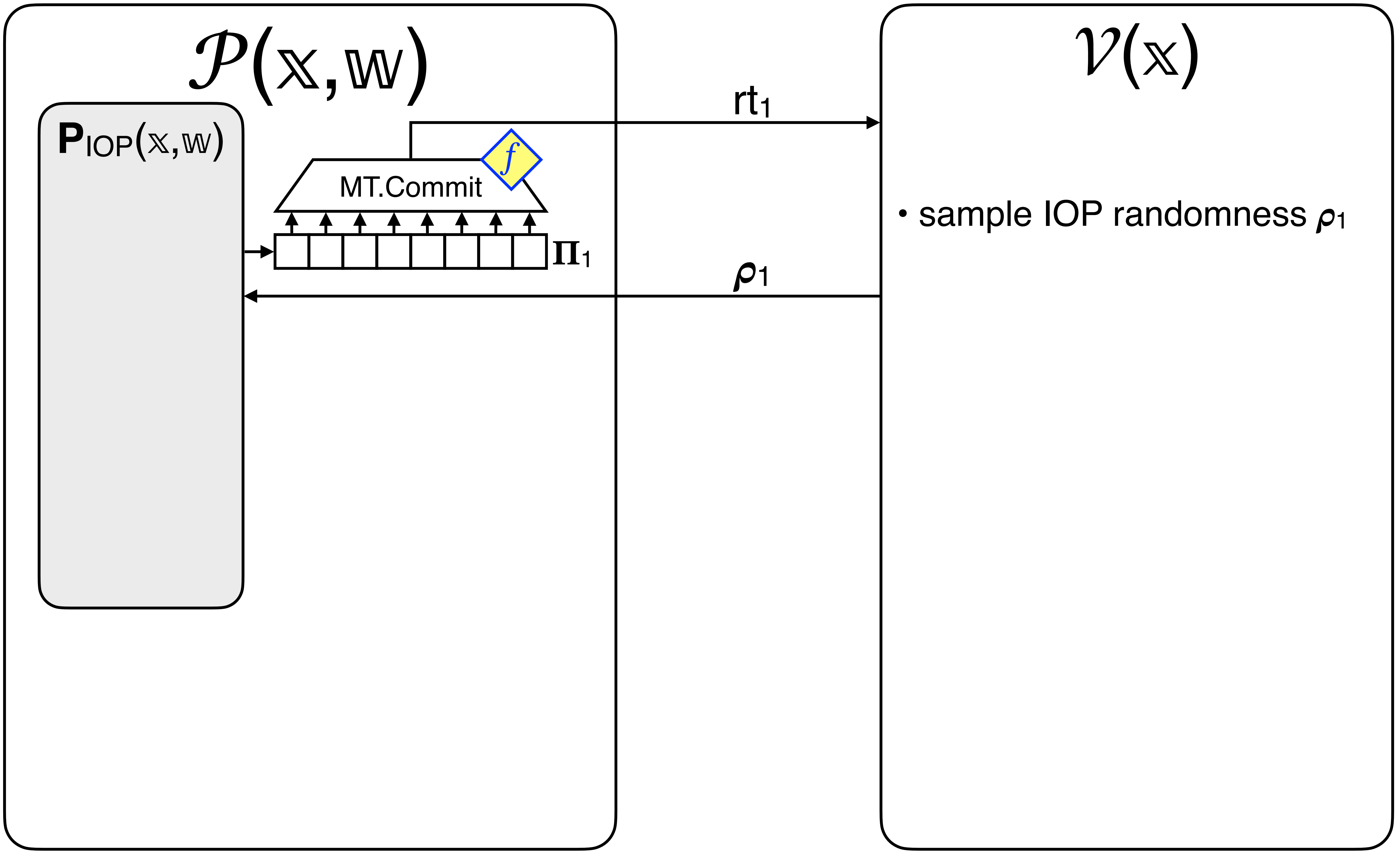
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



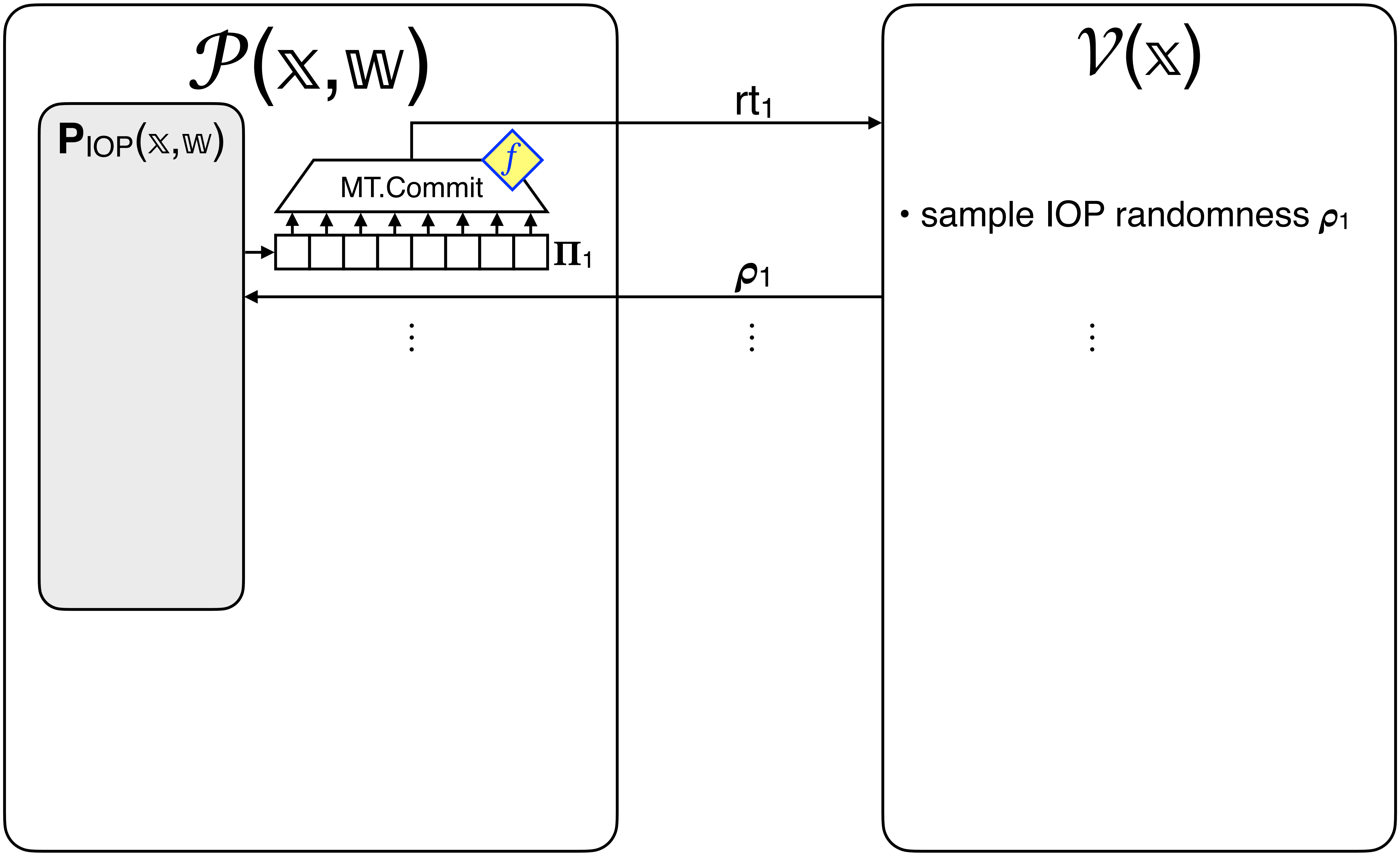
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



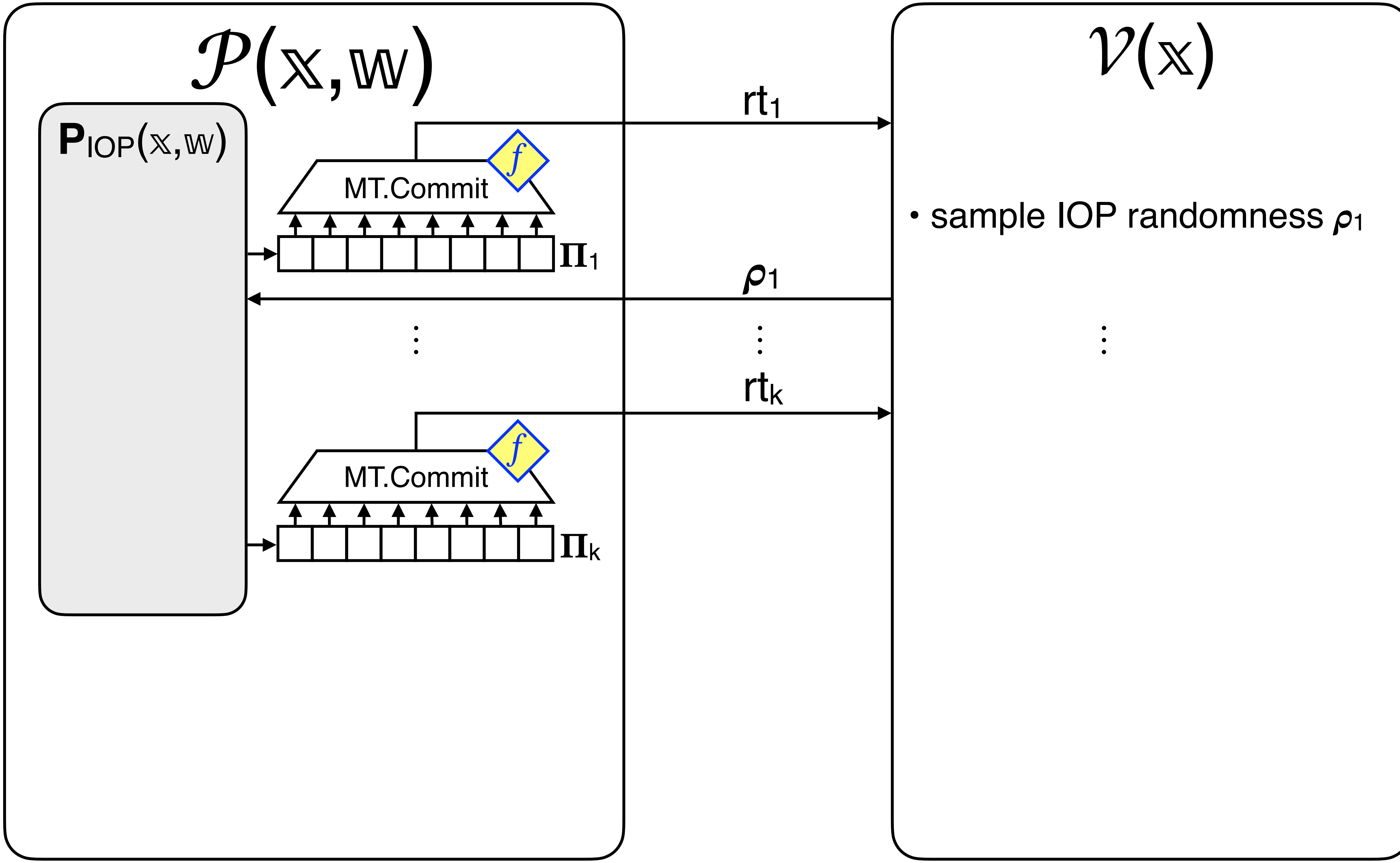
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



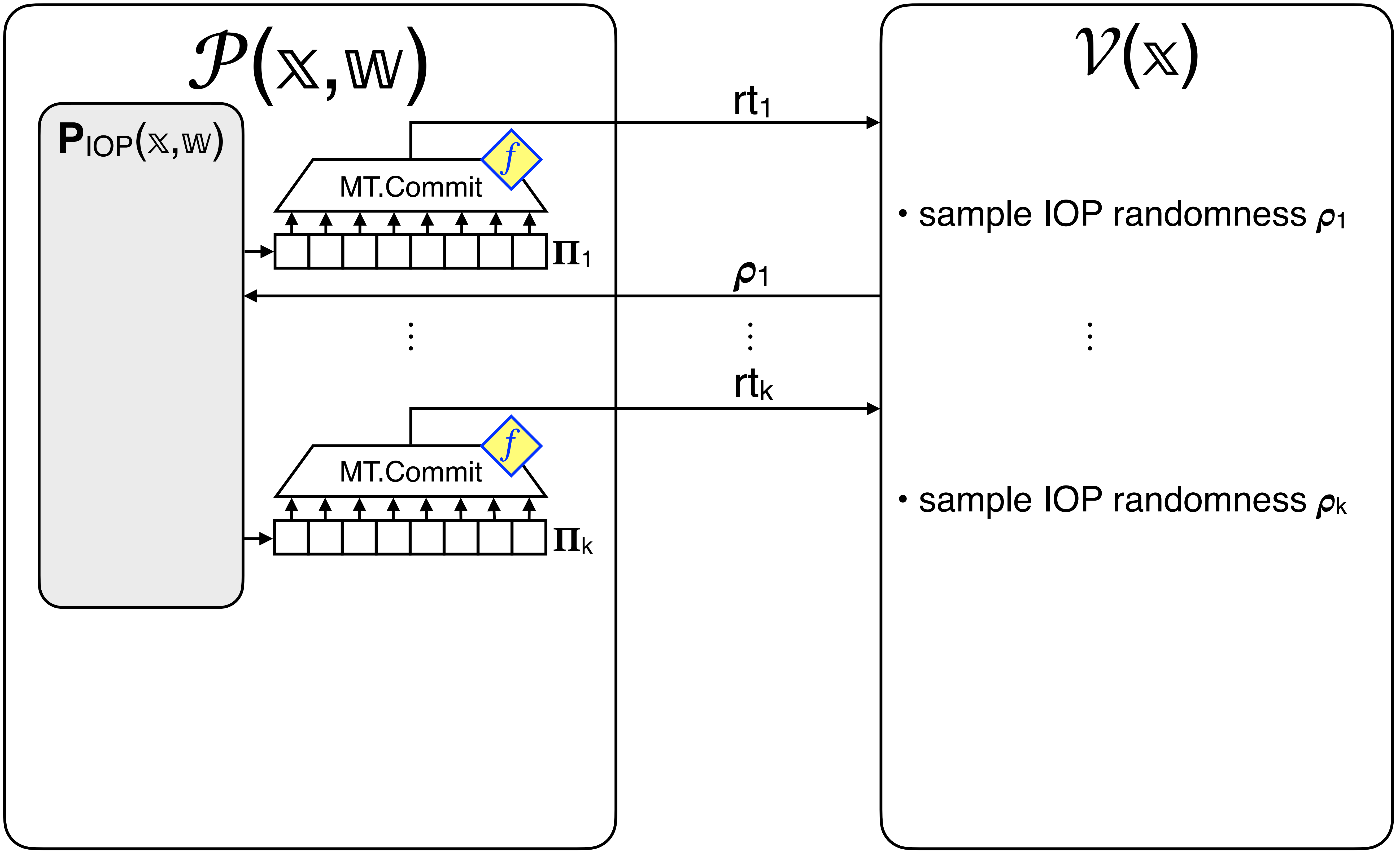
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



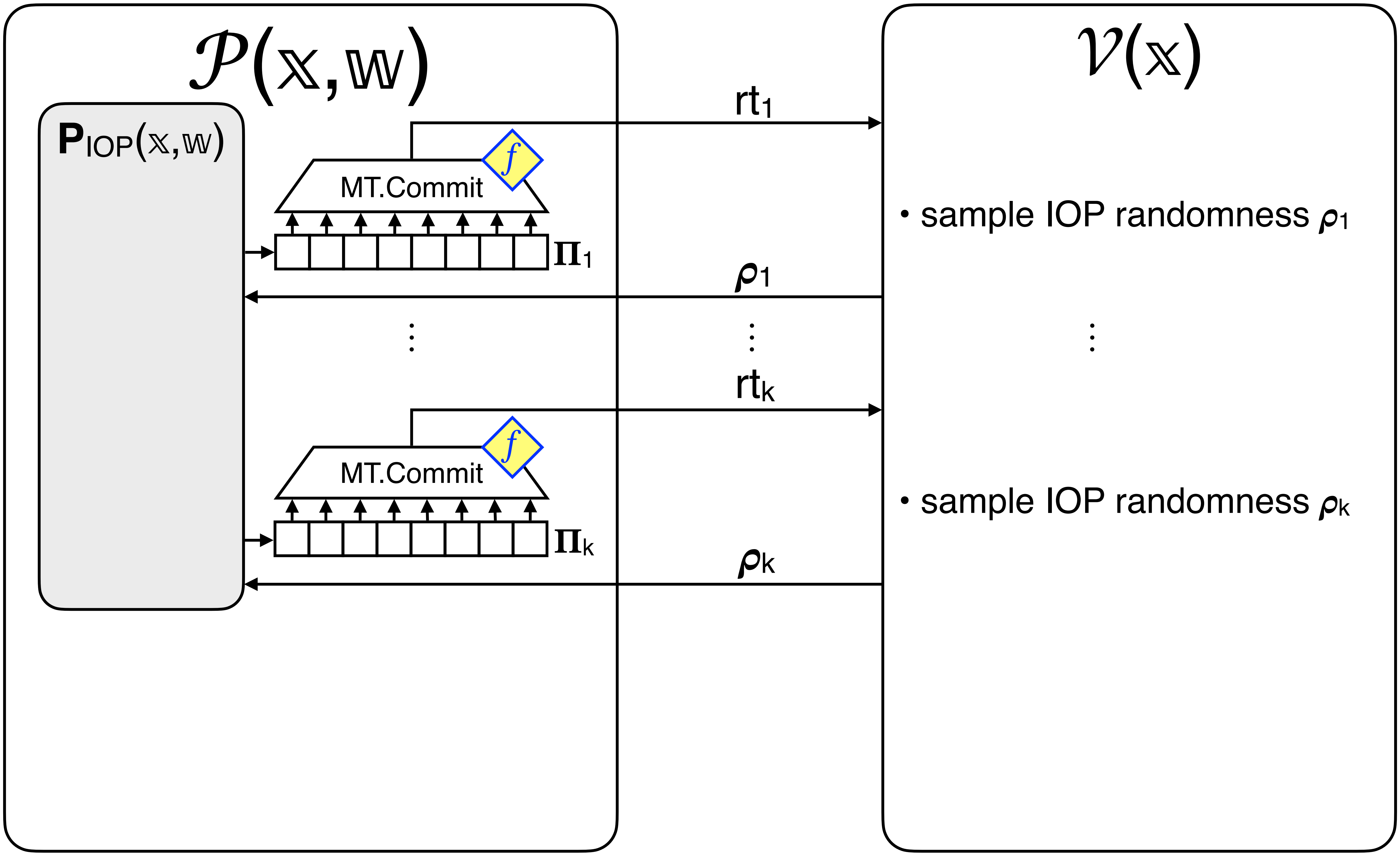
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



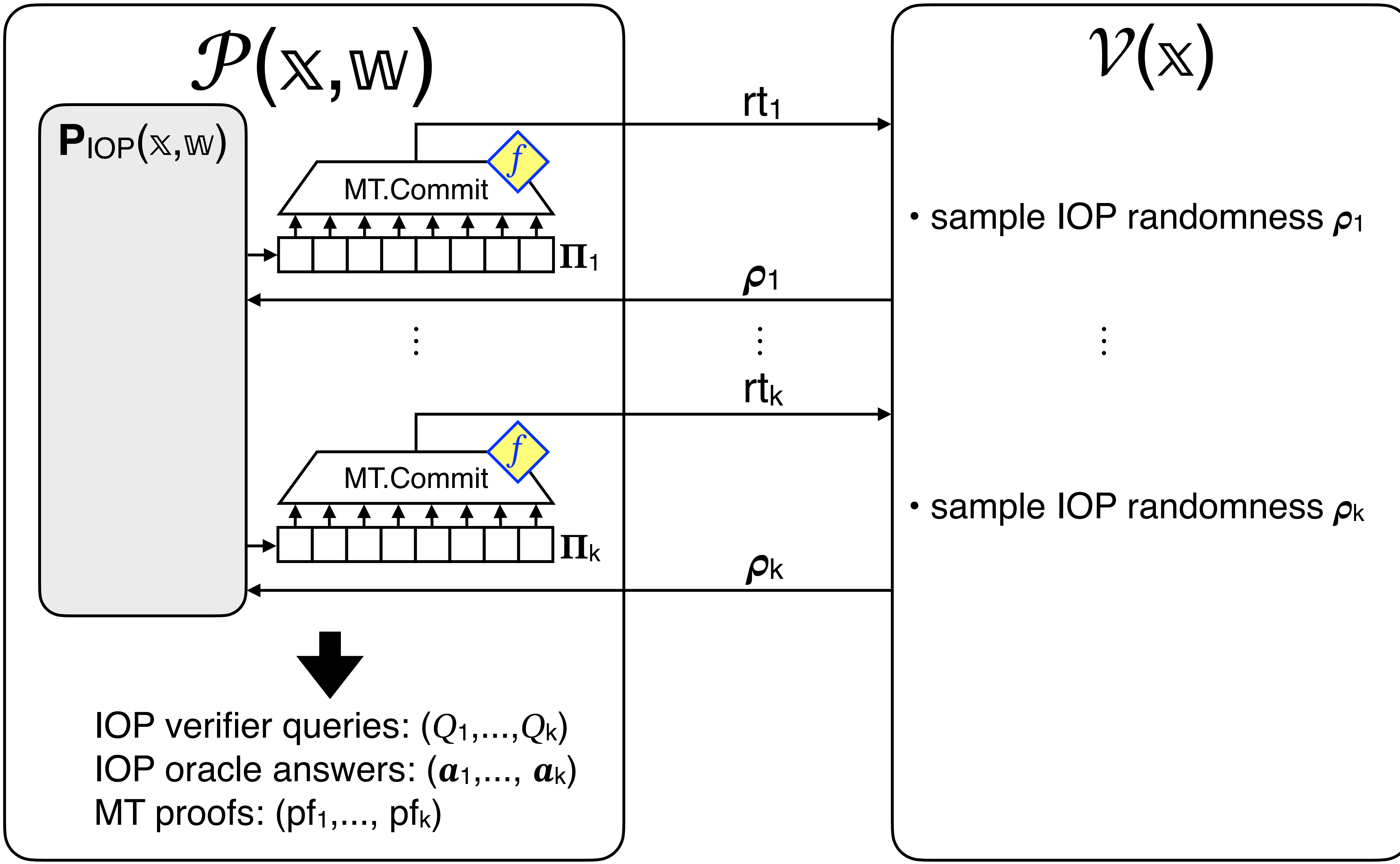
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



