Games and Logic for Cryptographic Protocol Verification

Justine Sauvage

Supervisors:

Bruno Blanchet

Adrien Koutsos

David Baelde

21-01-2026

1. Introduction

Games and Logic for Cryptographic Protocol Verification

Games and Logic for Cryptographic Protocol Verification

Cryptographic protocol:
Distributed program securing communications

- Messaging (e.g. Signal): Confidentiality
- Internet navigation (e.g. TLS): Authentication
- E-voting: Privacy

Games and Logic for Cryptographic Protocol Verification

Cryptographic protocol:
Distributed program securing communications

- Messaging (e.g. Signal): Confidentiality
- Internet navigation (e.g. TLS): Authentication
- E-voting: Privacy

Verification:
Several layers of verifications:
High-level specification

Games and logic

- Formal methods
- Cryptographic reductions
- Computational model ($\neq$ symbolic model)

Games and Logic for Cryptographic Protocol Verification

Cryptographic protocol:
Distributed program securing communications

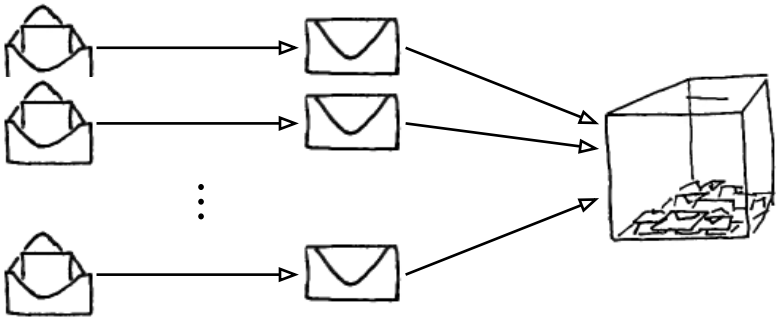
- Messaging (e.g. Signal): Confidentiality
- Internet navigation (e.g. TLS): Authentication
- E-voting: Privacy

Verification:
Several layers of verifications:
High-level specification

2. This thesis

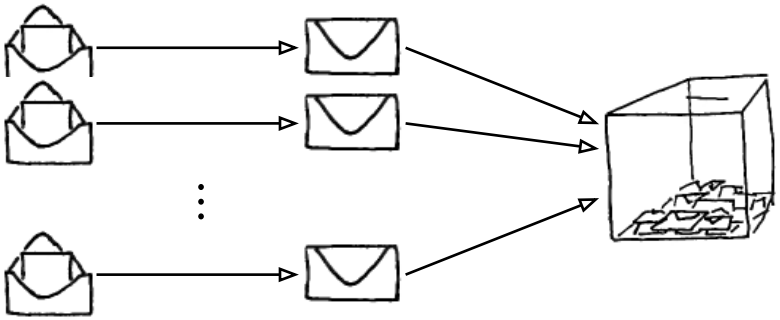
Running example: voting

Collect phase:

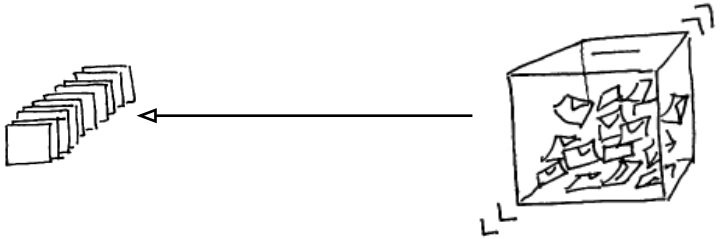


Running example: voting

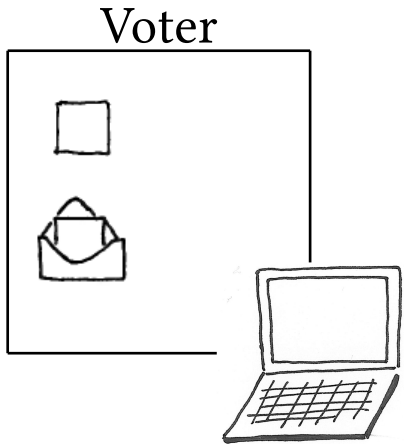
Collect phase:



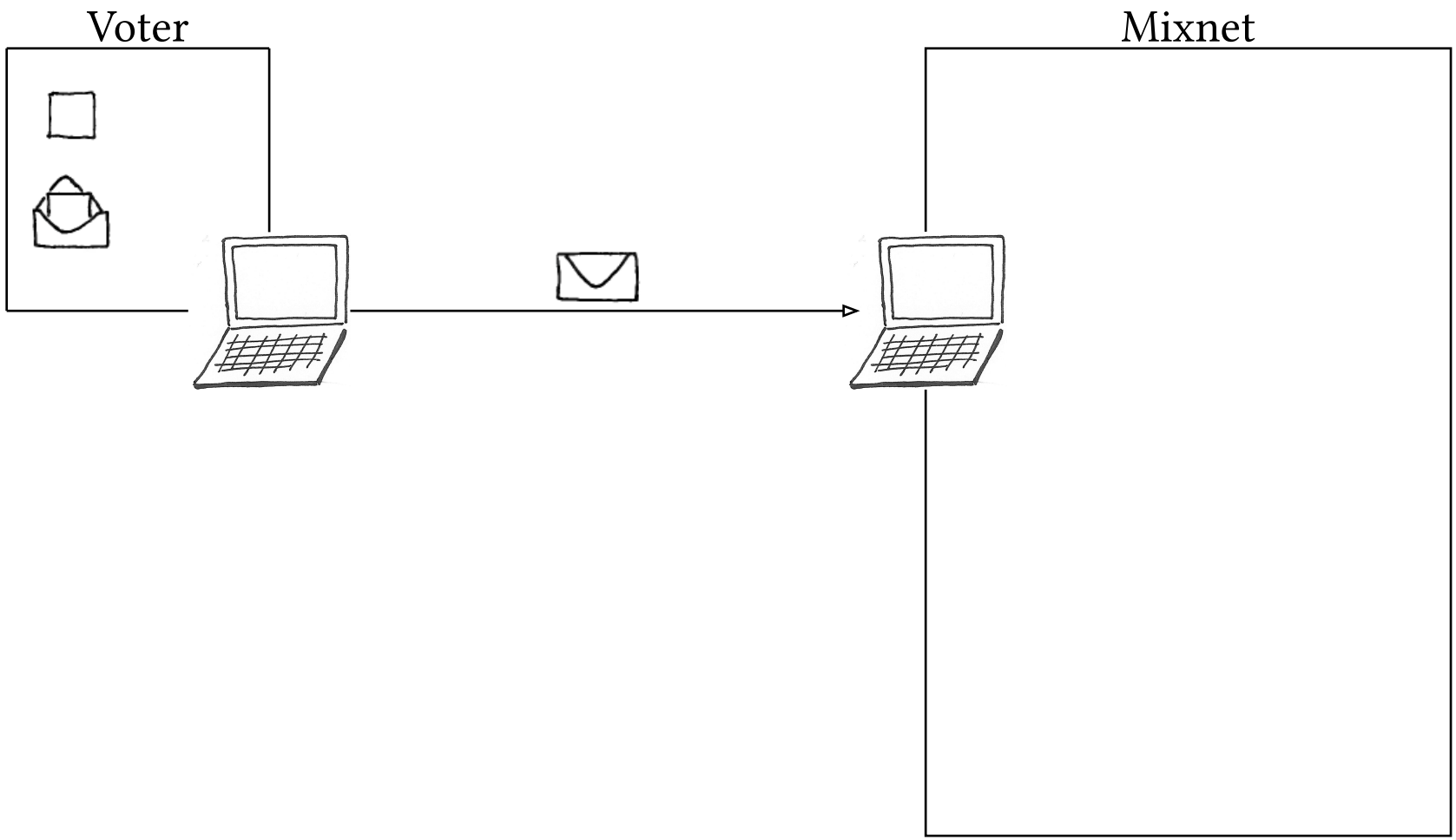
Publish phase:



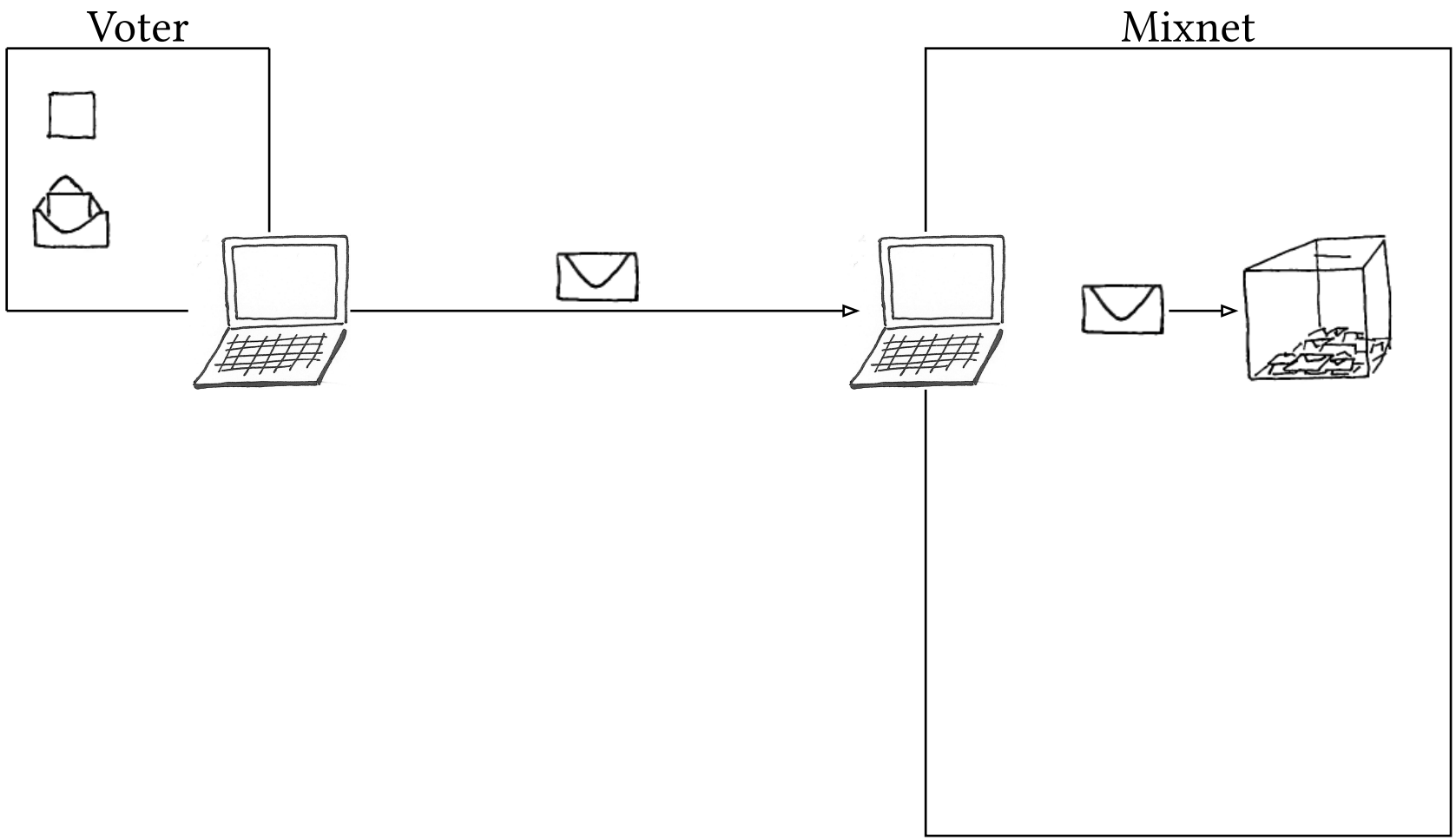
Running example: very simplified voting protocol