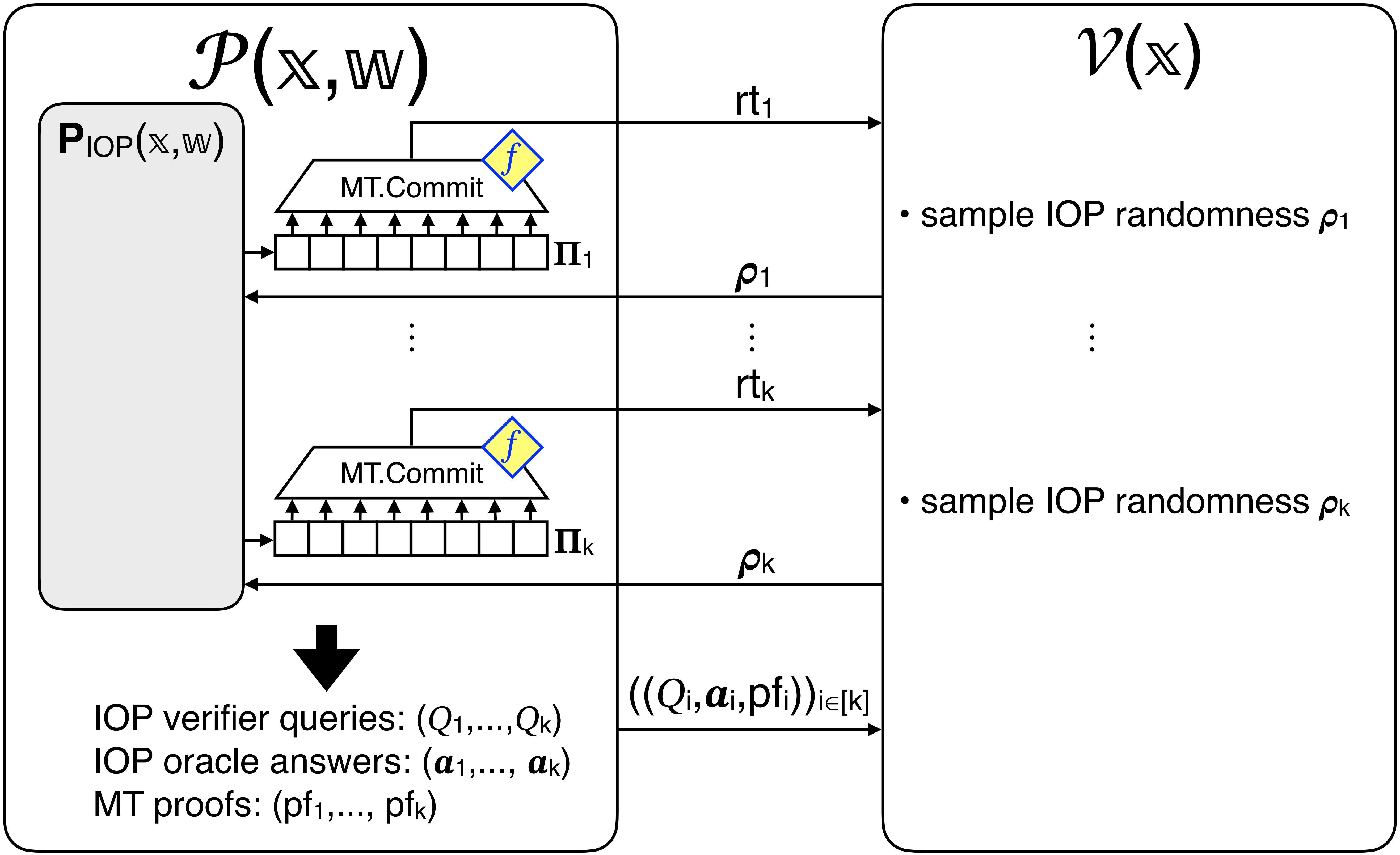
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



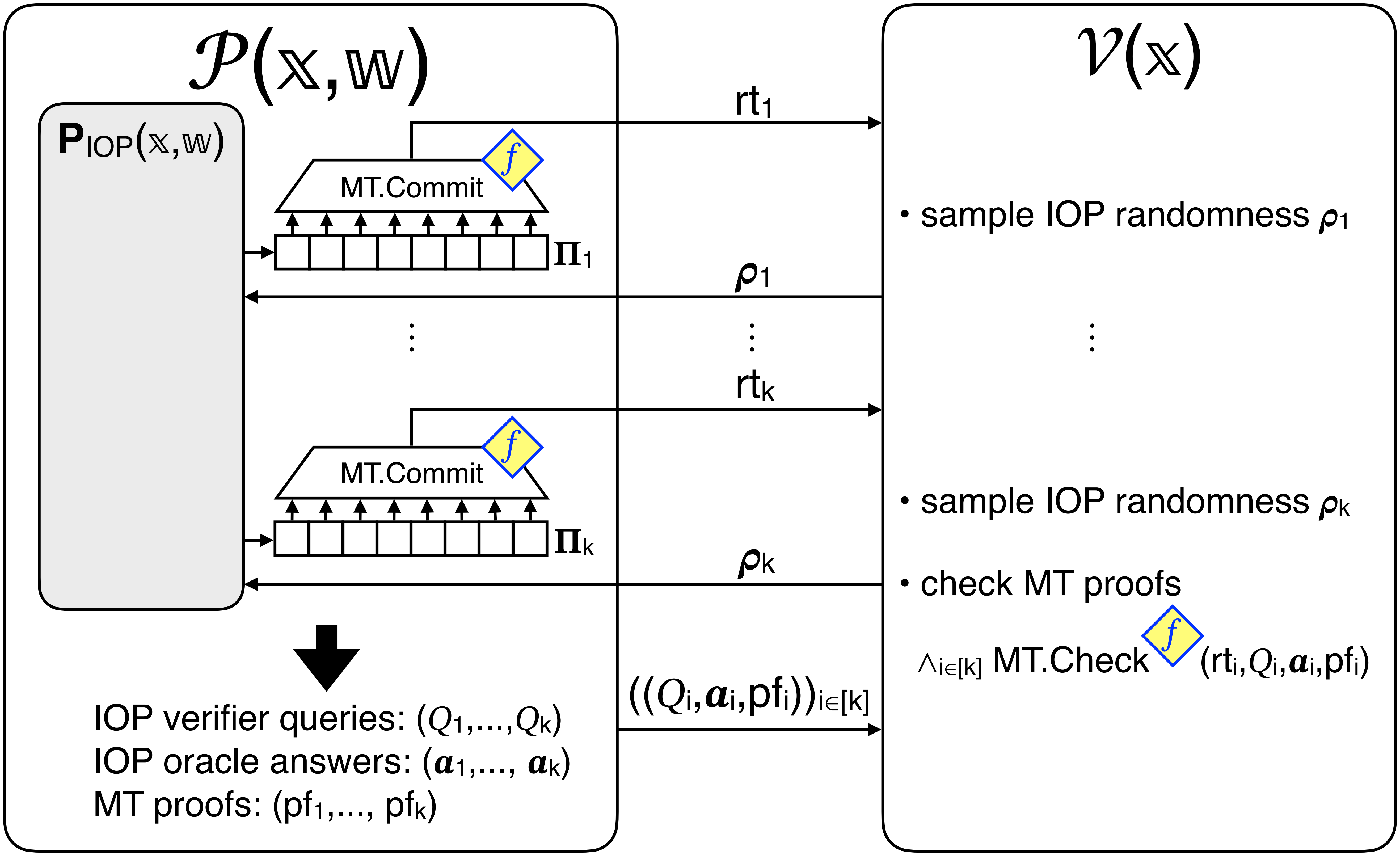
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



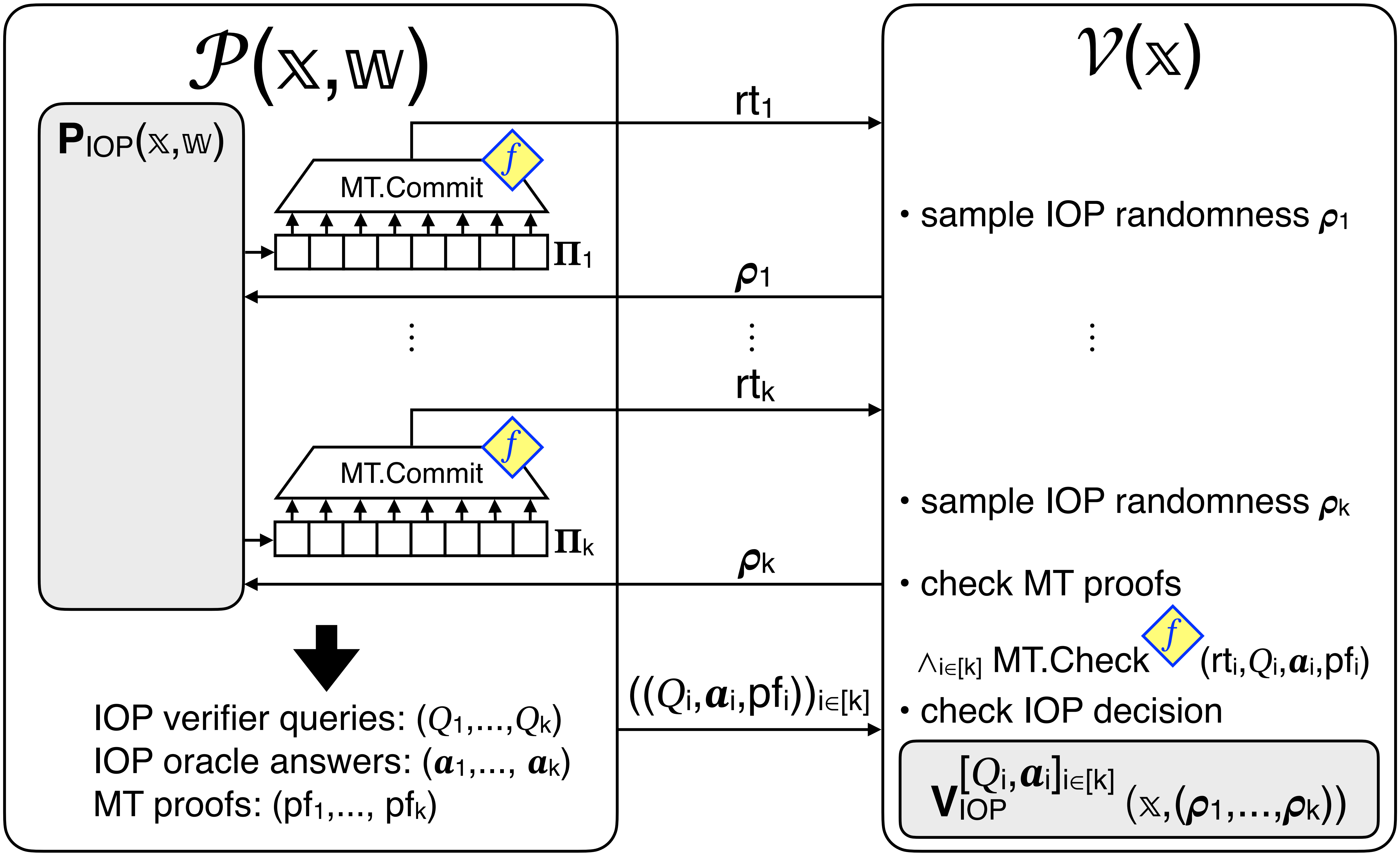
iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



iBCS Protocol

Idea: Use a Merkle Tree to commit to each IOP string.



iBCS Protocol

iBCS Protocol

IOP parameters:
- ε_{IOP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

iBCS Protocol

IOP parameters:
- $\varepsilon_{\mathrm{IOP}}$ is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

iBCS Protocol

IOP parameters:
- ε_{IOP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\text{time}(\mathcal{P}) = \text{time}(\mathbf{P}_{\text{IOP}}) + \ell \cdot \text{time}(f)$$

iBCS Protocol

IOP parameters:
- $\varepsilon_{\mathrm{IOP}}$ is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathbf{P}_{\mathrm{IOP}}) + \ell \cdot \mathrm{time}(f)$$

$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathbf{V}_{\mathrm{IOP}}) + q \cdot \log \ell \cdot \mathrm{time}(f)$$

iBCS Protocol

IOP parameters:
- ε_{IOP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathrm{P}_{\mathrm{IOP}}) + \boxed{\ell \cdot \mathrm{time}(f)}$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathrm{V}_{\mathrm{IOP}}) + \boxed{q \cdot \log \ell \cdot \mathrm{time}(f)}$$

small overheads

iBCS Protocol

IOP parameters:
- $\varepsilon_{\mathrm{IOP}}$ is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathbf{P}_{\mathrm{IOP}}) + \boxed{\ell \cdot \mathrm{time}(f)}$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathbf{V}_{\mathrm{IOP}}) + \boxed{q \cdot \log \ell \cdot \mathrm{time}(f)}$$

small overheads

Communication ≈ query complexity (rather than proof length):

iBCS Protocol

IOP parameters:
- $\varepsilon_{\mathrm{IOP}}$ is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathrm{P}_{\mathrm{IOP}}) + \boxed{\ell \cdot \mathrm{time}(f)}$$

$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathrm{V}_{\mathrm{IOP}}) + \boxed{q \cdot \log \ell \cdot \mathrm{time}(f)}$$

small overheads

Communication ≈ query complexity (rather than proof length):

$$|(\mathrm{rt}_i)_{i\in[k]}| + |((Q_i, \boldsymbol{a}_i, \mathrm{pf}_i))_{i\in[k]}| \approx q \cdot (\log|\Sigma| + 2\lambda \log \ell)$$

iBCS Protocol

IOP parameters:
- ε_{IOP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathrm{P}_{\mathrm{IOP}}) + \boxed{\ell \cdot \mathrm{time}(f)}$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathrm{V}_{\mathrm{IOP}}) + \boxed{q \cdot \log\ell \cdot \mathrm{time}(f)}$$

small overheads

Communication ≈ query complexity (rather than proof length):

$$|(\mathrm{rt}_i)_{i\in[k]}| + |((Q_i,\boldsymbol{a}_i,\mathrm{pf}_i))_{i\in[k]}| \approx q \cdot (\ \log|\Sigma| \ + 2\lambda \log\ell\ )$$

Security:

iBCS Protocol

IOP parameters:
- $\varepsilon_{\mathrm{IOP}}$ is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(\mathrm{P_{IOP}}) + \ell \cdot \mathrm{time}(f)$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(\mathrm{V_{IOP}}) + q \cdot \log\ell \cdot \mathrm{time}(f)$$

small overheads

Communication ≈ query complexity (rather than proof length):

$$|(\mathrm{rt}_i)_{i\in[k]}| + |((Q_i, \boldsymbol{a}_i, \mathrm{pf}_i))_{i\in[k]}| \approx q \cdot (\log|\Sigma| + 2\lambda \log\ell)$$

Security:

Theorem: ∀ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks iBCS}] \leq \epsilon_{\mathrm{IOP}} + \frac{t^2}{2^\lambda}$$

iBCS Protocol

IOP parameters:
- ε_{IOP} is soundness
- Σ is proof alphabet
- ℓ is proof length
- q is number of queries

Proving time and verification time are preserved:

$$\mathrm{time}(\mathcal{P}) = \mathrm{time}(P_{IOP}) + \boxed{\ell \cdot \mathrm{time}(f)}$$
$$\mathrm{time}(\mathcal{V}) = \mathrm{time}(V_{IOP}) + \boxed{q \cdot \log \ell \cdot \mathrm{time}(f)}$$

small overheads

Communication ≈ query complexity (rather than proof length):

$$|(\mathrm{rt}_i)_{i\in[k]}| + |((Q_i, \boldsymbol{a}_i, \mathrm{pf}_i))_{i\in[k]}| \approx q \cdot (\log|\Sigma| + 2\lambda \log \ell)$$

Security:

Theorem: ∀ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks iBCS}] \leq \epsilon_{\mathrm{IOP}} + \frac{t^2}{2^\lambda}$$

Example:

Set $\varepsilon_{IOP} = 1/4$ and $\lambda = 256+2 = 258$.

Every 2^{128}-query adversary breaks iBCS w.p. $\leq 1/4 + 1/4 = 1/2$.

BCS Protocol

BCS Protocol

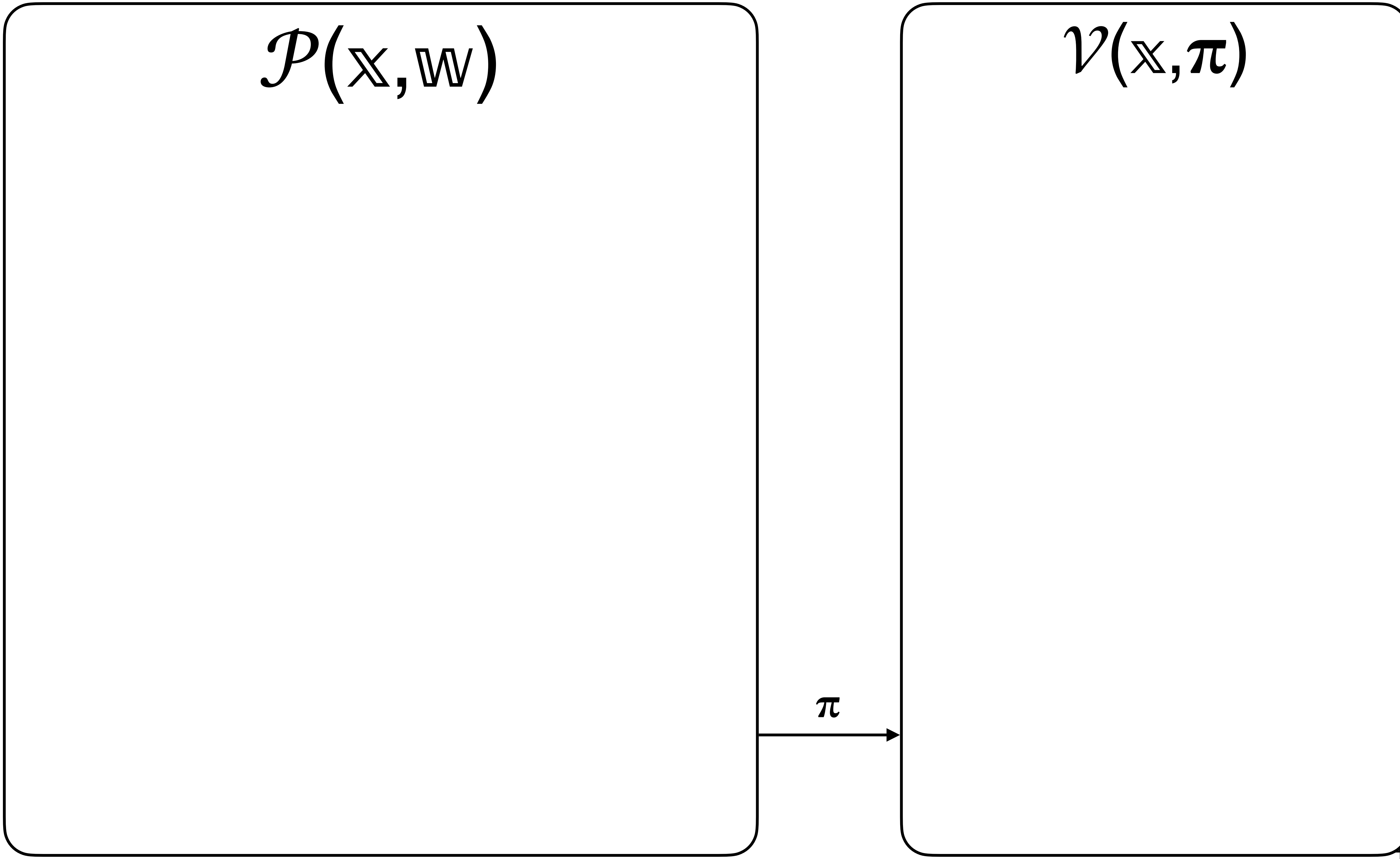
Idea: Use the random oracle to derive each IOP randomness.

(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)

BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

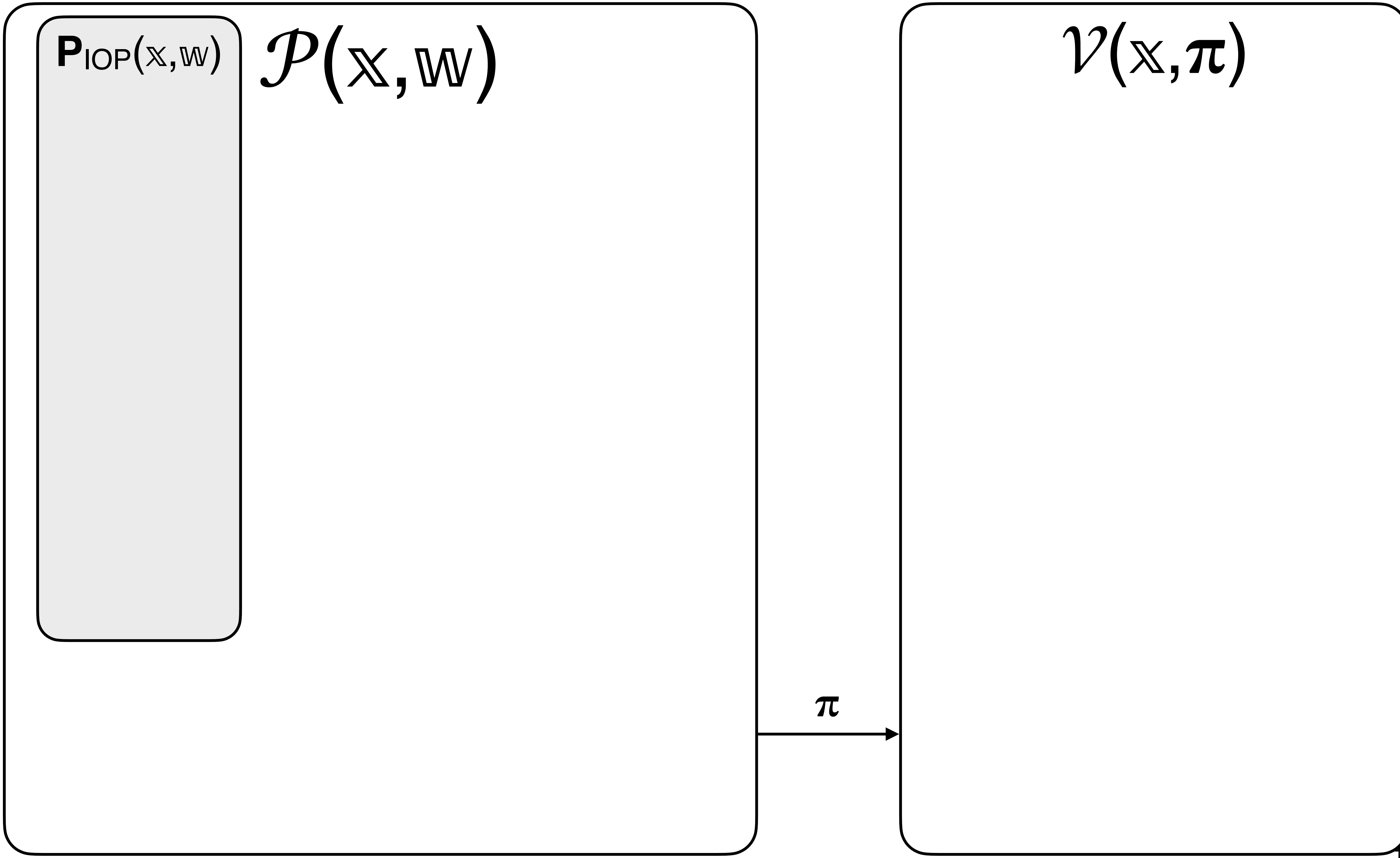
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