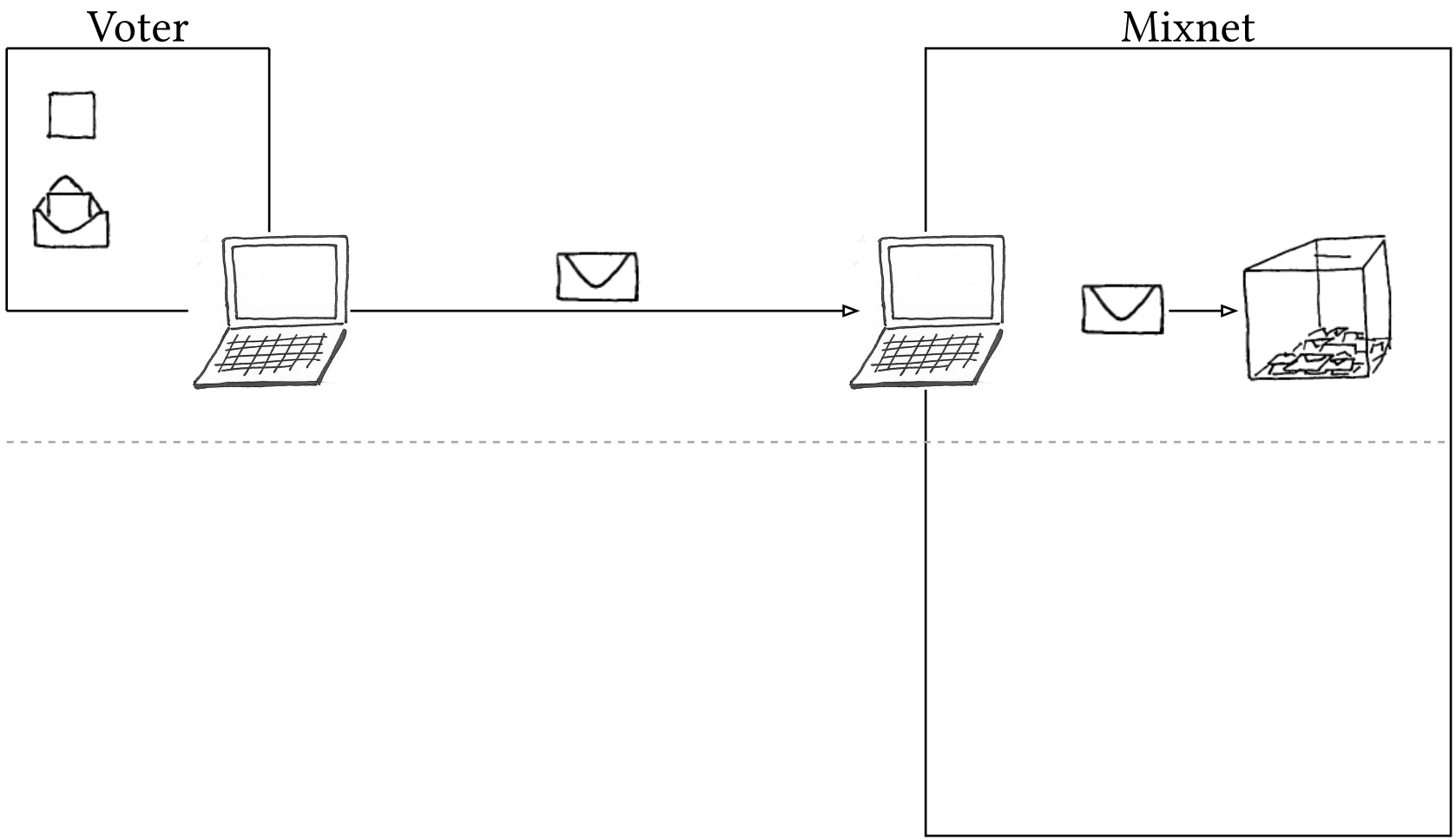
Running example: very simplified voting protocol