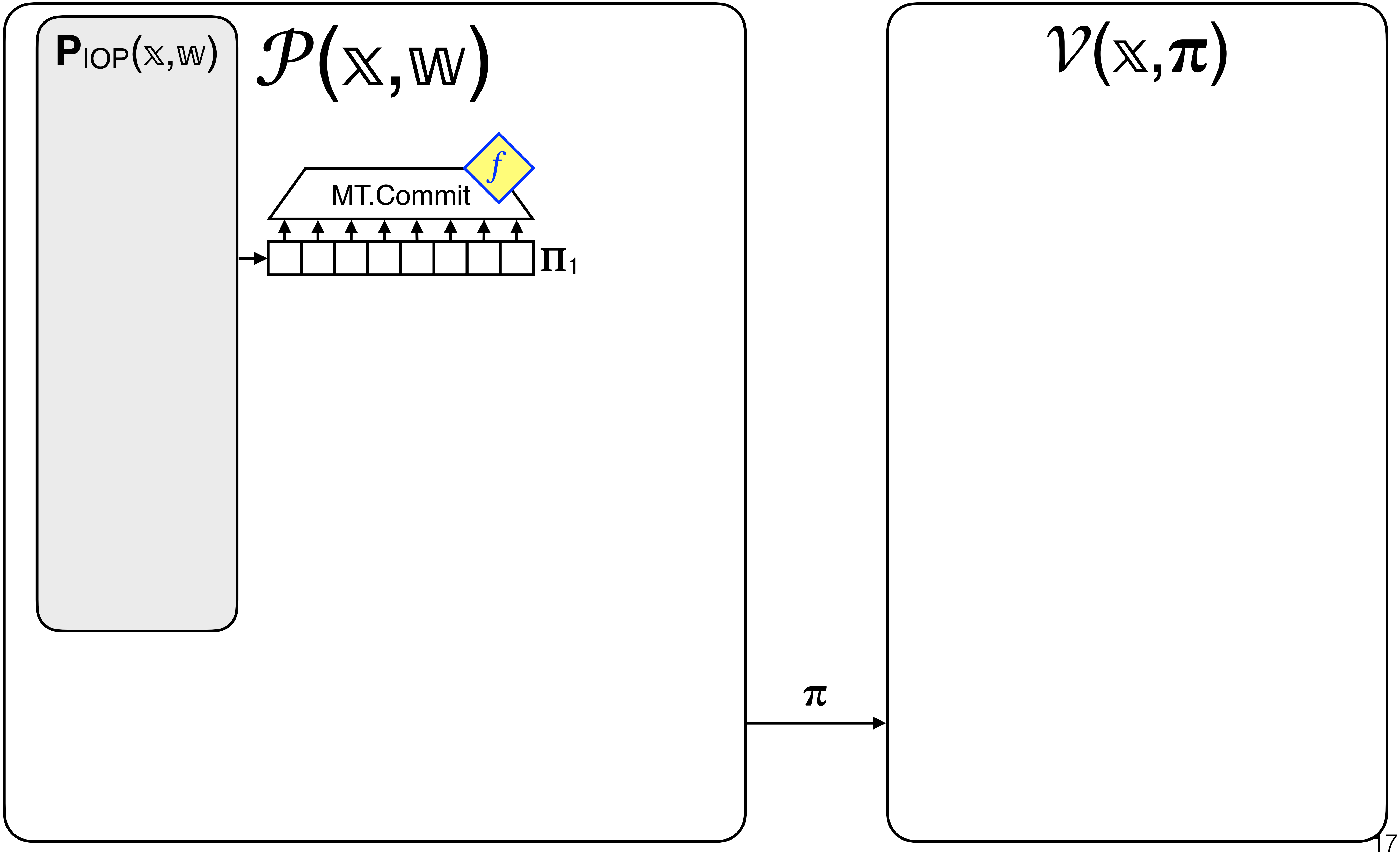
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

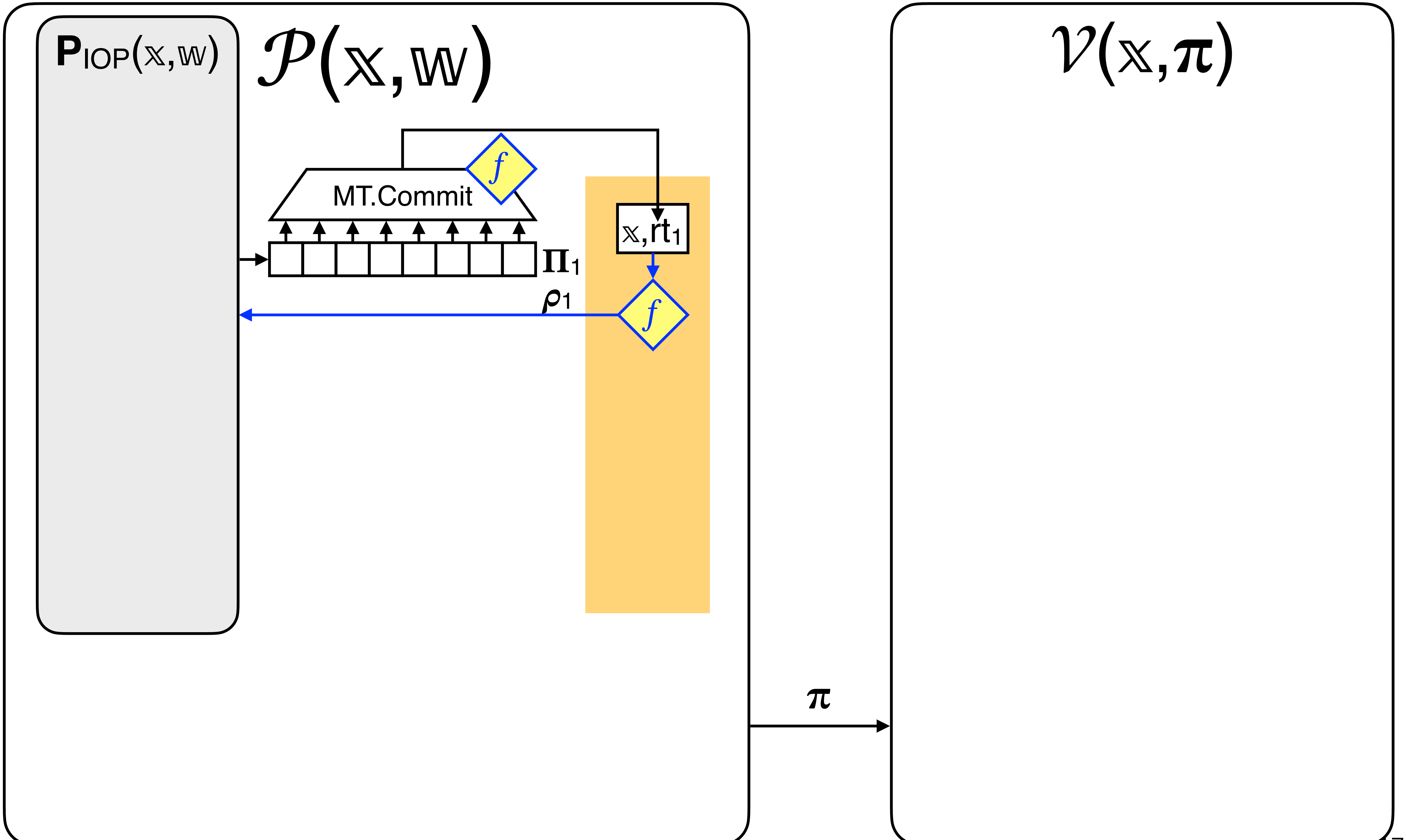
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

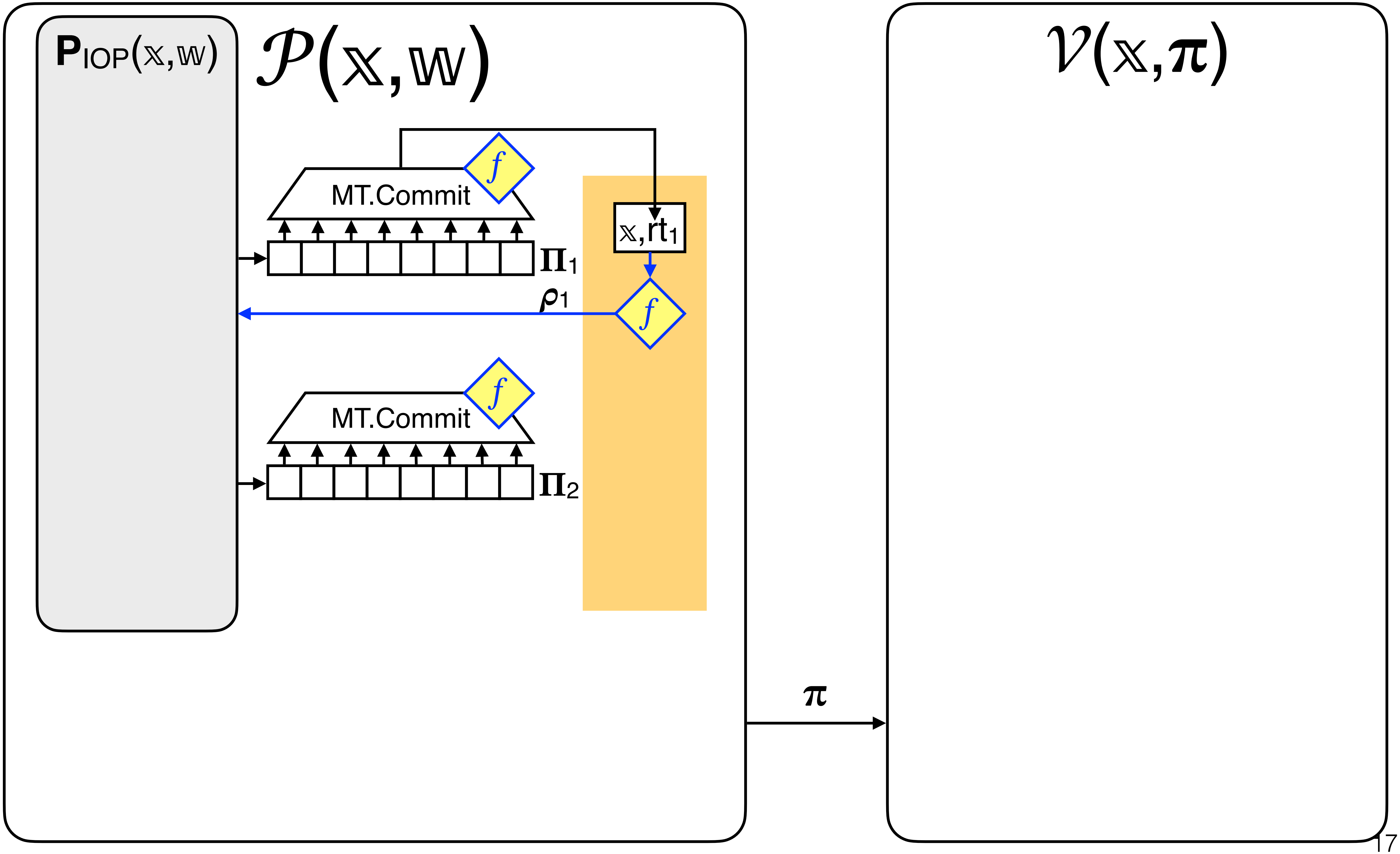
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

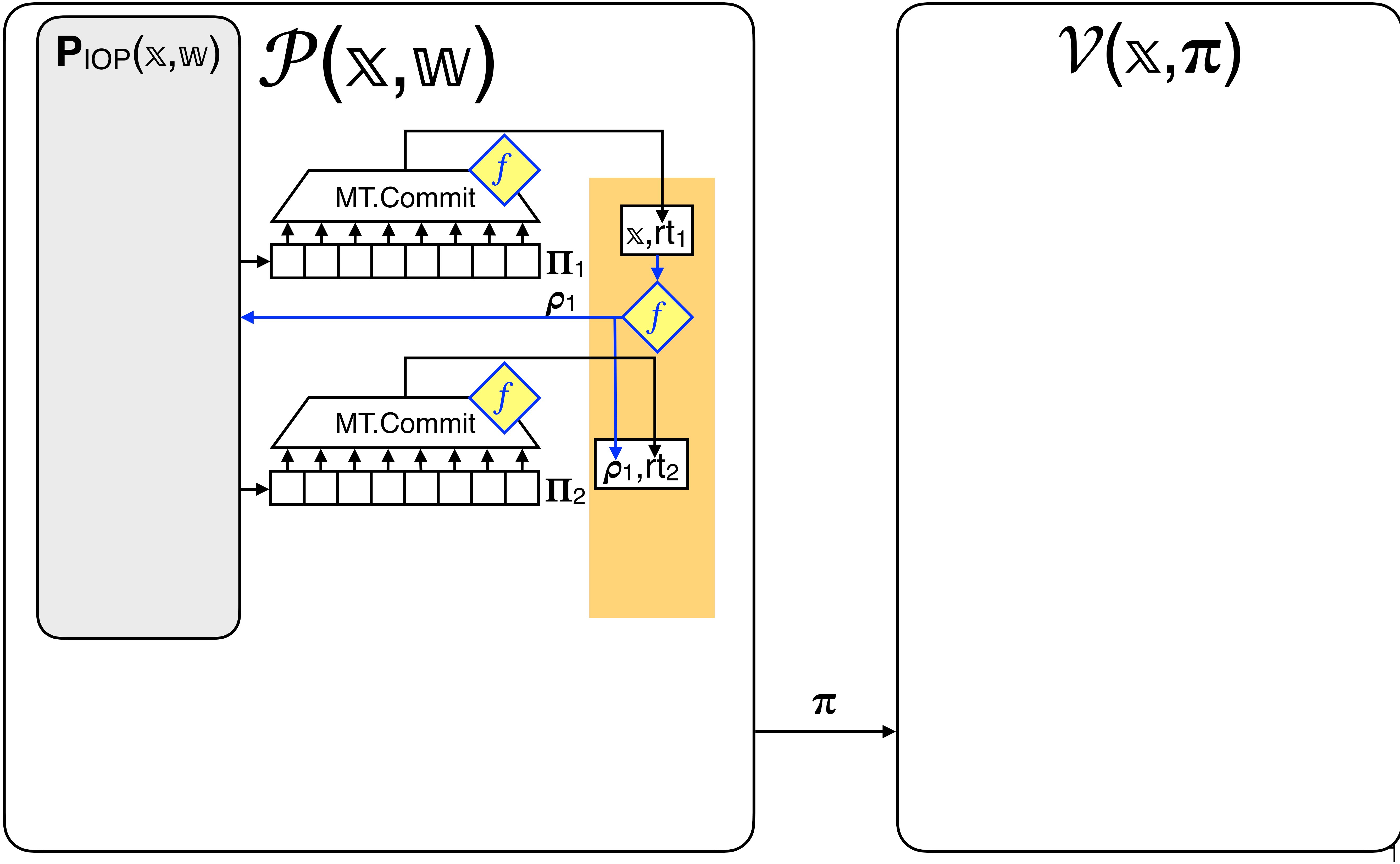
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

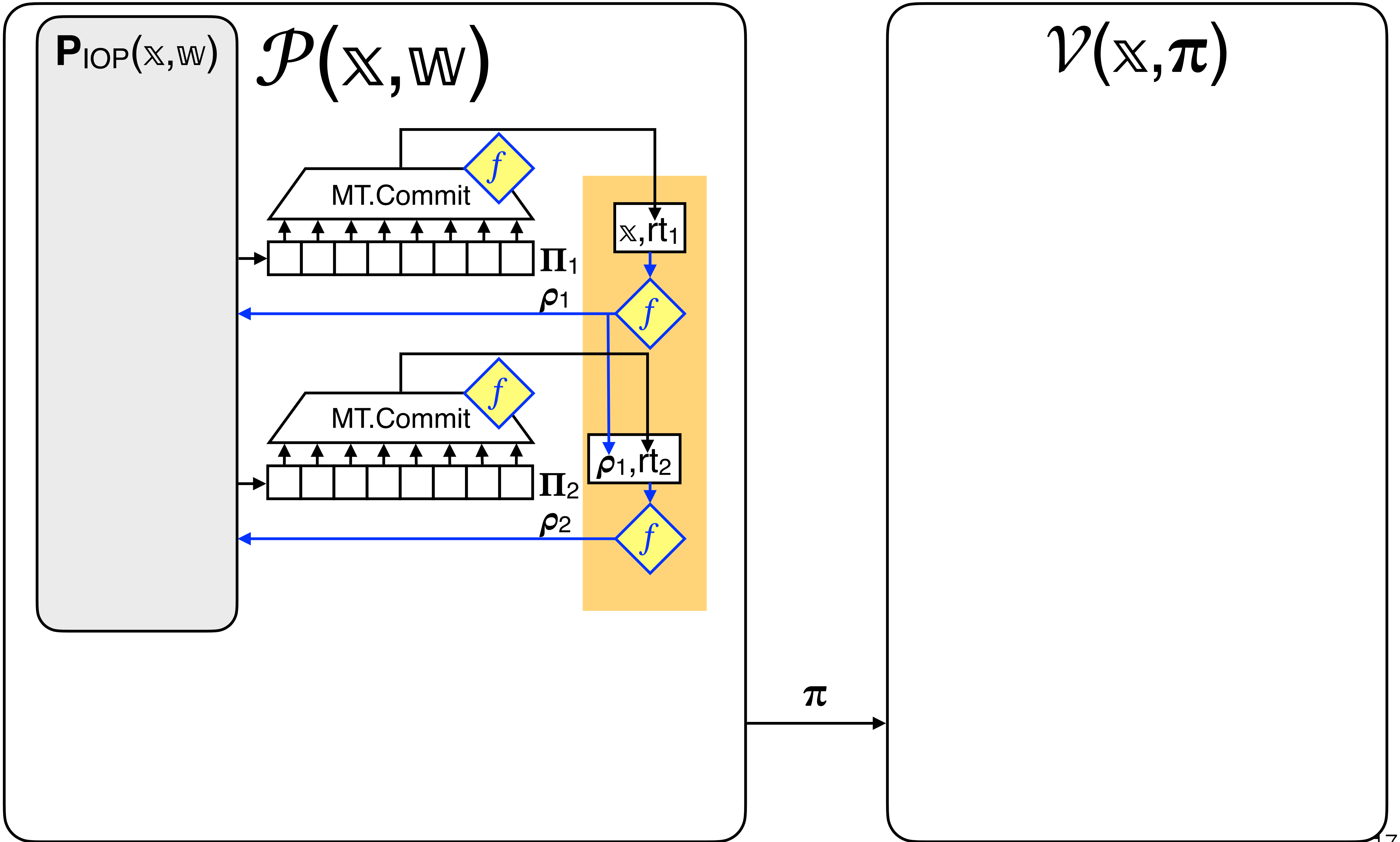
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

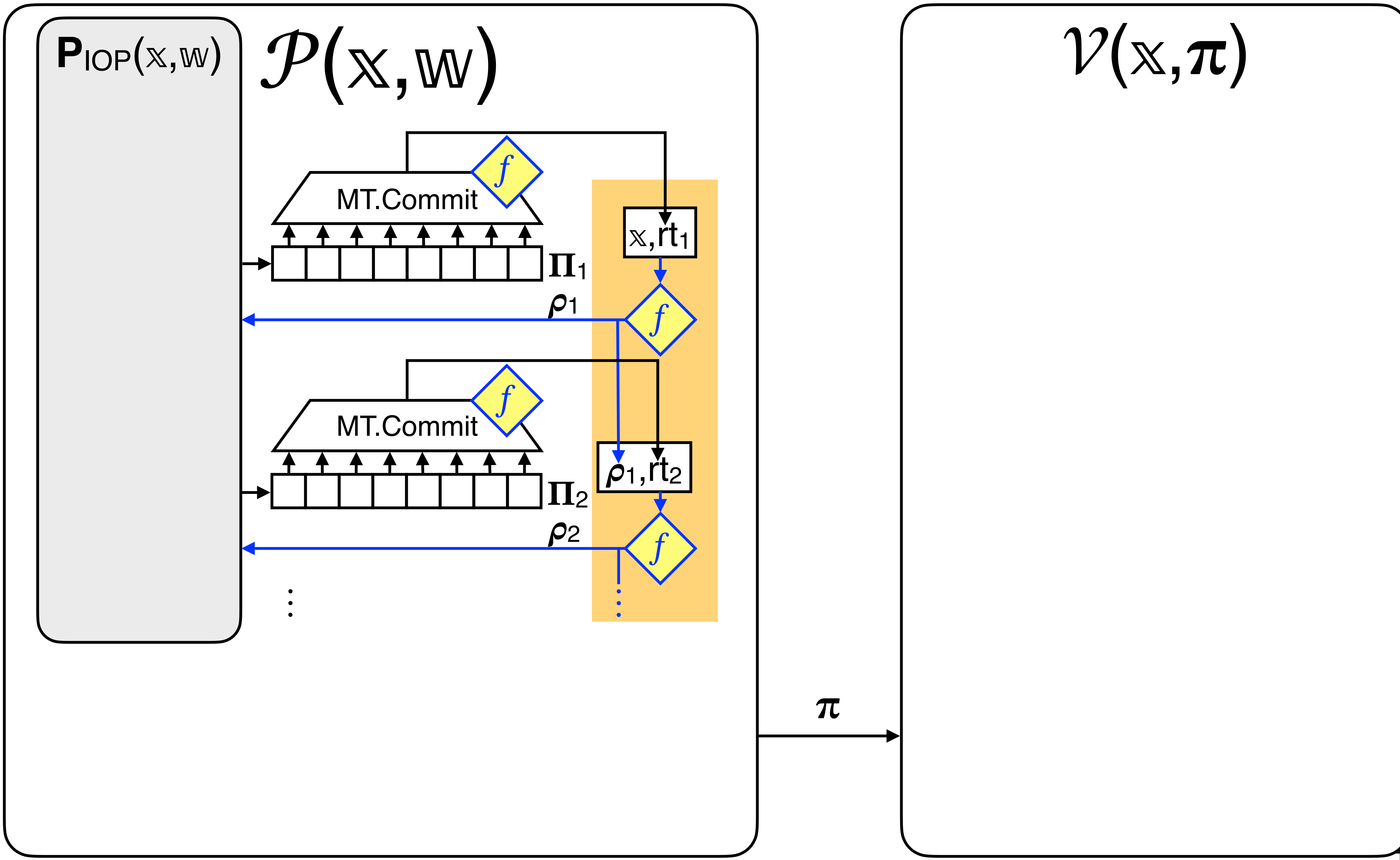
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