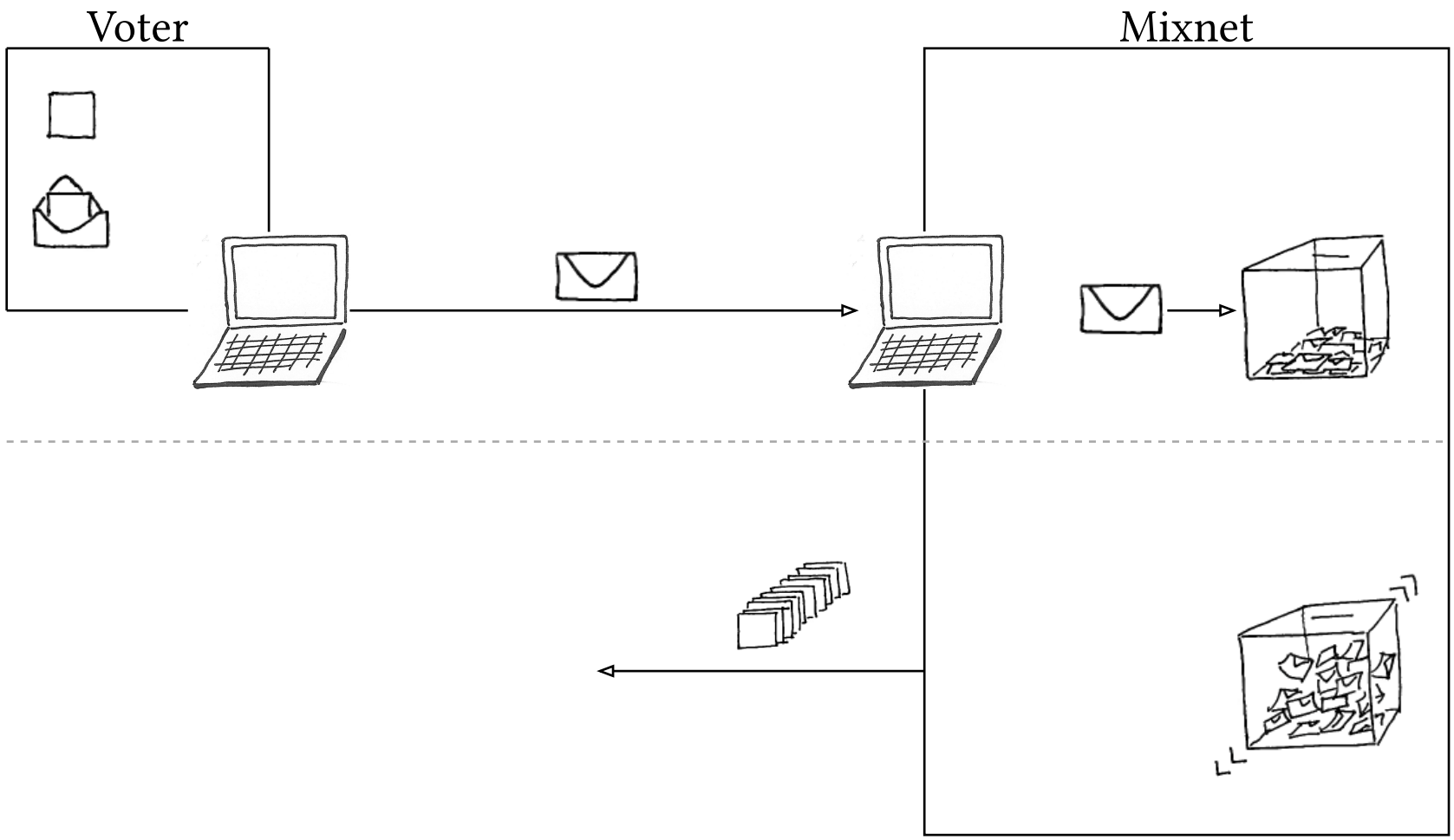
Running example: very simplified voting protocol