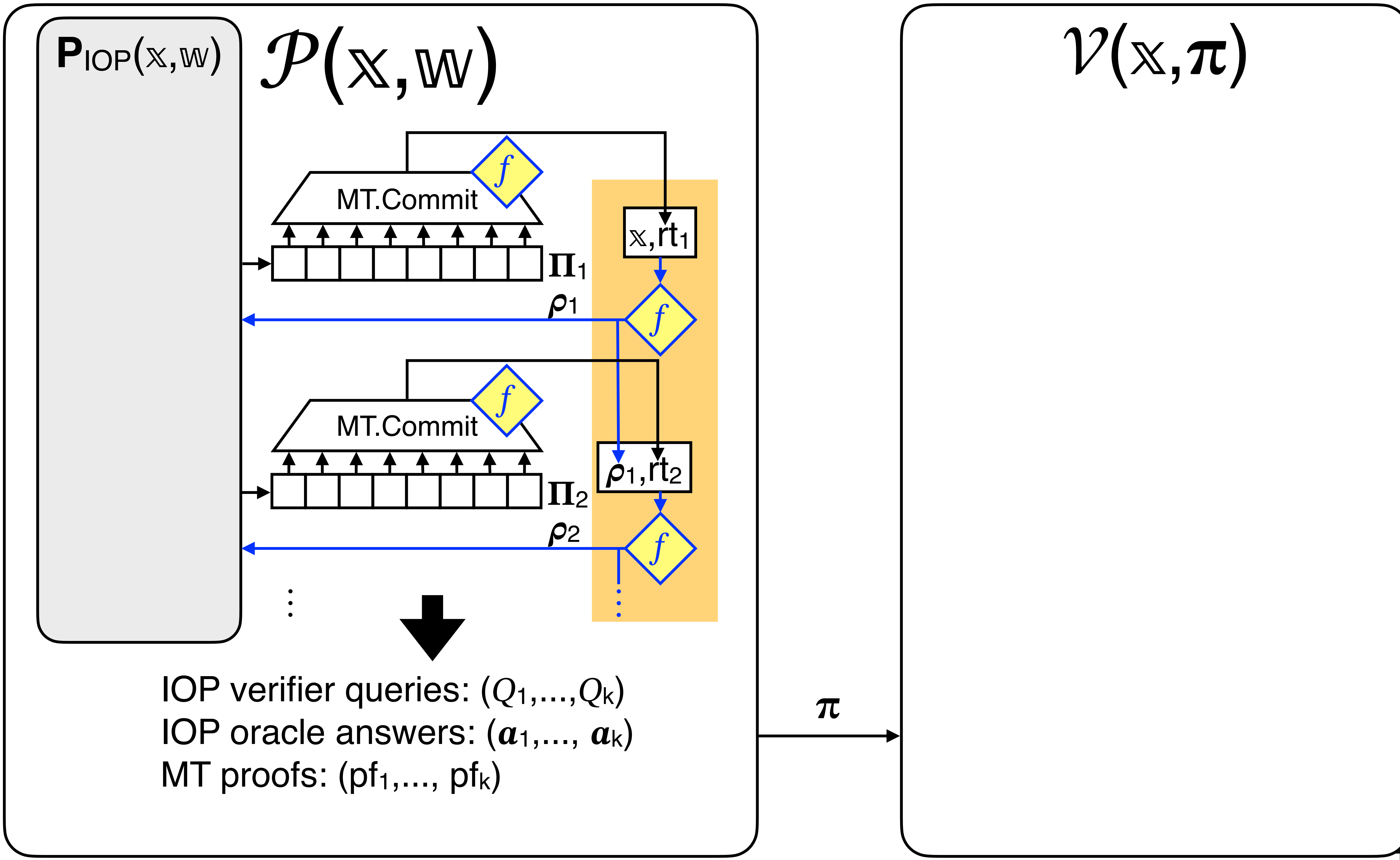
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

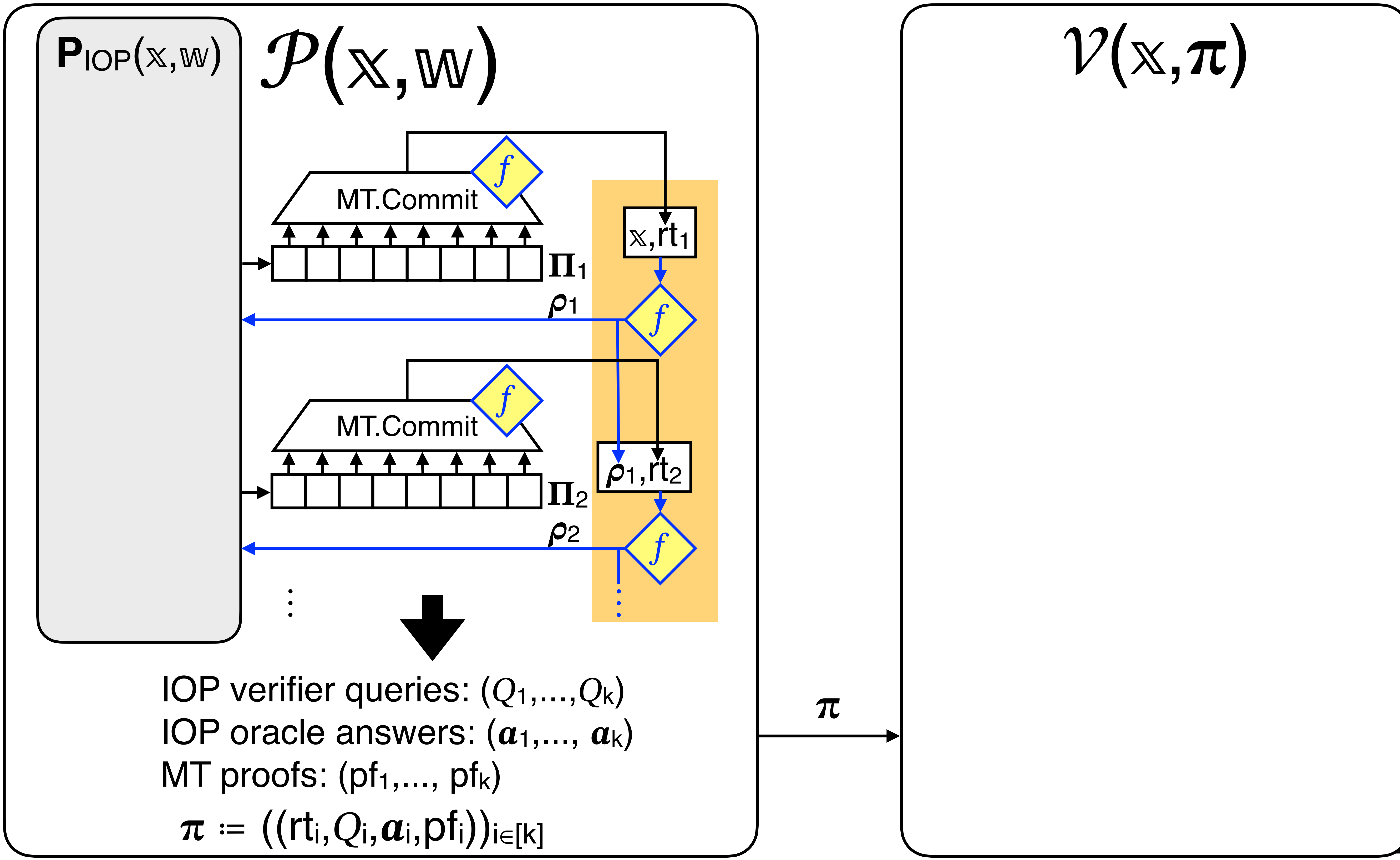
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)

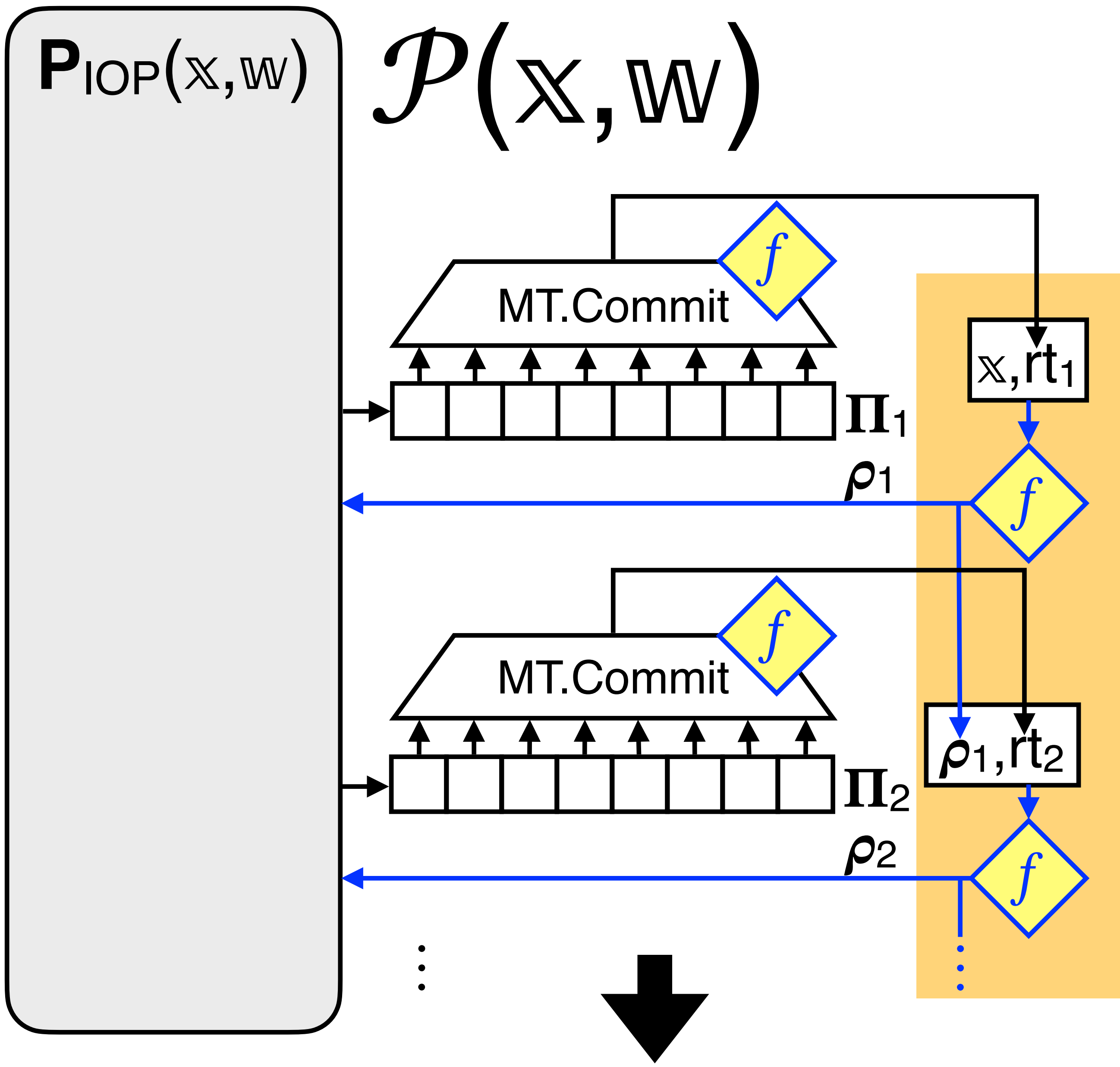


BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)

$\mathcal{P}(\mathbb{x},\mathbb{w})$



IOP verifier queries: $(Q_1,\ldots,Q_k)$

IOP oracle answers: $(\boldsymbol{a}_1,\ldots,\boldsymbol{a}_k)$

MT proofs: $(\mathrm{pf}_1,\ldots,\mathrm{pf}_k)$

$\boldsymbol{\pi} := ((\mathrm{rt}_i,Q_i,\boldsymbol{a}_i,\mathrm{pf}_i))_{i\in[k]}$

$\boldsymbol{\pi}$

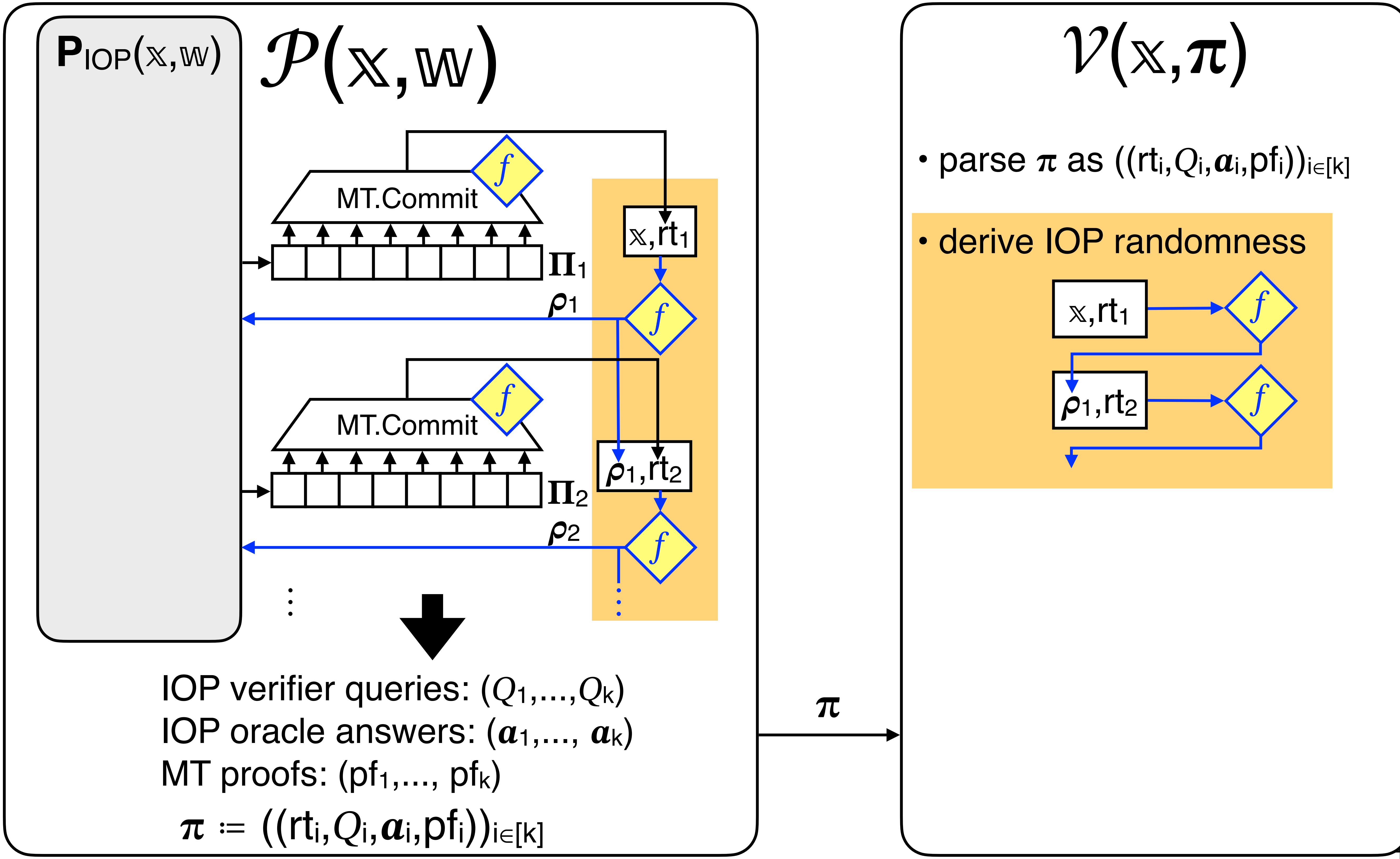
$\mathcal{V}(\mathbb{x},\boldsymbol{\pi})$

- parse $\boldsymbol{\pi}$ as $((\mathrm{rt}_i,Q_i,\boldsymbol{a}_i,\mathrm{pf}_i))_{i\in[k]}$

BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

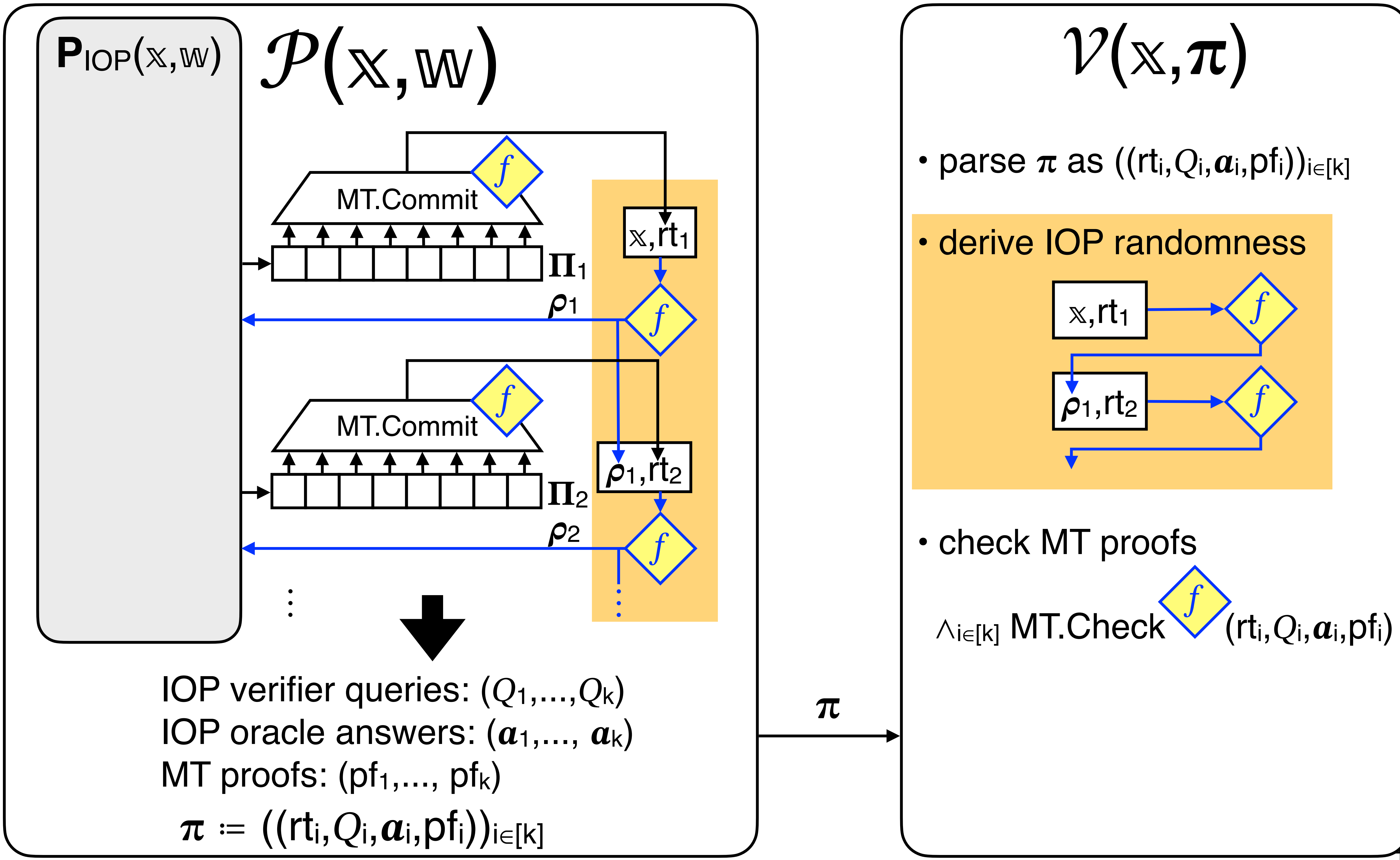
(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)



BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)

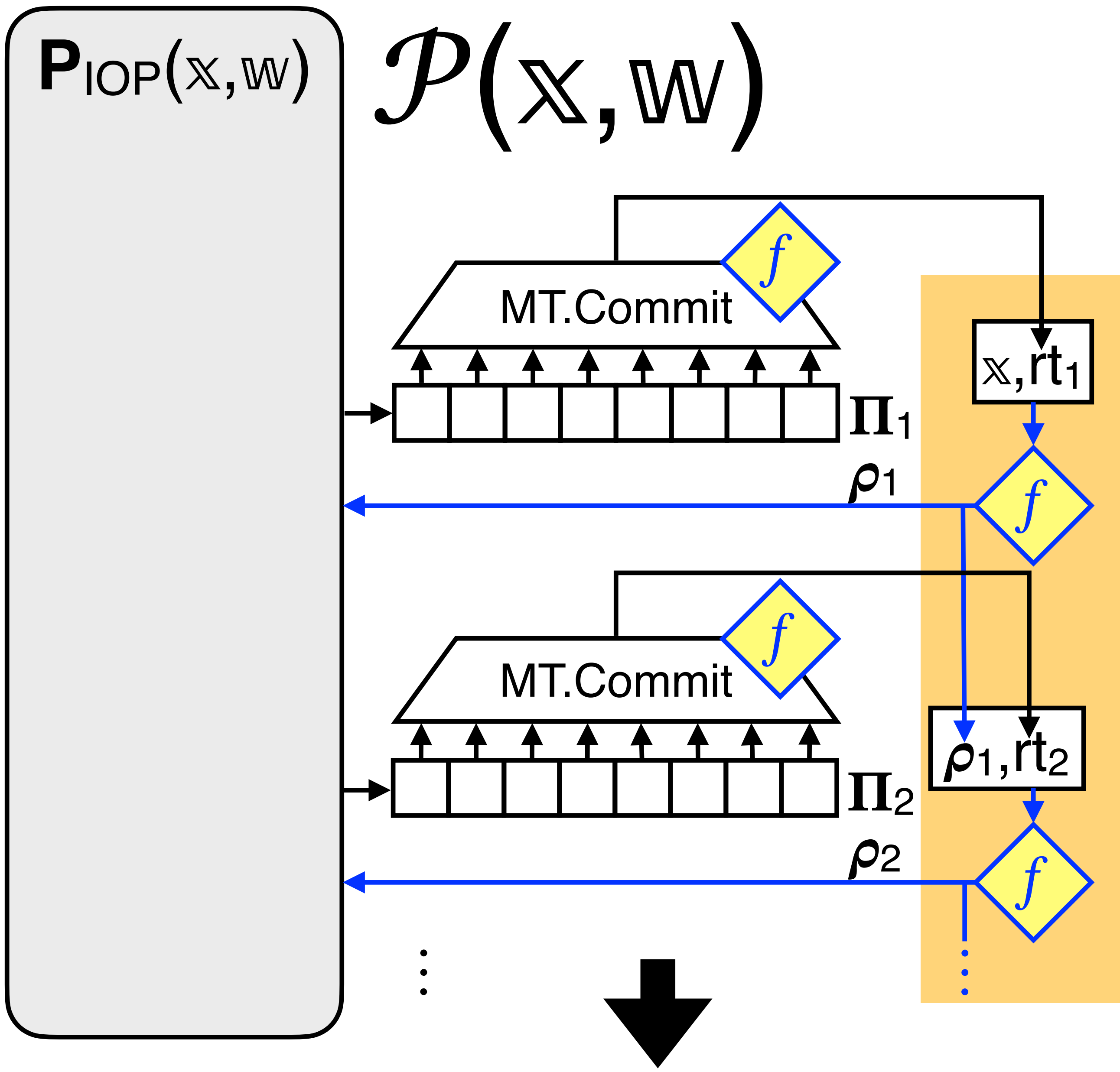


BCS Protocol

Idea: Use the random oracle to derive each IOP randomness.

(For cryptographers: apply the Fiat–Shamir transformation to the iBCS protocol.)

$\mathcal{P}(\mathbb{x},\mathbb{w})$



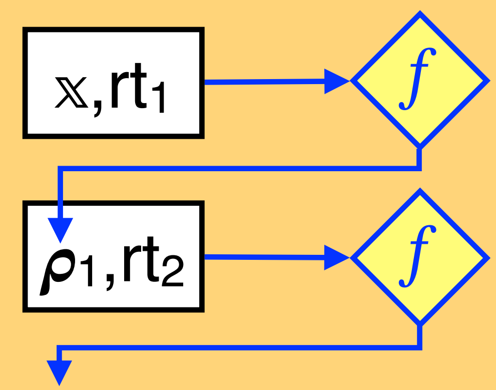
IOP verifier queries: $(Q_1,\ldots,Q_k)$
IOP oracle answers: $(\boldsymbol{a}_1,\ldots,\boldsymbol{a}_k)$
MT proofs: (pf$_1$,..., pf$_k$)

$\pi :=$ ((rt$_i$,Q_i,$\boldsymbol{a}_i$,pf$_i$))$_{i\in[k]}$

π

$\mathcal{V}(\mathbb{x},\pi)$

- parse π as ((rt$_i$,Q_i,$\boldsymbol{a}_i$,pf$_i$))$_{i\in[k]}$
- derive IOP randomness



- check MT proofs

$\wedge_{i\in[k]}$ MT.Check^{f}(rt$_i$,Q_i,$\boldsymbol{a}_i$,pf$_i$)

- check IOP decision

$$V_{IOP}^{[Q_i,\boldsymbol{a}_i]_{i\in[k]}}(\mathbb{x},(\rho_1,\ldots,\rho_k))$$

BCS Protocol

BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.

BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

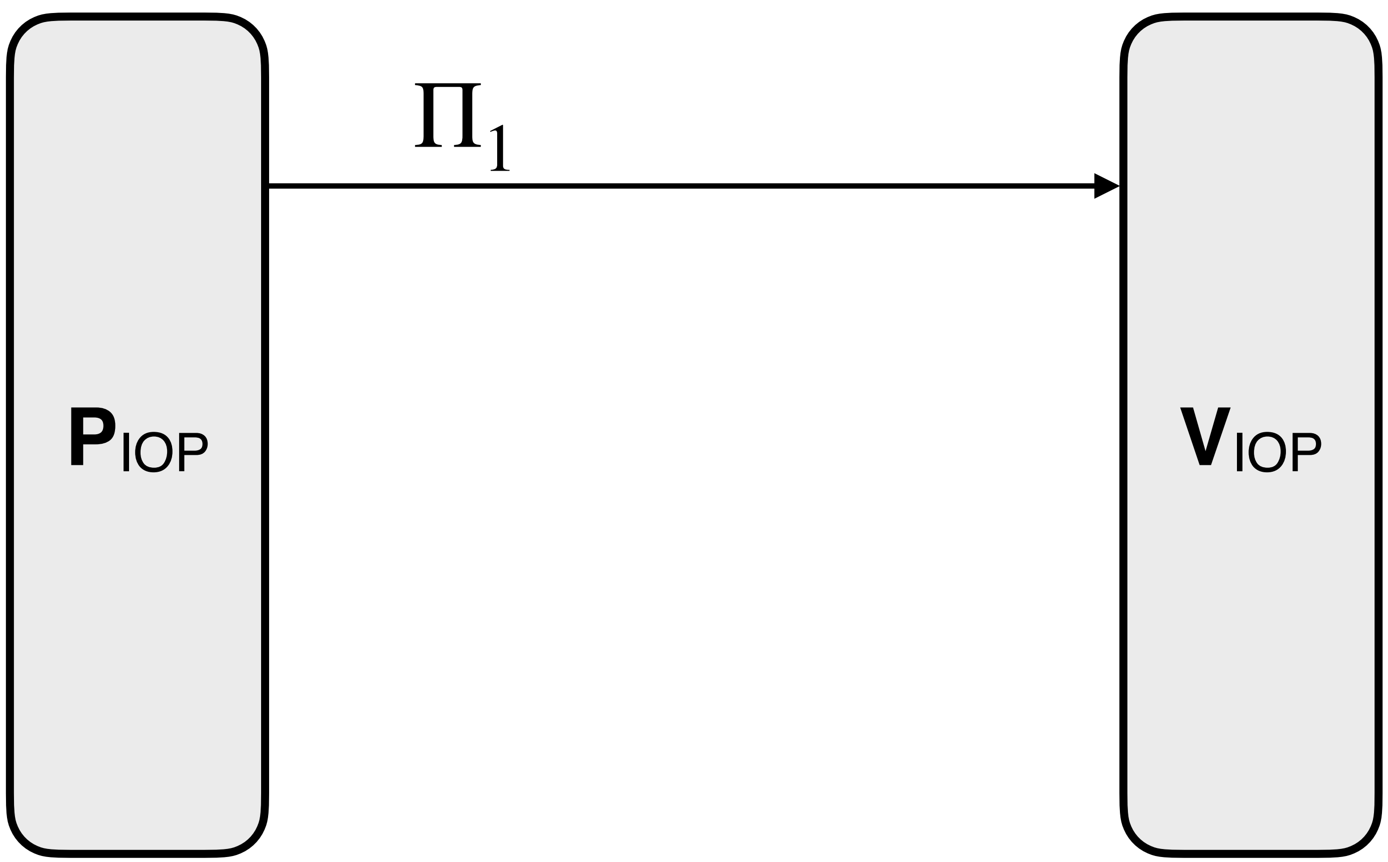
An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.

BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.

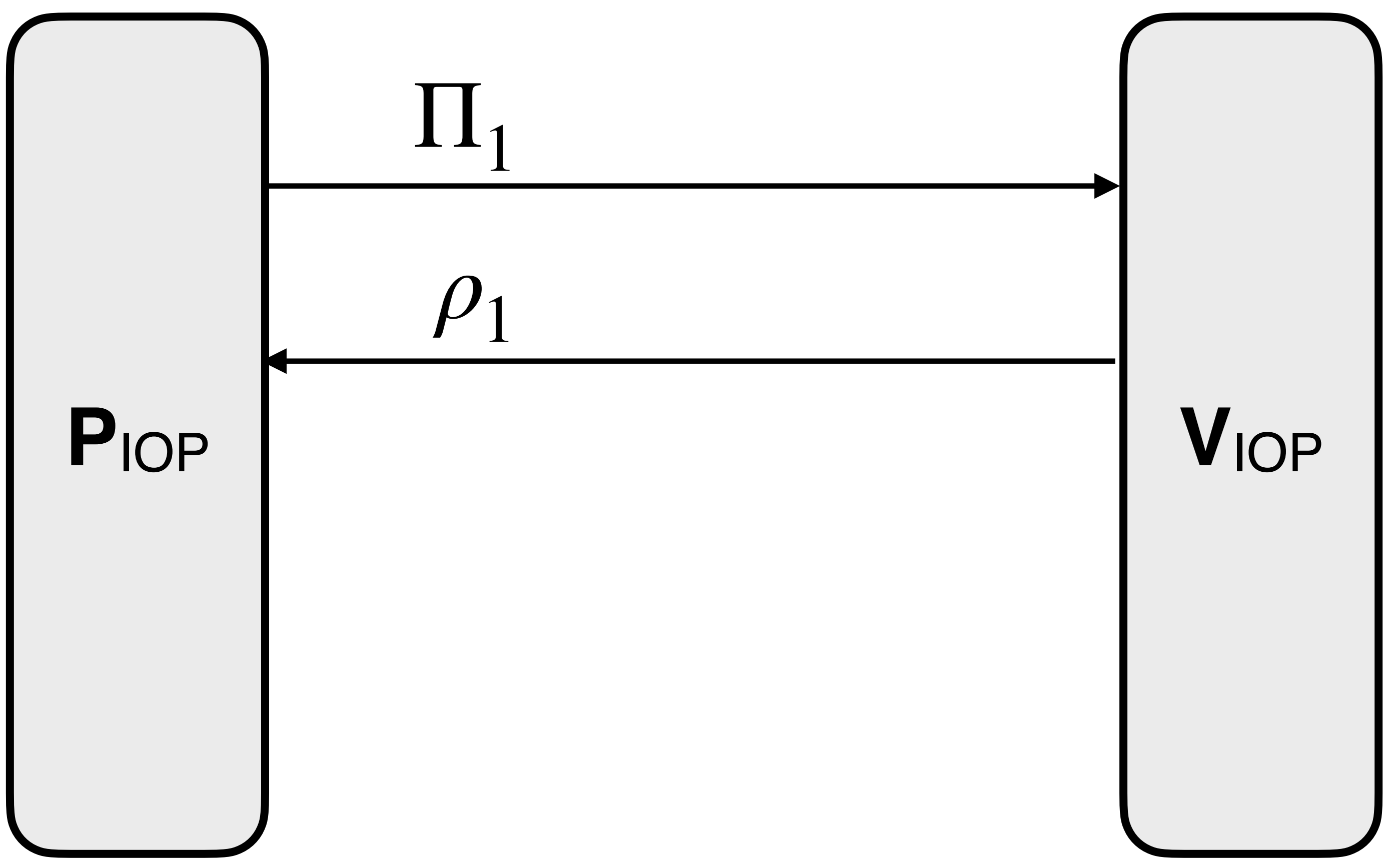


BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.

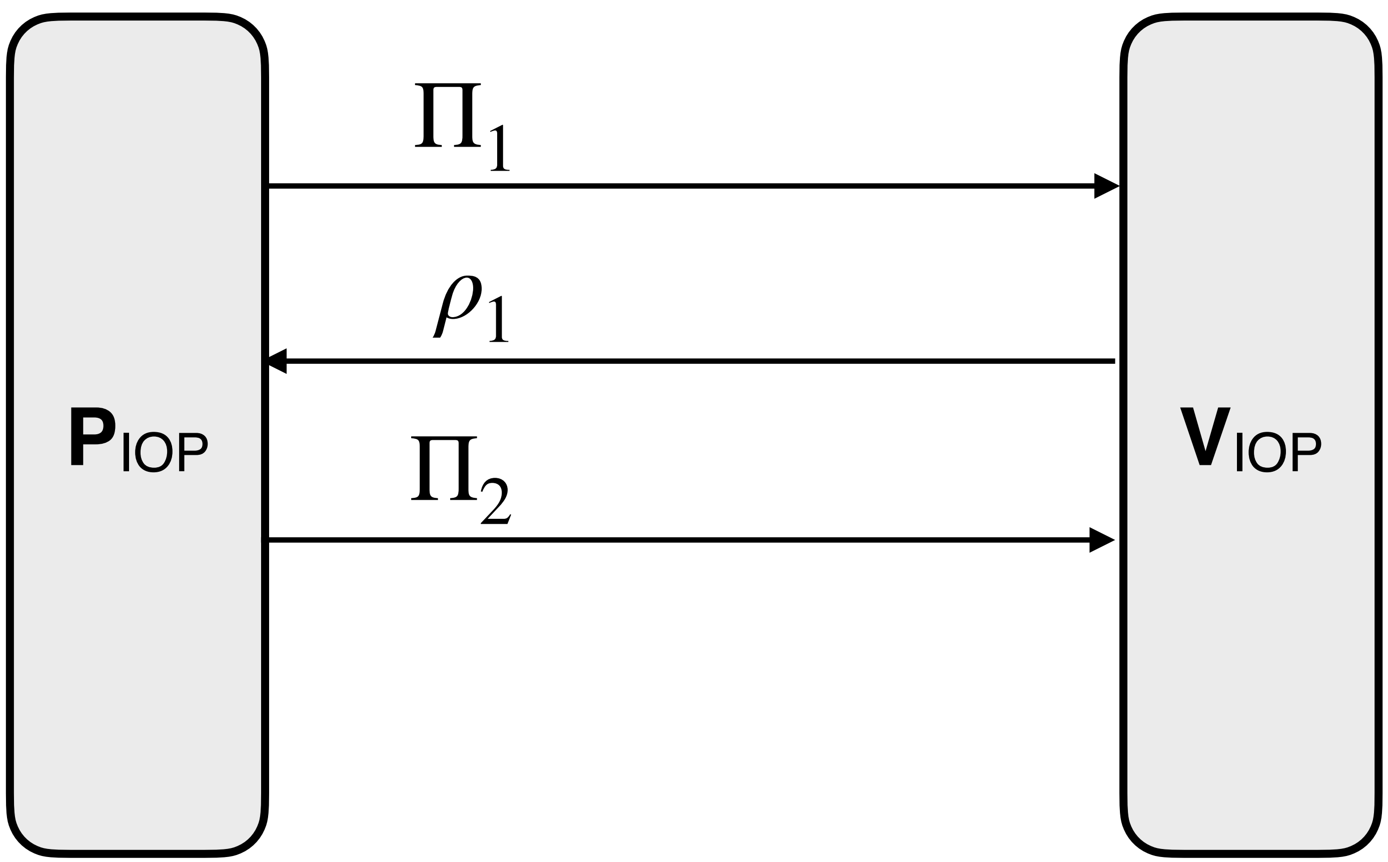


BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.

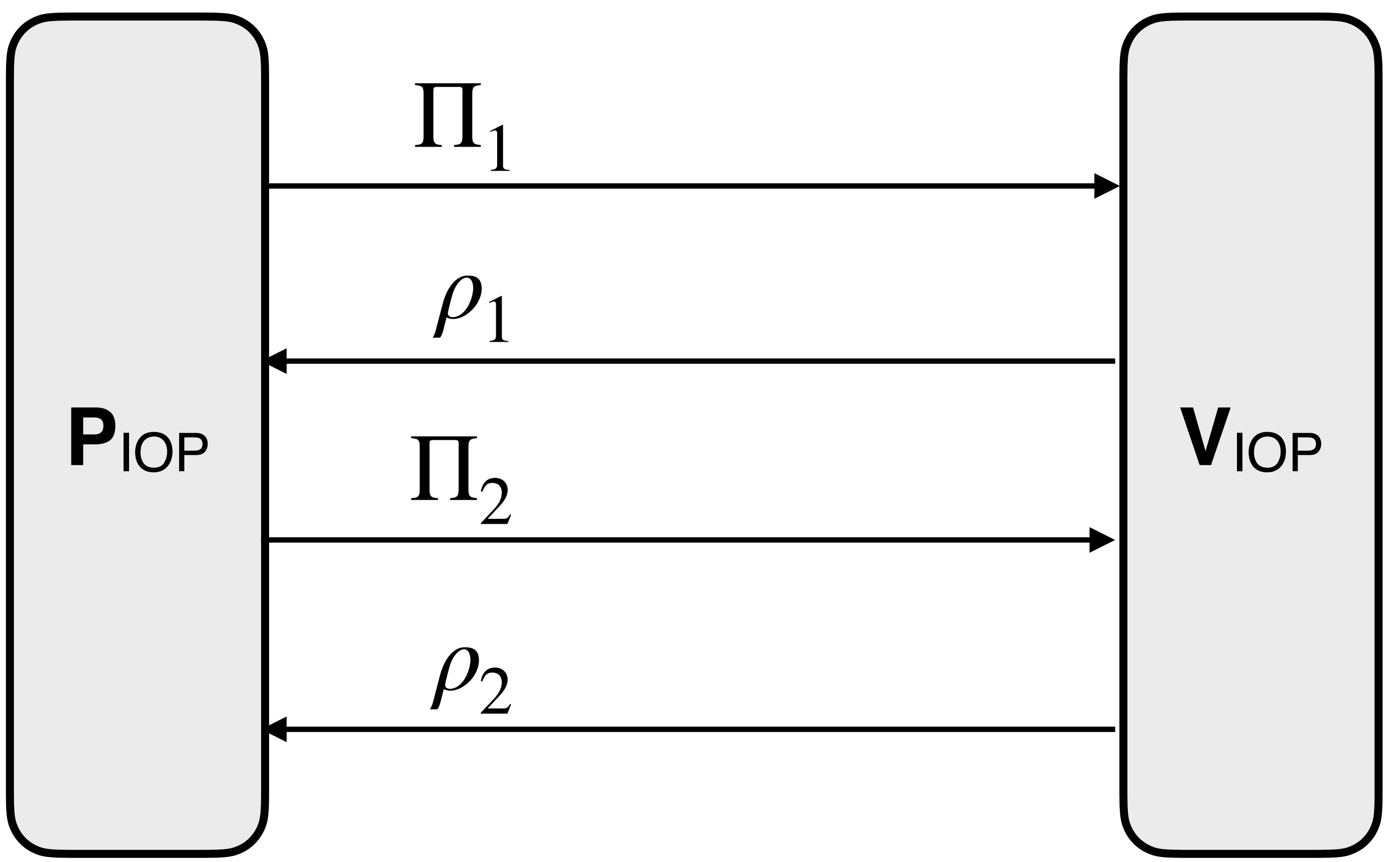


BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.

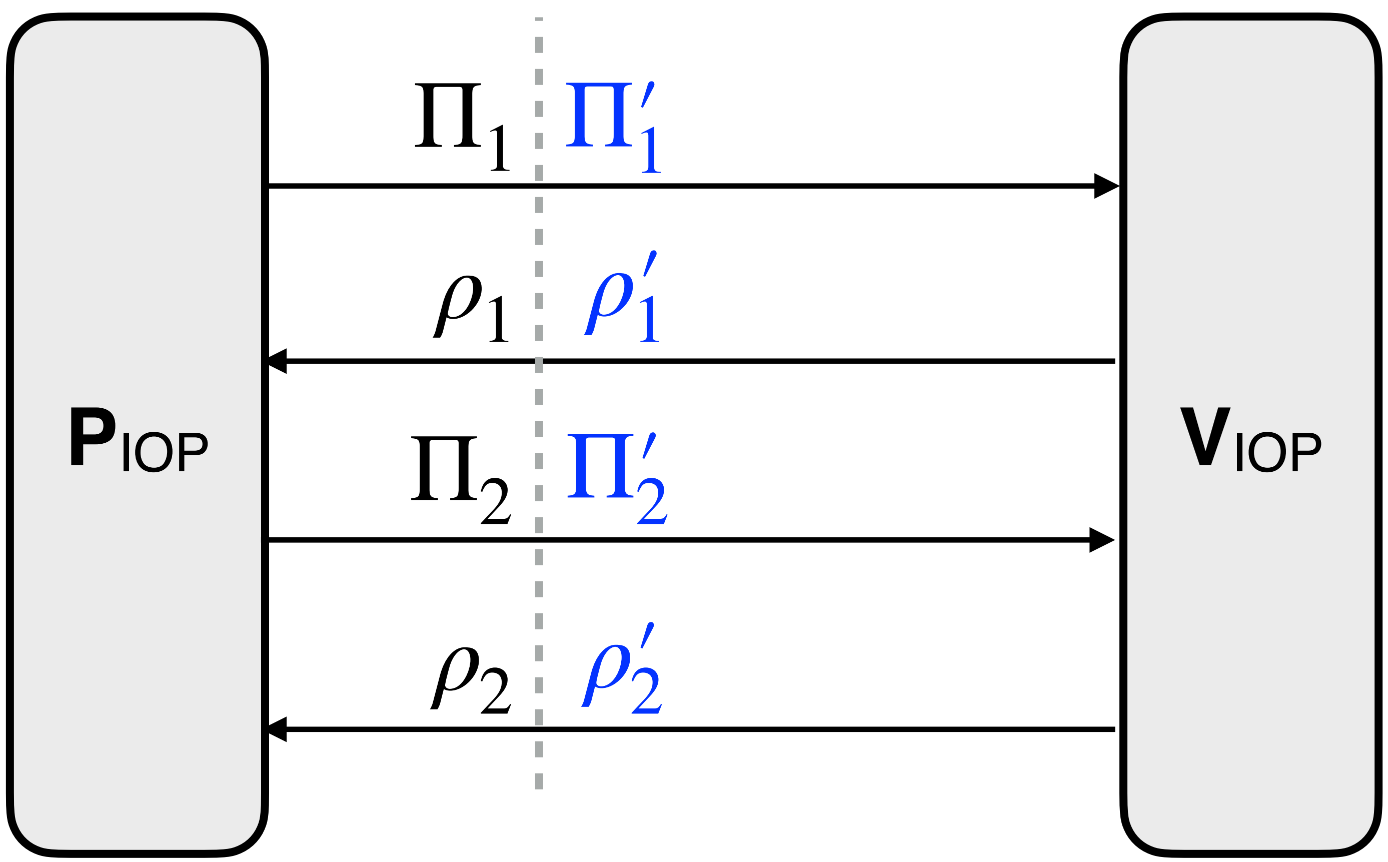


BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.

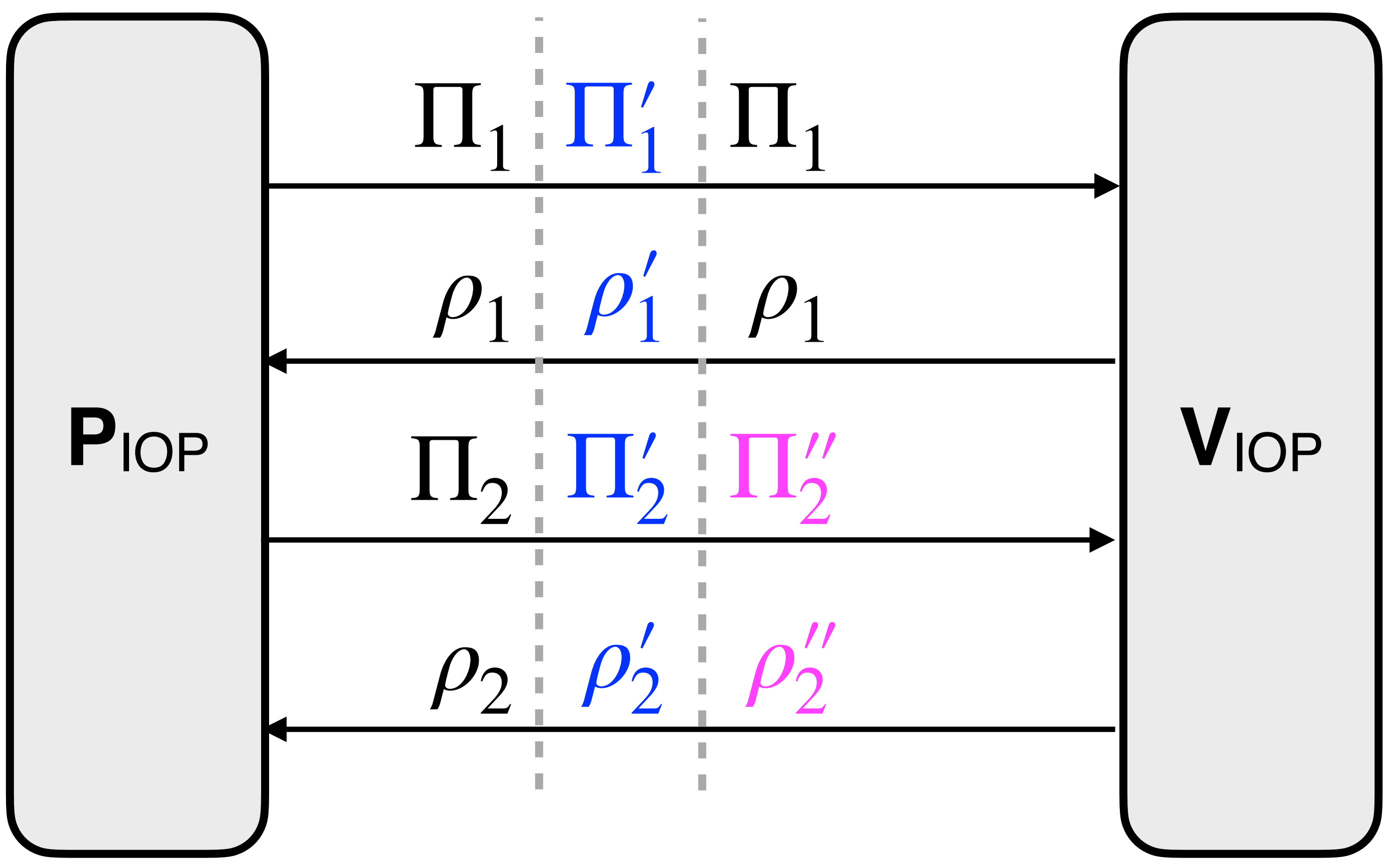


BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.