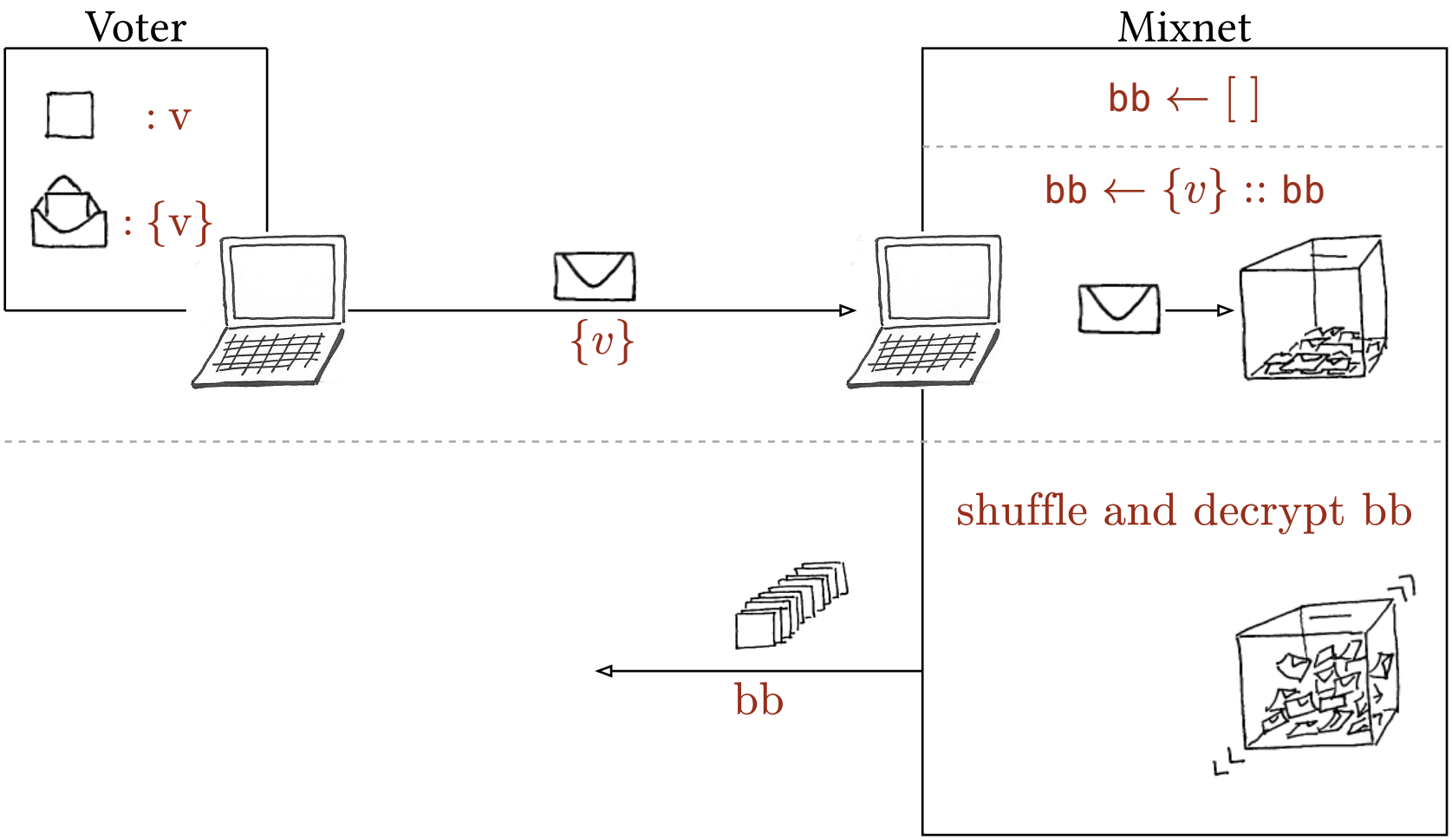
Running example: very simplified voting protocol