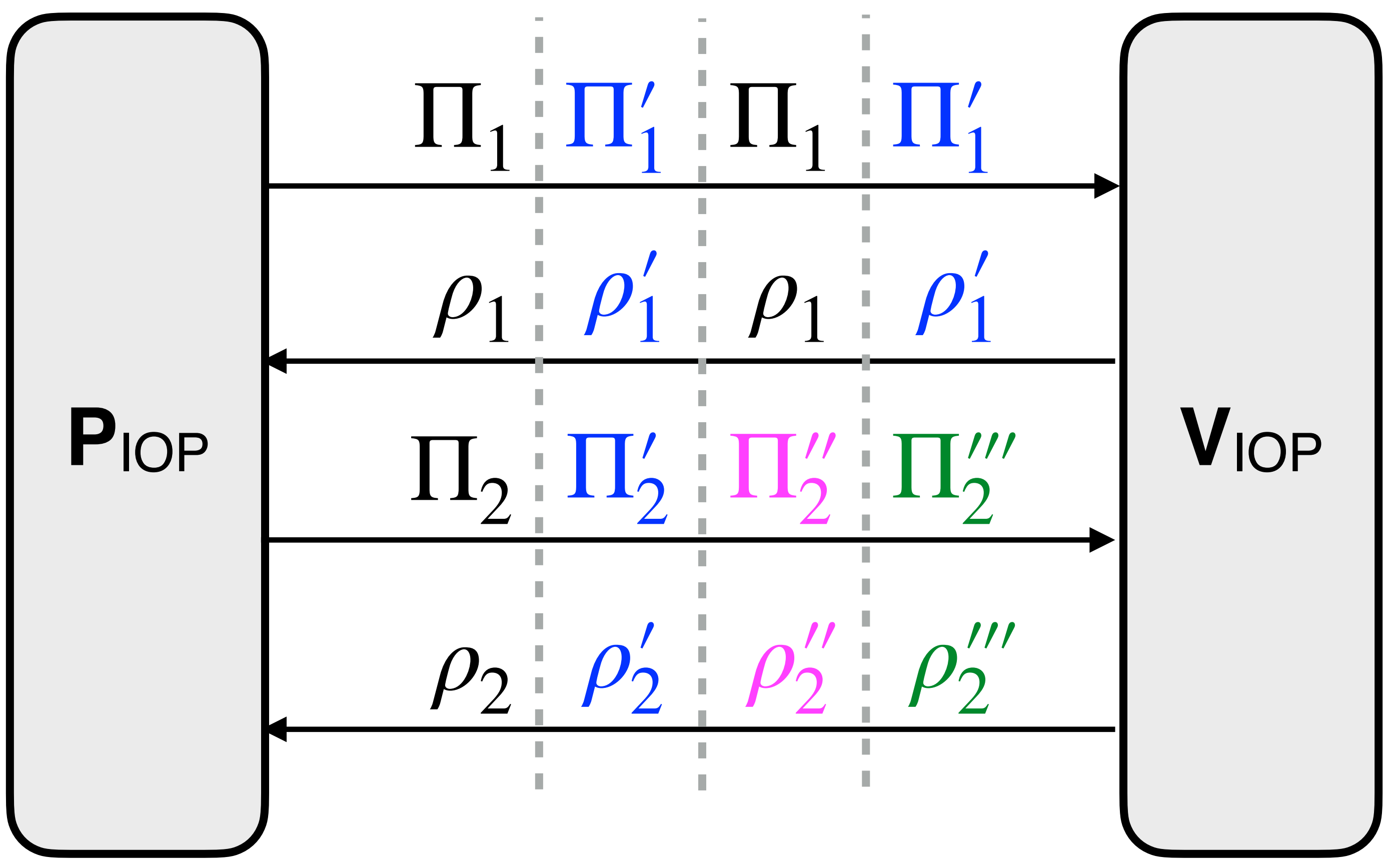


BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.

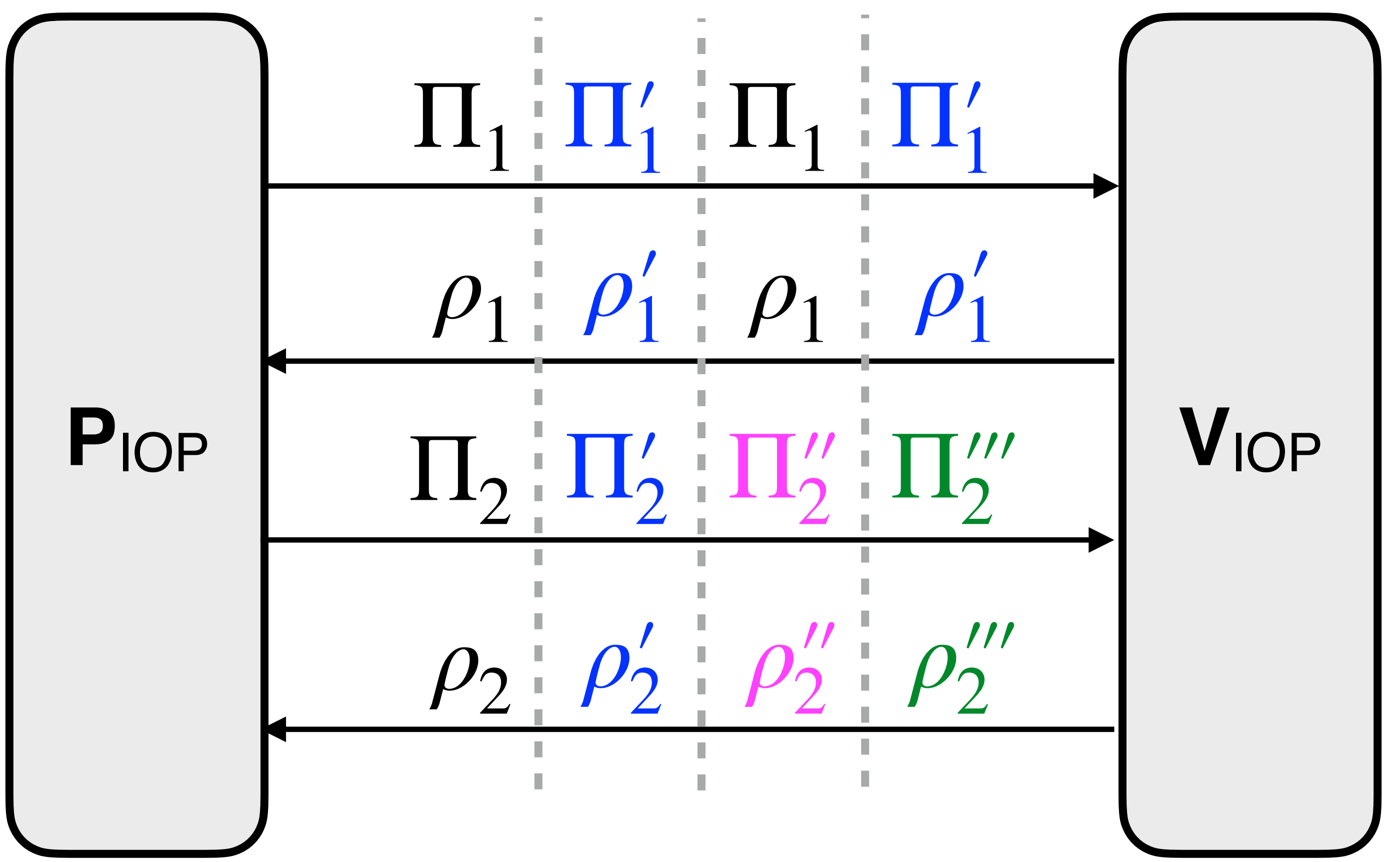


BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.



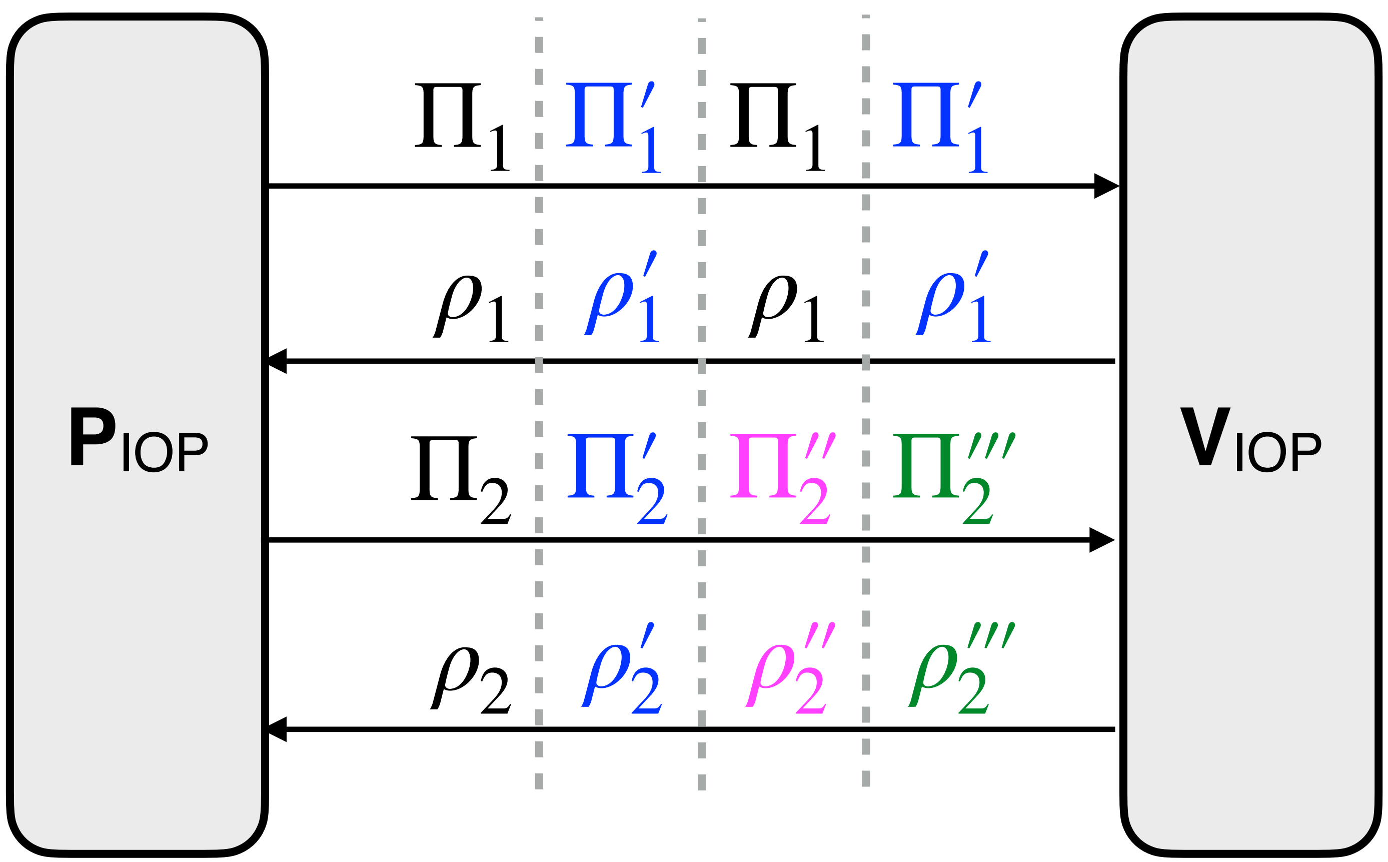
The cheating prover wins by finding **any** accepting transcript.

BCS Protocol

Time efficiency and communication are similar to the iBCS protocol.

Security of the BCS protocol is DIFFERENT.

An adversary can attack the IOP across multiple interactions, by trying different Merkle roots to obtain different IOP transcripts.



The cheating prover wins by finding **any** accepting transcript.

This is known as a **state-restoration attack**.

BCS Protocol

BCS Protocol

Definition: an IOP has *state-restoration soundness error* $\epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t)$ if every adversary that makes at most t moves in a state-restoration attack wins with probability at most $\epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t)$.

BCS Protocol

Definition: an IOP has *state-restoration soundness error* $\epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t)$ if every adversary that makes at most t moves in a state-restoration attack wins with probability at most $\epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t)$.

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks BCS}] \leq \epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t) + \frac{t^2}{2^\lambda}$$

BCS Protocol

Definition: an IOP has *state-restoration soundness error* $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t)$ if every adversary that makes at most t moves in a state-restoration attack wins with probability at most $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t)$.

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks BCS}] \leq \epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) + \frac{t^2}{2^\lambda}$$

Bad news: in general $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t)$ can be MUCH worse than ϵ_{IOP}

BCS Protocol

Definition: an IOP has *state-restoration soundness error* $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t)$ if every adversary that makes at most t moves in a state-restoration attack wins with probability at most $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t)$.

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks BCS}] \leq \epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) + \frac{t^2}{2^\lambda}$$

Bad news: in general $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t)$ can be MUCH worse than ϵ_{IOP}

Good news: IOPs of practical interest have $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) \approx t \cdot \epsilon_{\mathrm{IOP}}$

BCS Protocol

Definition: an IOP has *state-restoration soundness error* $\epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t)$ if every adversary that makes at most t moves in a state-restoration attack wins with probability at most $\epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t)$.

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks BCS}] \leq \epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t) + \frac{t^2}{2^\lambda}$$

Bad news: in general $\epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t)$ can be MUCH worse than ϵ_{IOP}

Good news: IOPs of practical interest have $\epsilon^{\mathrm{SR}}_{\mathrm{IOP}}(t) \approx t \cdot \epsilon_{\mathrm{IOP}}$

In this case, "security of BCS" ≈ "security of Micali", so using IOPs instead of PCPs gives practical efficiency "for free".

BCS Protocol

Definition: an IOP has *state-restoration soundness error* $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t)$ if every adversary that makes at most t moves in a state-restoration attack wins with probability at most $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t)$.

Theorem: $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ breaks BCS}] \leq \epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) + \frac{t^2}{2^\lambda}$$

Bad news: in general $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t)$ can be MUCH worse than ϵ_{IOP}

Good news: IOPs of practical interest have $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) \approx t \cdot \epsilon_{\mathrm{IOP}}$

In this case, "security of BCS" ≈ "security of Micali",
so using IOPs instead of PCPs gives practical efficiency "for free".

Remark: a PCP has state-restoration soundness error $t \cdot \epsilon_{\mathrm{PCP}}$, explaining the error term that appears for the Micali construction.

Achieving SR Soundness

Achieving SR Soundness

SR soundness is <u>necessary and sufficient</u> for the security of BCS.

Achieving SR Soundness

SR soundness is <u>necessary and sufficient</u> for the security of BCS.

Proving that an IOP satisfies SR soundness can be **laborious**.

Achieving SR Soundness

SR soundness is <u>necessary and sufficient</u> for the security of BCS.

Proving that an IOP satisfies SR soundness can be **laborious**.

Often easier: prove that the IOP satisfies a stronger soundness notion.

Achieving SR Soundness

SR soundness is necessary and sufficient for the security of BCS.

Proving that an IOP satisfies SR soundness can be **laborious**.

Often easier: prove that the IOP satisfies a stronger soundness notion.

Option #1: RBR (round-by-round) soundness

Achieving SR Soundness

SR soundness is necessary and sufficient for the security of BCS.

Proving that an IOP satisfies SR soundness can be **laborious**.

Often easier: prove that the IOP satisfies a stronger soundness notion.

Option #1: RBR (round-by-round) soundness

Lemma: k-round IOP has RBR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{RBR}}$
→ IOP has SR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) \leq (t+k) \cdot \epsilon_{\mathrm{IOP}}^{\mathrm{RBR}}$

Achieving SR Soundness

SR soundness is necessary and sufficient for the security of BCS.

Proving that an IOP satisfies SR soundness can be **laborious**.

Often easier: prove that the IOP satisfies a stronger soundness notion.

Option #1: RBR (round-by-round) soundness

Lemma: k-round IOP has RBR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{RBR}}$
→ IOP has SR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) \leq (t+k) \cdot \epsilon_{\mathrm{IOP}}^{\mathrm{RBR}}$

Option #2: special soundness

Achieving SR Soundness

SR soundness is <u>necessary and sufficient</u> for the security of BCS.

Proving that an IOP satisfies SR soundness can be **laborious**.

Often easier: prove that the IOP satisfies a stronger soundness notion.

Option #1: RBR (round-by-round) soundness

> **Lemma:** k-round IOP has RBR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{RBR}}$
> → IOP has SR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) \leq (t+k) \cdot \epsilon_{\mathrm{IOP}}^{\mathrm{RBR}}$

Option #2: special soundness

> **Lemma:** k-round IOP has $((a_i, C_i))_{i \in [k]}$-special soundness
> → IOP has SR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) \leq (t+1) \cdot \sum_{i \in [k]} \frac{a_i - 1}{C_i}$

Achieving SR Soundness

SR soundness is <u>necessary and sufficient</u> for the security of BCS.

Proving that an IOP satisfies SR soundness can be **laborious**.

Often easier: prove that the IOP satisfies a stronger soundness notion.

Option #1: RBR (round-by-round) soundness

Lemma: k-round IOP has RBR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{RBR}}$
→ IOP has SR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) \leq (t+k) \cdot \epsilon_{\mathrm{IOP}}^{\mathrm{RBR}}$

Option #2: special soundness

Lemma: k-round IOP has $((a_i, C_i))_{i \in [k]}$-special soundness
→ IOP has SR soundness error $\epsilon_{\mathrm{IOP}}^{\mathrm{SR}}(t) \leq (t+1) \cdot \sum_{i \in [k]} \frac{a_i - 1}{C_i}$

(Similar implications hold for knowledge soundness.)

Beyond Soundness

Additional Security Properties

Additional Security Properties

Knowledge soundness. Similar statements for knowledge soundness, provided the underlying probabilistic proof is knowledge sound.

Additional Security Properties

Knowledge soundness. Similar statements for knowledge soundness, provided the underlying probabilistic proof is knowledge sound.

Theorem: $\exists$ extractor E $\forall$ t-query adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}\mathcal{A} \text{ convinces BCS verifier} \\ \& \ E \text{ does not output a witness}\end{array}\right] \leq \kappa_{\mathrm{IOP}}^{\mathrm{SR}}(t) + \frac{t^2}{2^\lambda}$$

Additional Security Properties

Knowledge soundness. Similar statements for knowledge soundness, provided the underlying probabilistic proof is knowledge sound.

> **Theorem:** $\exists$ extractor E $\forall$ t-query adversary $\mathcal{A}$,
>
> $$\Pr\left[\begin{array}{c}\mathcal{A} \text{ convinces BCS verifier} \\ \& \; E \text{ does not output a witness}\end{array}\right] \leq \kappa_{\mathrm{IOP}}^{\mathrm{SR}}(t) + \frac{t^2}{2^\lambda}$$

Zero knowledge. If the MT.Commit is hiding (salt the leaves) and the Fiat–Shamir queries are salted, then zero-knowledge is achieved.

Additional Security Properties

Knowledge soundness. Similar statements for knowledge soundness, provided the underlying probabilistic proof is knowledge sound.

Theorem: ∃ extractor E ∀ t-query adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}\mathcal{A} \text{ convinces BCS verifier} \\ \& \ E \text{ does not output a witness}\end{array}\right] \leq \kappa_{\mathrm{IOP}}^{\mathrm{SR}}(t) + \frac{t^2}{2^\lambda}$$

Zero knowledge. If the MT.Commit is hiding (salt the leaves) and the Fiat–Shamir queries are salted, then zero-knowledge is achieved.

Theorem: ∀ t-query adversary $\mathcal{A}$, the statistical distance between real-world and ideal-world views in BCS is $\leq z_{\mathrm{IOP}} + z_{\mathrm{MT}}(t) + \dfrac{t}{2^\sigma}$

Additional Security Properties

Knowledge soundness. Similar statements for knowledge soundness, provided the underlying probabilistic proof is knowledge sound.

> **Theorem:** $\exists$ extractor E $\forall$ t-query adversary $\mathcal{A}$,
>
> $$\Pr\left[\begin{array}{c}\mathcal{A} \text{ convinces BCS verifier} \\ \& \ E \text{ does not output a witness}\end{array}\right] \le \kappa_{\mathrm{IOP}}^{\mathrm{SR}}(t) + \frac{t^2}{2^\lambda}$$

Zero knowledge. If the MT.Commit is hiding (salt the leaves) and the Fiat–Shamir queries are salted, then zero-knowledge is achieved.

> **Theorem:** $\forall$ t-query adversary $\mathcal{A}$, the statistical distance between real-world and ideal-world views in BCS is $\le z_{\mathrm{IOP}} + z_{\mathrm{MT}}(t) + \dfrac{t}{2^\sigma}$

What about **Non-Malleability**?

Simulation Soundness?

Simulation Knowledge Soundness?

UC-Security

UC-Security

Achieving security in the framework of **Universal Composability** (UC) is the gold standard for security in cryptography.

UC-Security

Achieving security in the framework of **Universal Composability** (UC) is the gold standard for security in cryptography.

UC-secure protocols achieve strong security guarantees against powerful adaptive adversaries, and retain these guarantees when used as part of larger protocols.

UC-Security

Achieving security in the framework of **Universal Composability** (UC) is the gold standard for security in cryptography.

UC-secure protocols achieve strong security guarantees against powerful adaptive adversaries, and retain these guarantees when used as part of larger protocols.

$\mathscr{F}_{\mathrm{ARG}}$:= ideal functionality for a zero-knowledge non-interactive argument of knowledge

UC-Security

Achieving security in the framework of **Universal Composability** (UC) is the gold standard for security in cryptography.

UC-secure protocols achieve strong security guarantees against powerful adaptive adversaries, and retain these guarantees when used as part of larger protocols.

$\mathscr{F}_{\mathrm{ARG}}$:= ideal functionality for a zero-knowledge non-interactive argument of knowledge

$\mathscr{F}_{\mathrm{GROM}}$:= ideal functionality for a GLOBAL random oracle

UC-Security

Achieving security in the framework of **Universal Composability** (UC) is the gold standard for security in cryptography.

UC-secure protocols achieve strong security guarantees against powerful adaptive adversaries, and retain these guarantees when used as part of larger protocols.

$\mathscr{F}_{\text{ARG}}$:= ideal functionality for a zero-knowledge non-interactive argument of knowledge

$\mathscr{F}_{\text{GROM}}$:= ideal functionality for a GLOBAL random oracle

Theorem: Micali and BCS emulate $\mathscr{F}_{\text{ARG}}$ in the $\mathscr{F}_{\text{GROM}}$-hybrid model

zkSNARKs in the ROM with Unconditional UC-Security

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Post-Quantum Security

Post-Quantum Security

A quantum adversary may query the random oracle **in superposition**.

Post-Quantum Security

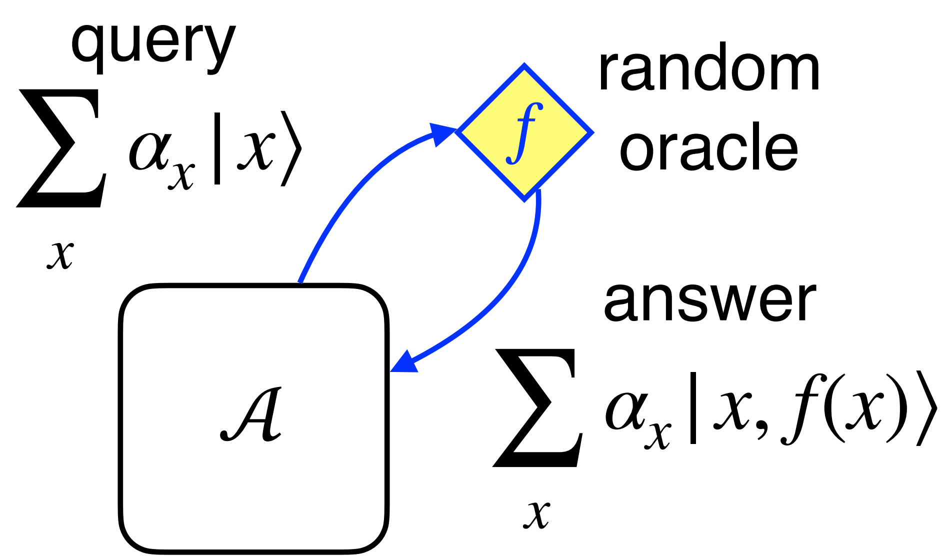
A quantum adversary may query the random oracle **in superposition**.