Running example: very simplified voting protocol



Running example: very simplified voting protocol



Running example: attacker model and privacy modeling

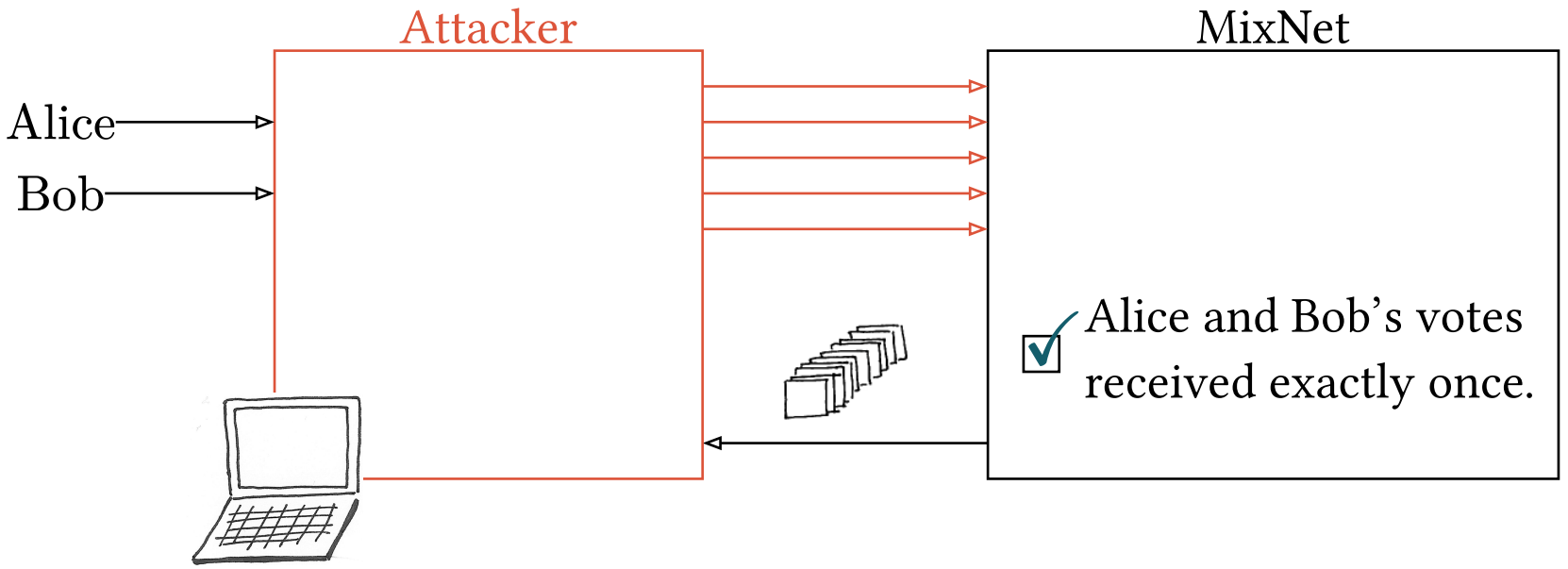
Privacy

No link possible between voters and votes.

Running example: attacker model and privacy modeling

Privacy

No link possible between voters and votes.