QROM
(quantum random
oracle model)

Post-Quantum Security

A quantum adversary may query the random oracle **in superposition**.

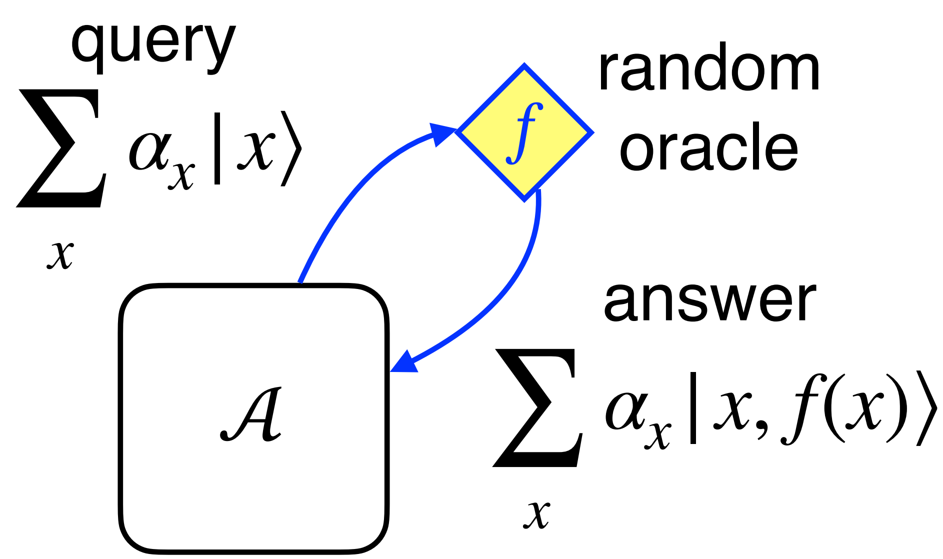
QROM
(quantum random oracle model)



Post-Quantum Security

A quantum adversary may query the random oracle **in superposition**.

QROM
(quantum random oracle model)



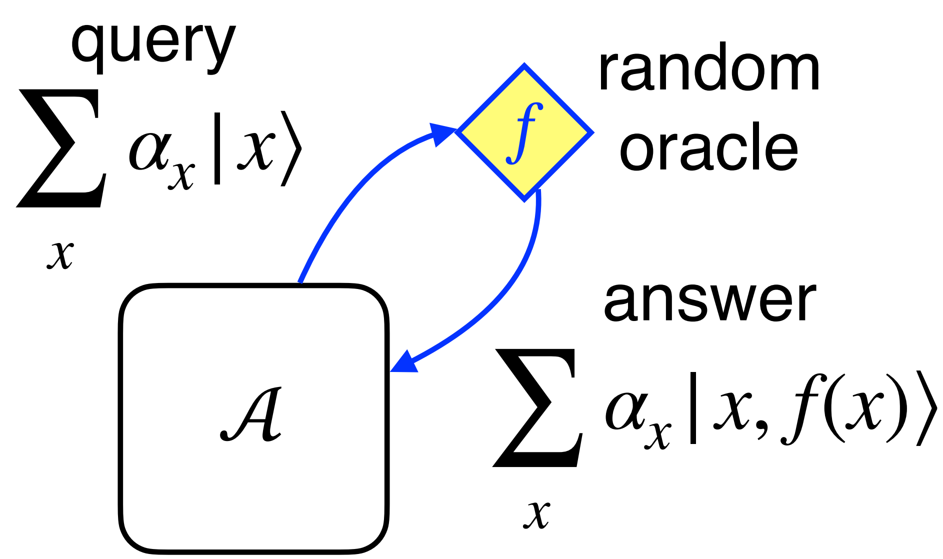
Security in the ROM does NOT imply security in the QROM:

Post-Quantum Security

A quantum adversary may query the random oracle **in superposition**.

QROM

(quantum random oracle model)



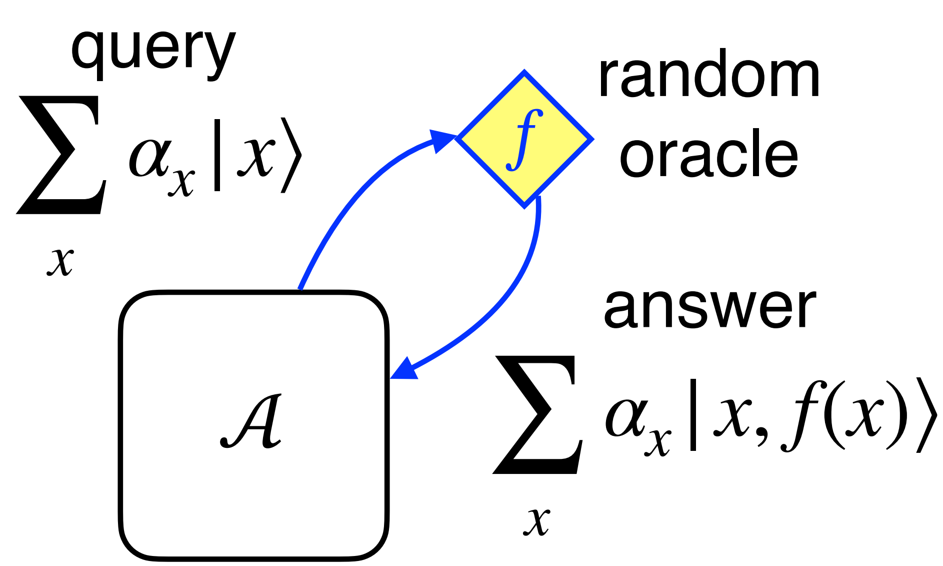
Security in the ROM does NOT imply security in the QROM:

∃ SNARG that is secure in ROM but not QROM.

Post-Quantum Security

A quantum adversary may query the random oracle **in superposition**.

QROM
(quantum random oracle model)



Security in the ROM does NOT imply security in the QROM:

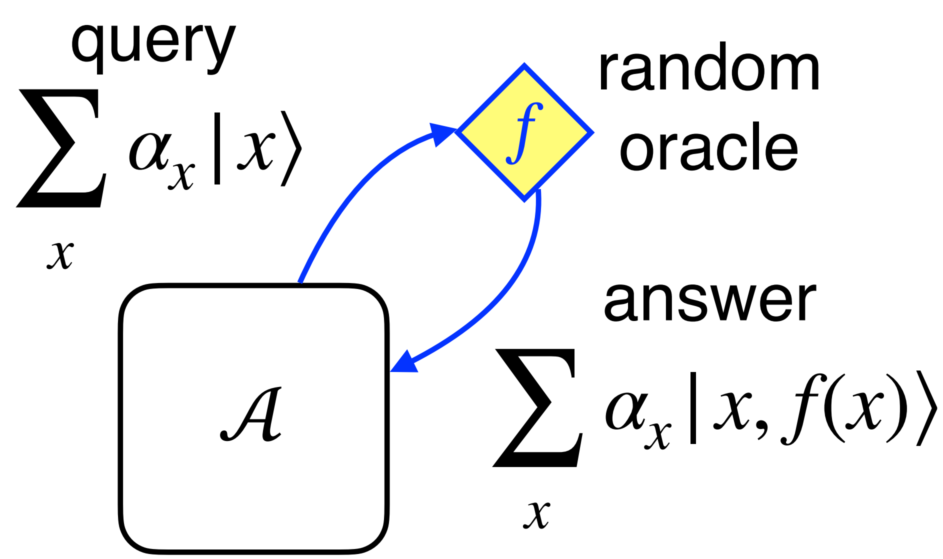
∃ SNARG that is secure in ROM but not QROM.

Nevertheless, SNARGs of interest **are** secure in the QROM:

Post-Quantum Security

A quantum adversary may query the random oracle **in superposition**.

QROM
(quantum random oracle model)



Security in the ROM does NOT imply security in the QROM:

∃ SNARG that is secure in ROM but not QROM.

Nevertheless, SNARGs of interest **are** secure in the QROM:

(1) ∀ t-query quantum $\mathcal{A}$, Pr[$\mathcal{A}$ breaks Micali] $\leq t^2 \cdot \epsilon_{\mathrm{PCP}} + \frac{t^3}{2^\lambda}$

(2) ∀ t-query quantum $\mathcal{A}$, Pr[$\mathcal{A}$ breaks BCS] $\leq (t+k)^2 \cdot \epsilon_{\mathrm{IOP}}^{\mathrm{RBR}} + \frac{t^3}{2^\lambda}$

Towards Optimal Argument Size

Towards Optimal Argument Size

Any (t,ϵ)-secure succinct argument in the ROM has argument size

$$\Omega\left(\log\frac{t}{\epsilon}\right).$$

Towards Optimal Argument Size

Any (t,ϵ)-secure succinct argument in the ROM has argument size

$$\Omega\left(\log \frac{t}{\epsilon}\right).$$

The Micali & BCS constructions achieve argument size $\tilde{O}\left(\log^2 \frac{t}{\epsilon}\right)$.

Towards Optimal Argument Size

Any (t,ϵ)-secure succinct argument in the ROM has argument size

$$\Omega\left(\log\frac{t}{\epsilon}\right).$$

The Micali & BCS constructions achieve argument size $\tilde{O}\left(\log^2\frac{t}{\epsilon}\right)$.

[CY21]: a SNARG with argument size $\tilde{O}\left(\log\frac{t}{\epsilon}\cdot\log t\right)$.

Towards Optimal Argument Size

Any (t,ϵ)-secure succinct argument in the ROM has argument size

$$\Omega\left(\log\frac{t}{\epsilon}\right).$$

The Micali & BCS constructions achieve argument size $\tilde{O}\left(\log^2\frac{t}{\epsilon}\right)$.

[CY21]: a SNARG with argument size $\tilde{O}\left(\log\frac{t}{\epsilon}\cdot\log t\right)$.

[HNY22]: this argument size is tight for a wide class of SNARGs.

Towards Optimal Argument Size

Any (t,ϵ)-secure succinct argument in the ROM has argument size

$$\Omega\left(\log\frac{t}{\epsilon}\right).$$

The Micali & BCS constructions achieve argument size $\tilde{O}\left(\log^2\frac{t}{\epsilon}\right)$.

[CY21]: a SNARG with argument size $\tilde{O}\left(\log\frac{t}{\epsilon}\cdot\log t\right)$.

[HNY22]: this argument size is tight for a wide class of SNARGs.

What is the optimal argument size of SNARGs in the ROM?

Towards Optimal Argument Size

Any (t,ϵ)-secure succinct argument in the ROM has argument size

$$\Omega\left(\log\frac{t}{\epsilon}\right).$$

The Micali & BCS constructions achieve argument size $\tilde{O}\left(\log^2\frac{t}{\epsilon}\right)$.

[CY21]: a SNARG with argument size $\tilde{O}\left(\log\frac{t}{\epsilon}\cdot\log t\right)$.

[HNY22]: this argument size is tight for a wide class of SNARGs.

What is the optimal argument size of SNARGs in the ROM?

IOPs are inherent: [CY20] shows a transformation **T** such that

succinct argument in the ROM $\longrightarrow \boxed{\mathbf{T}} \longrightarrow$ IOP

Want to Learn More?

Building Cryptographic Proofs from Hash Functions

A book by **Alessandro Chiesa** & **Eylon Yogev**

Comprehensive and rigorous treatment of SNARGs in the ROM.

Building Cryptographic Proofs from Hash Functions

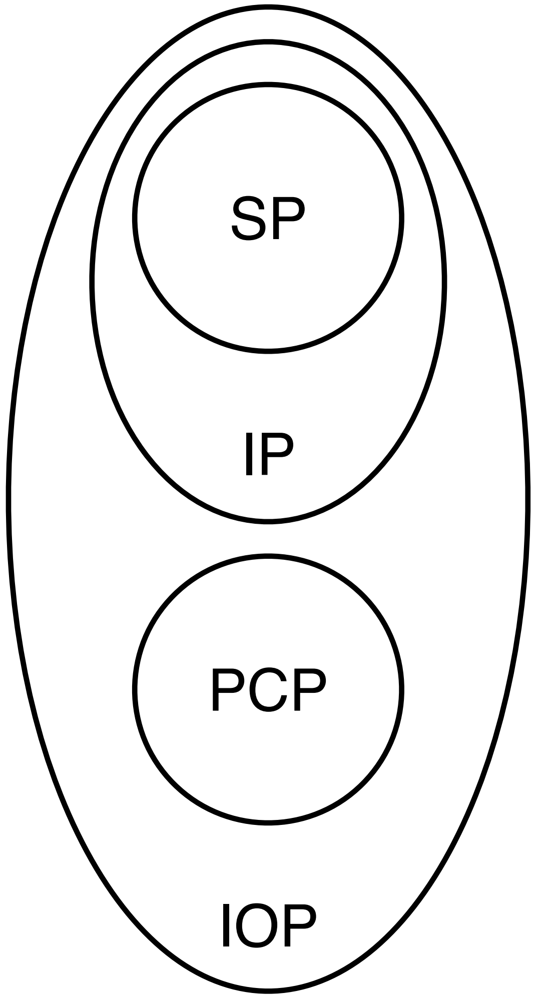
A book by **Alessandro Chiesa** & **Eylon Yogev**

Comprehensive and rigorous treatment of SNARGs in the ROM.

Recall: "SNARG in the ROM" ≡ "**compilation** of a probabilistic proof"

Comprehensive and rigorous treatment of SNARGs in the ROM.

Recall: "SNARG in the ROM" ≡ "**compilation** of a probabilistic proof"

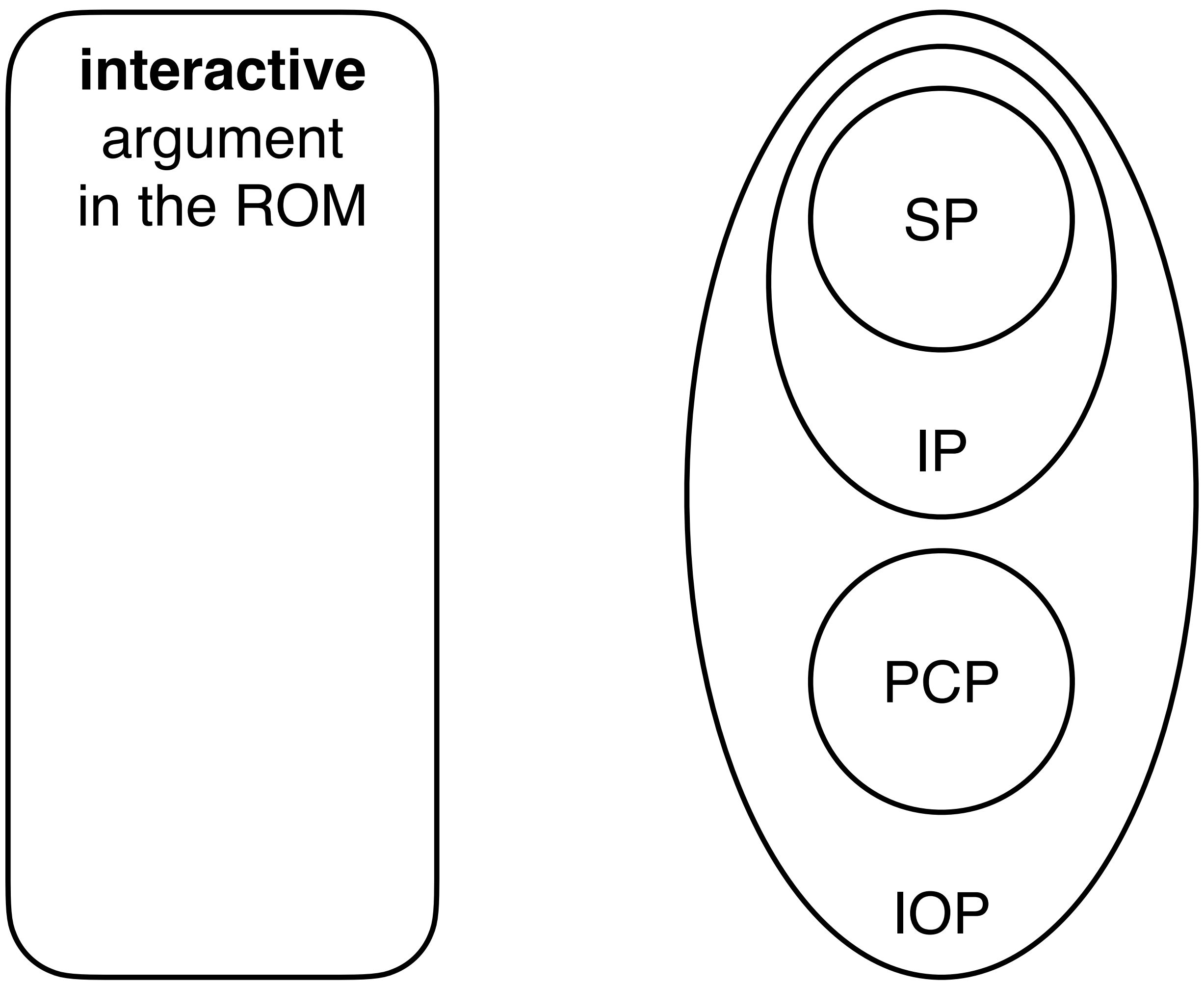


Building Cryptographic Proofs from Hash Functions

A book by **Alessandro Chiesa** & **Eylon Yogev**

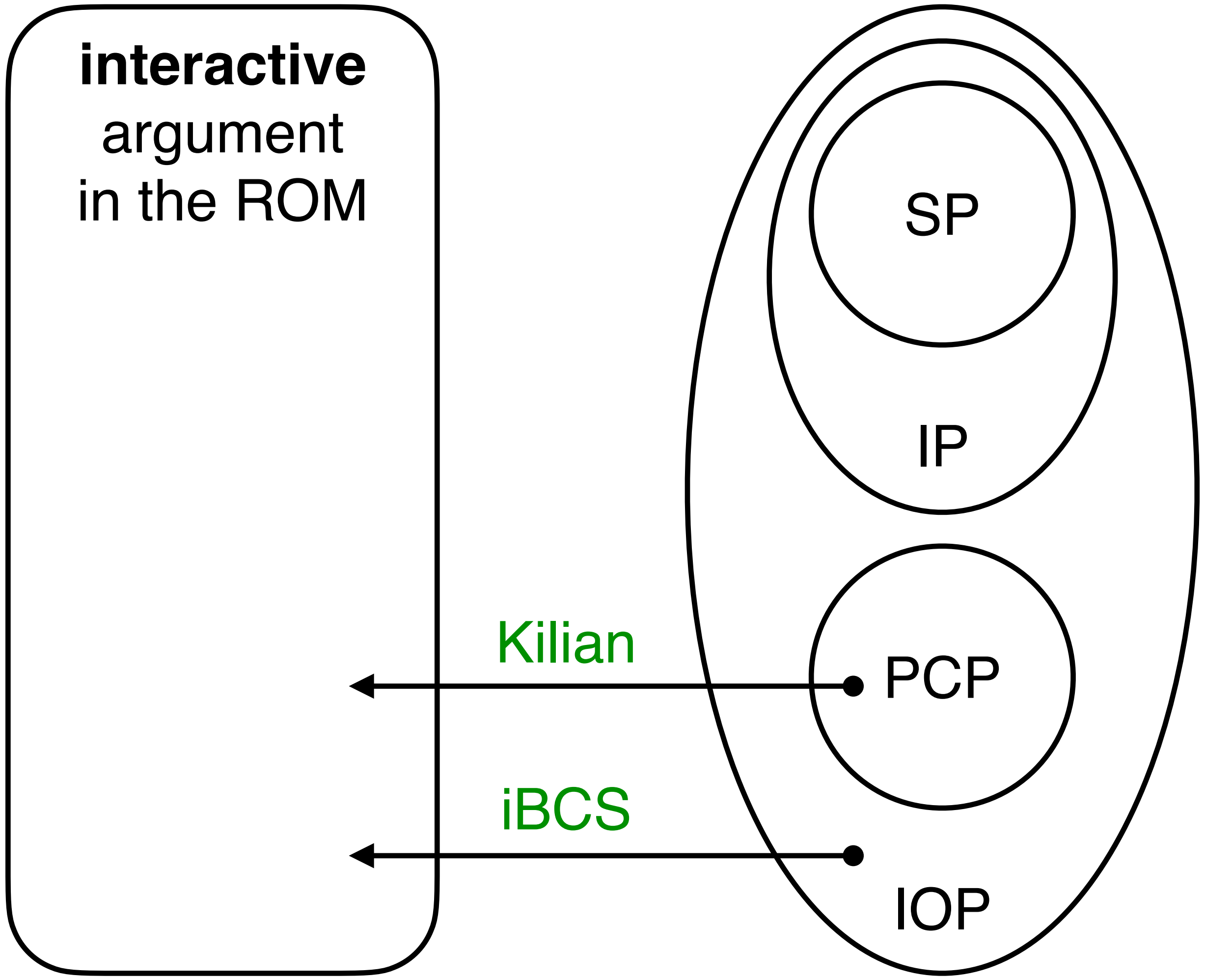
Comprehensive and rigorous treatment of SNARGs in the ROM.

Recall: "SNARG in the ROM" ≡ "**compilation** of a probabilistic proof"



Comprehensive and rigorous treatment of SNARGs in the ROM.

Recall: "SNARG in the ROM" ≡ "**compilation** of a probabilistic proof"

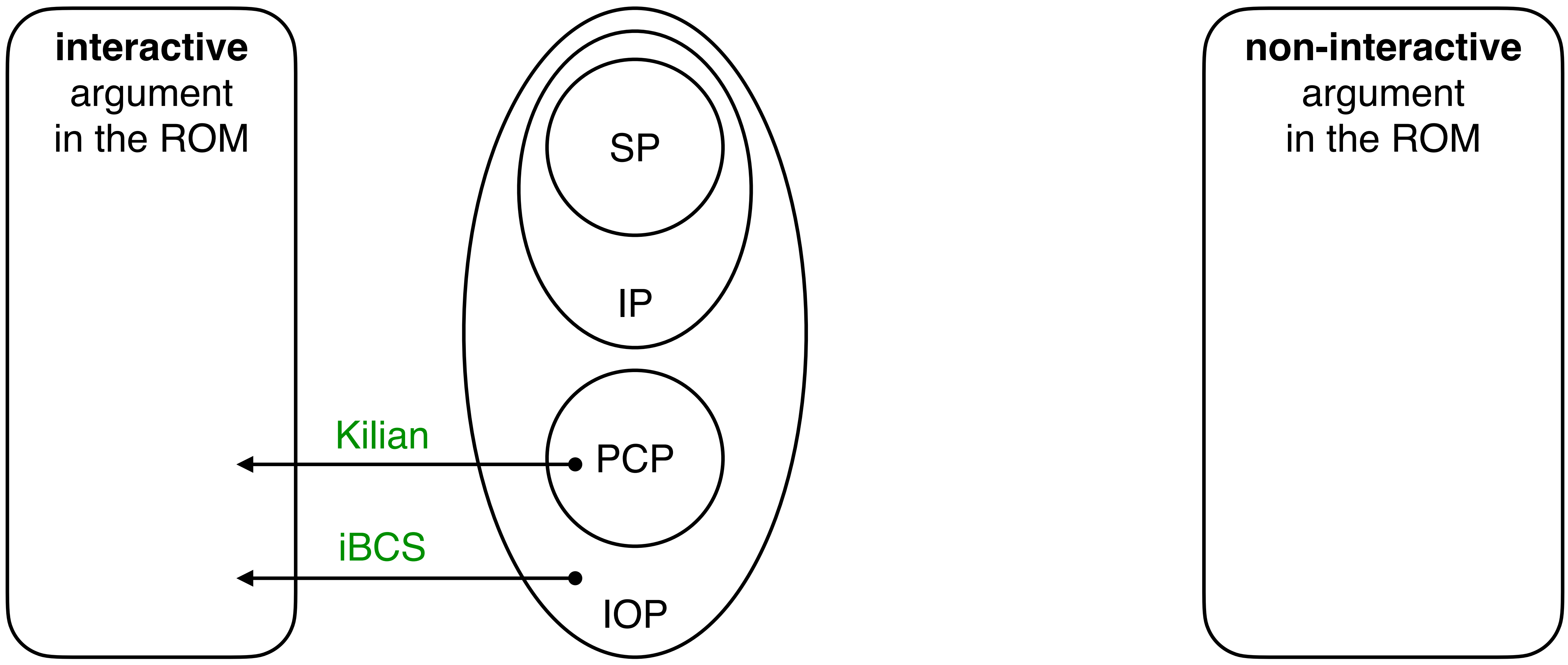


Building Cryptographic Proofs from Hash Functions

A book by **Alessandro Chiesa** & **Eylon Yogev**

Comprehensive and rigorous treatment of SNARGs in the ROM.

Recall: "SNARG in the ROM" ≡ "**compilation** of a probabilistic proof"

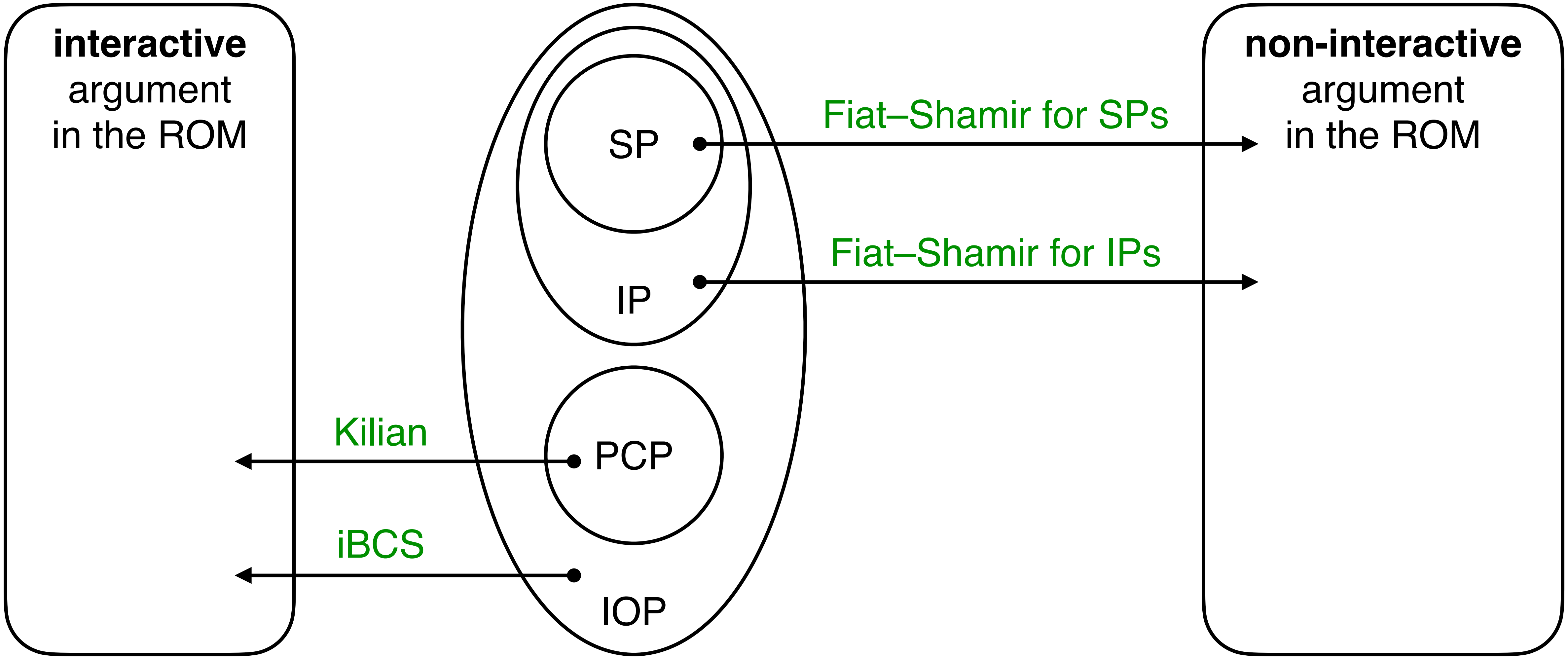


Building Cryptographic Proofs from Hash Functions

A book by **Alessandro Chiesa** & **Eylon Yogev**

Comprehensive and rigorous treatment of SNARGs in the ROM.

Recall: "SNARG in the ROM" ≡ "**compilation** of a probabilistic proof"

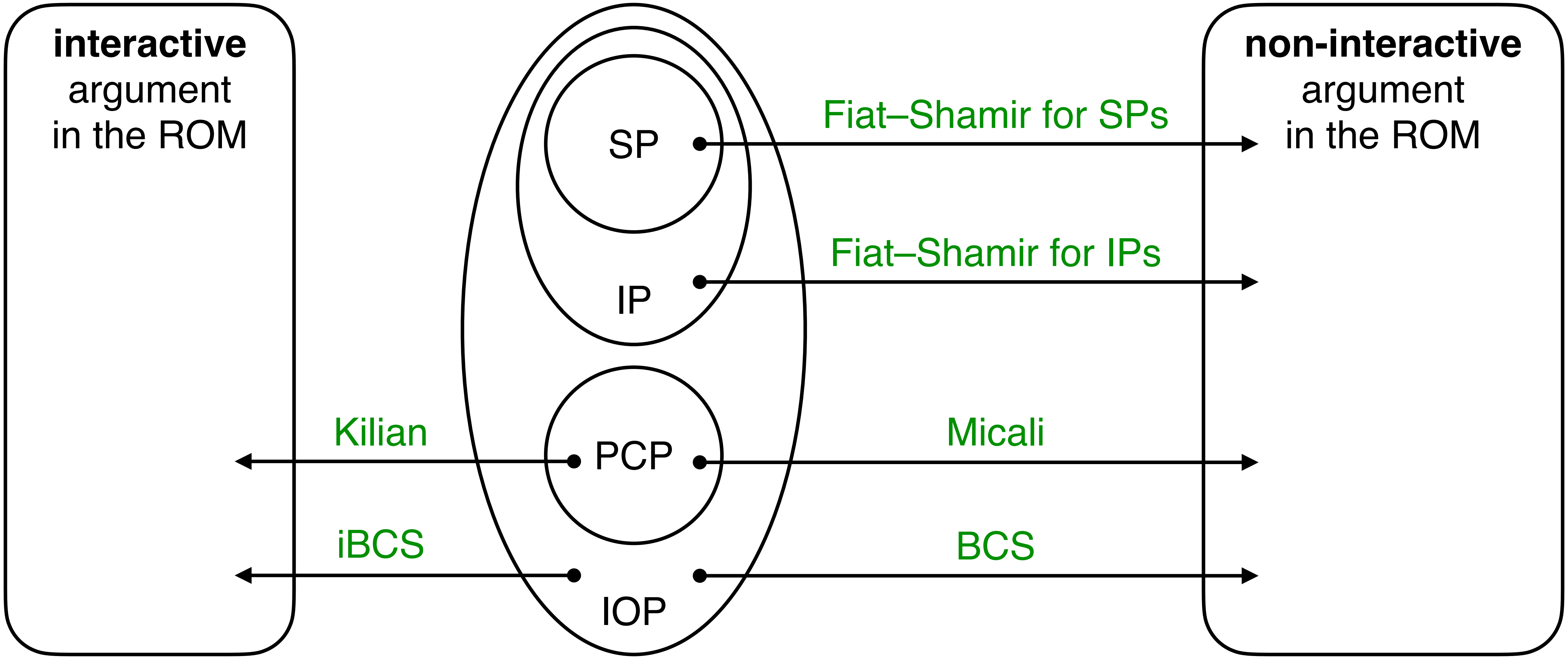


Building Cryptographic Proofs from Hash Functions

A book by **Alessandro Chiesa** & **Eylon Yogev**

Comprehensive and rigorous treatment of SNARGs in the ROM.

Recall: "SNARG in the ROM" ≡ "**compilation** of a probabilistic proof"



Book Contents

Book Contents

The book is divided in several parts:

Book Contents

The book is divided in several parts:

Part I
What are
Cryptographic Proofs?

Book Contents

The book is divided in several parts:

Part I
What are
Cryptographic Proofs?

Part II
NARGs Based on SPs
• FS protocol for SPs

Book Contents

The book is divided in several parts:

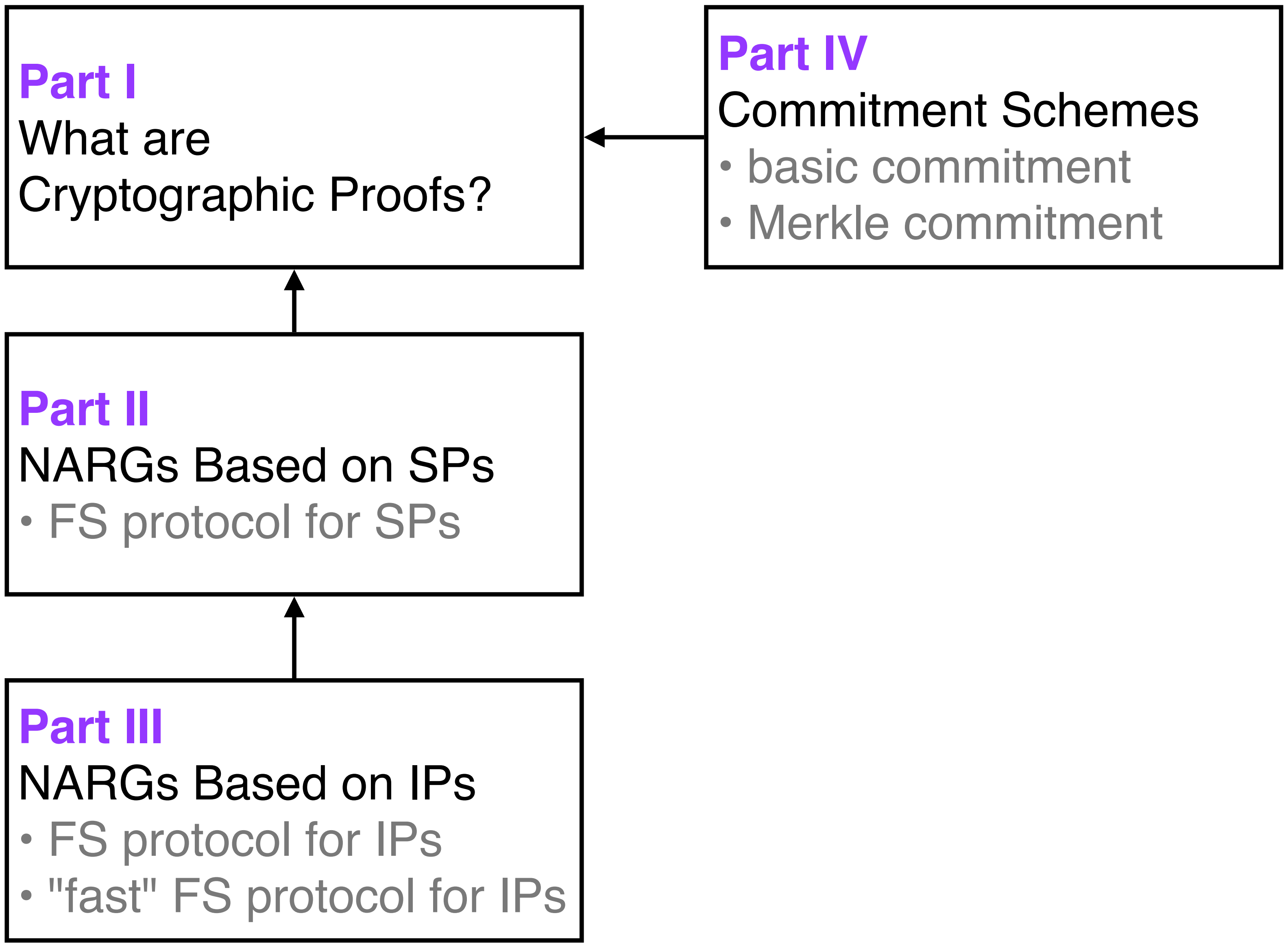
Part I
What are
Cryptographic Proofs?

Part II
NARGs Based on SPs
• FS protocol for SPs

Part III
NARGs Based on IPs
• FS protocol for IPs
• "fast" FS protocol for IPs

Book Contents

The book is divided in several parts:



Book Contents

The book is divided in several parts:

Part I
What are Cryptographic Proofs?

Part II
NARGs Based on SPs
- FS protocol for SPs

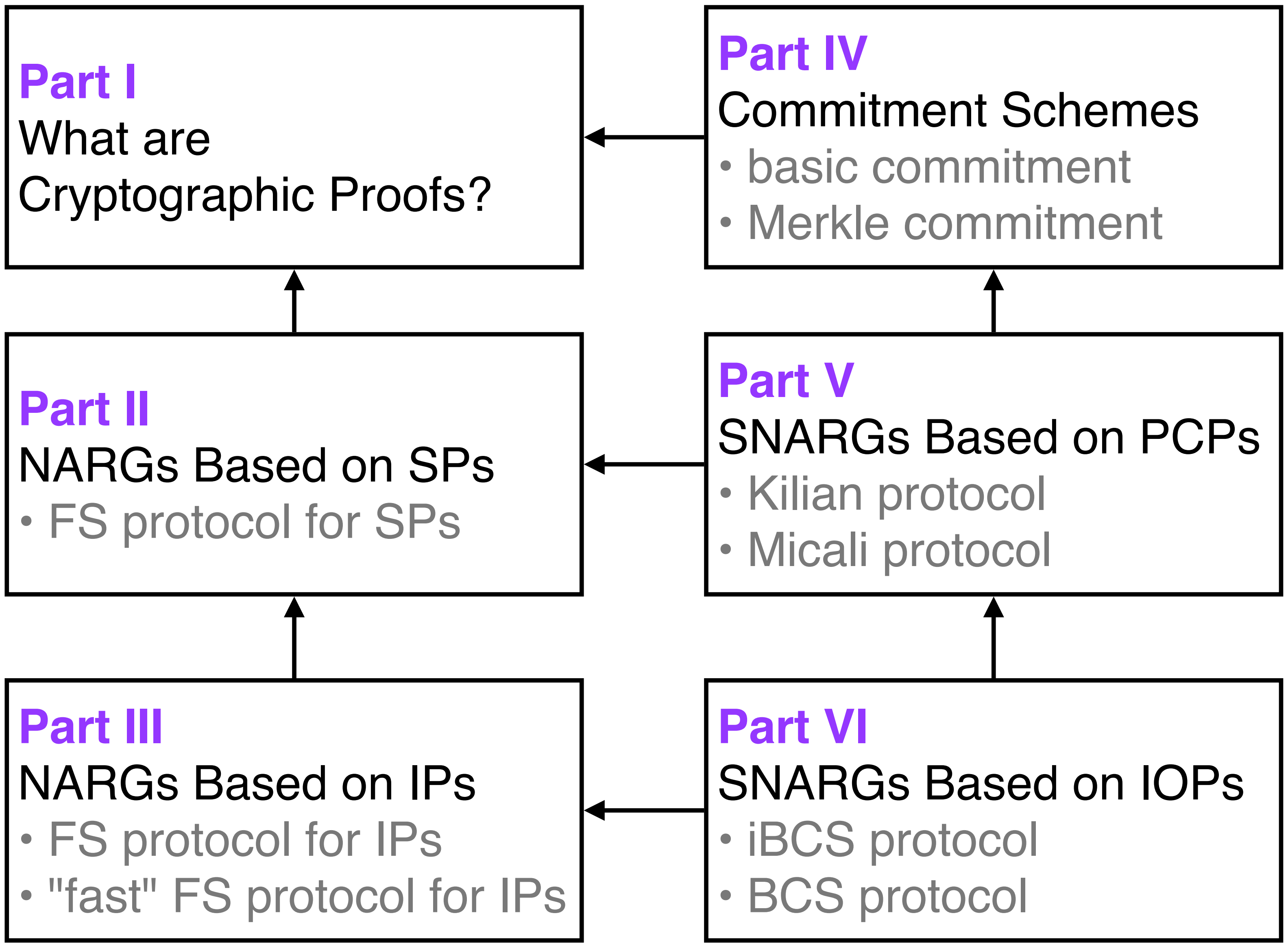
Part III
NARGs Based on IPs
- FS protocol for IPs
- "fast" FS protocol for IPs

Part IV
Commitment Schemes
- basic commitment
- Merkle commitment

Part V
SNARGs Based on PCPs
- Kilian protocol
- Micali protocol

Book Contents

The book is divided in several parts:



Book Contents

The book is divided in several parts:

Part I
What are Cryptographic Proofs?

Part II
NARGs Based on SPs
- FS protocol for SPs

Part III
NARGs Based on IPs
- FS protocol for IPs
- "fast" FS protocol for IPs

Part IV
Commitment Schemes
- basic commitment
- Merkle commitment

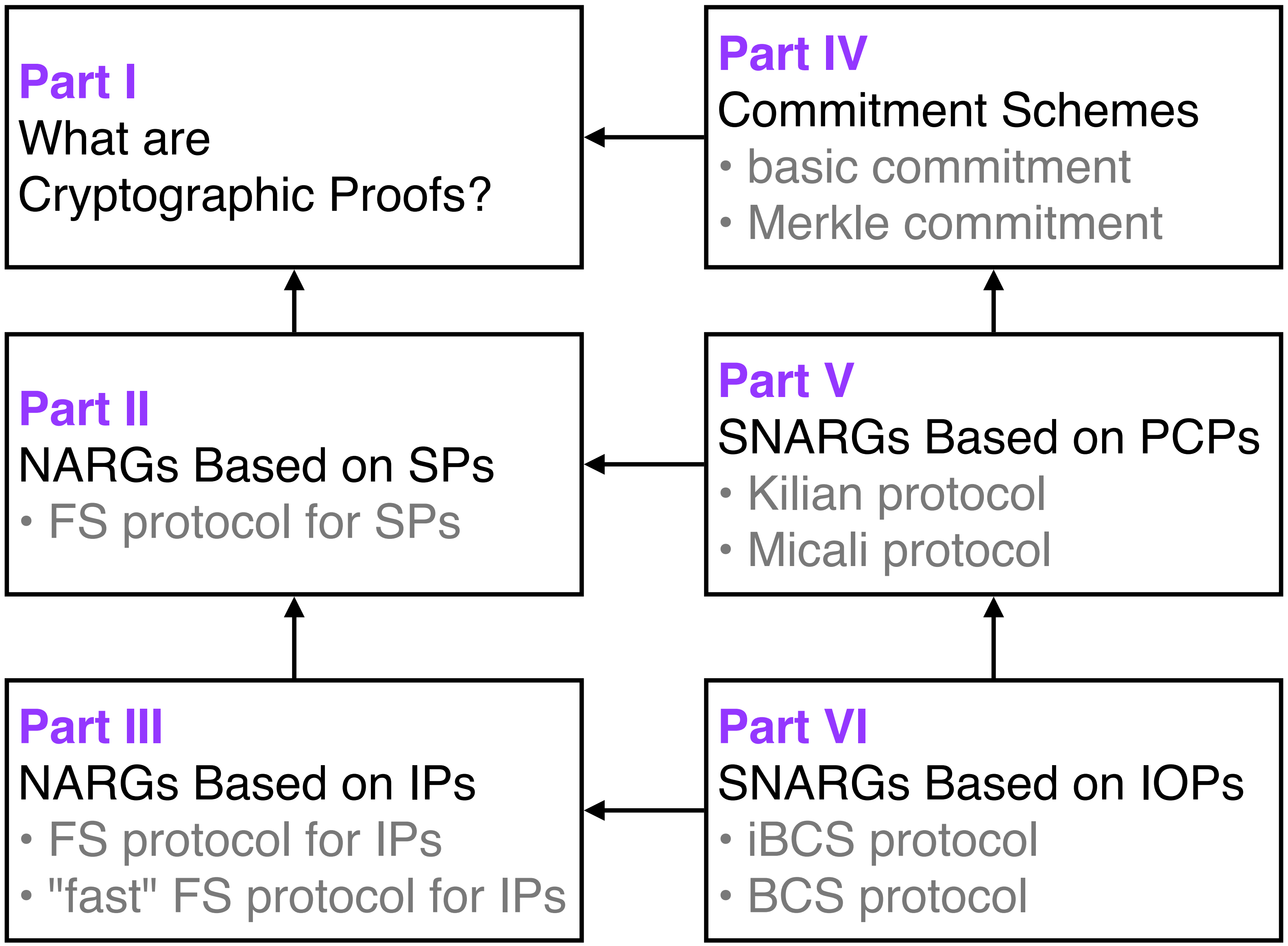
Part V
SNARGs Based on PCPs
- Kilian protocol
- Micali protocol

Part VI
SNARGs Based on IOPs
- iBCS protocol
- BCS protocol

Part VII
Practical Considerations
- setting parameters
- Merkle optimizations
- RBR soundness
- special soundness

Book Contents

The book is divided in several parts:



Part VII
Practical Considerations
- setting parameters
- Merkle optimizations
- RBR soundness
- special soundness

`hash-based-snargs-book.github.io`

Book Contents

The book is divided in several parts:

Part I
What are Cryptographic Proofs?

Part II
NARGs Based on SPs
- FS protocol for SPs

Part III
NARGs Based on IPs
- FS protocol for IPs
- "fast" FS protocol for IPs

Part IV
Commitment Schemes
- basic commitment
- Merkle commitment

Part V
SNARGs Based on PCPs
- Kilian protocol
- Micali protocol

Part VI
SNARGs Based on IOPs
- iBCS protocol
- BCS protocol

Part VII
Practical Considerations
- setting parameters
- Merkle optimizations
- RBR soundness
- special soundness

`hash-based-snargs-book.github.io`

Book Contents

The book is divided in several parts:

Part I
What are Cryptographic Proofs?

Part II
NARGs Based on SPs
- FS protocol for SPs

Part III
NARGs Based on IPs
- FS protocol for IPs
- "fast" FS protocol for IPs

Part IV
Commitment Schemes
- basic commitment
- Merkle commitment

Part V
SNARGs Based on PCPs
- Kilian protocol
- Micali protocol

Part VI
SNARGs Based on IOPs
- iBCS protocol
- BCS protocol

Part VII
Practical Considerations
- setting parameters
- Merkle optimizations
- RBR soundness
- special soundness

`hash-based-snargs-book.github.io`

Comments and suggestions are welcome.

Thanks!