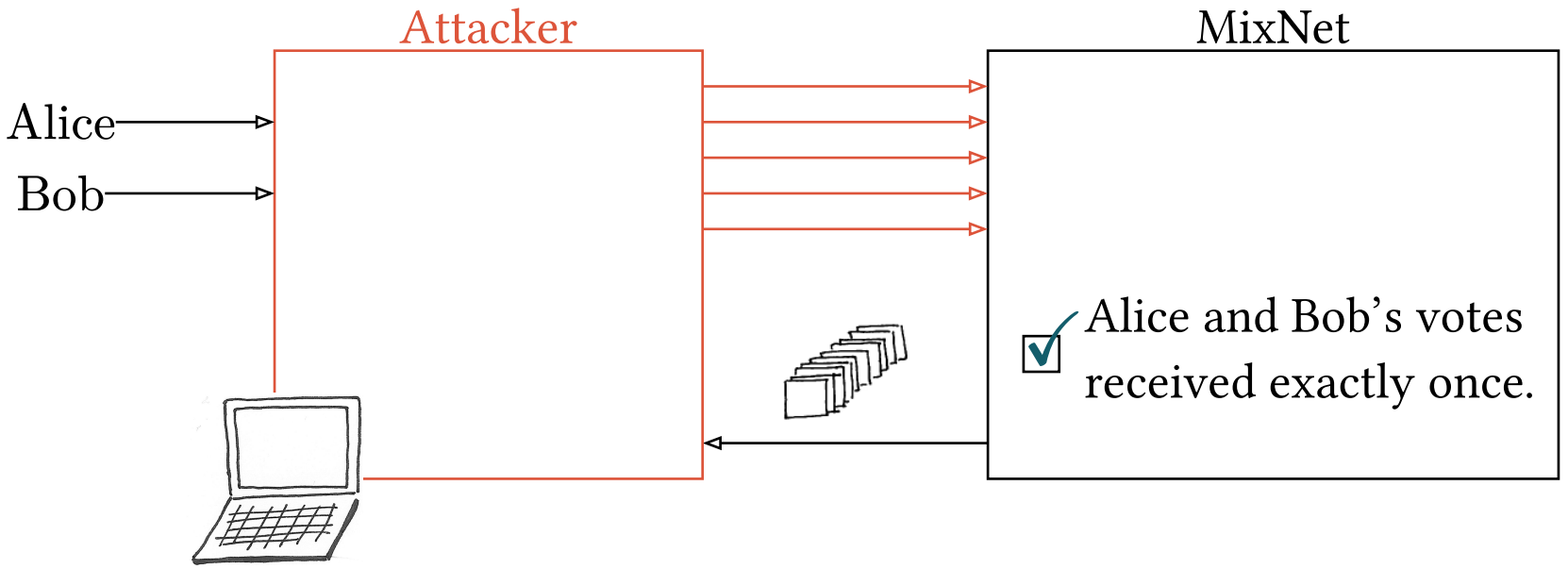
Alice: v_0

Bob: v_1

Running example: attacker model and privacy modeling

Privacy

No link possible between voters and votes.



Alice: v_0

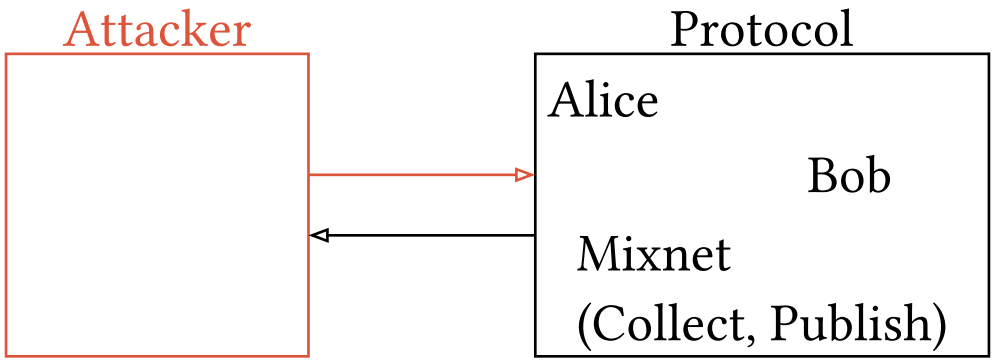
Bob: v_1

$\sim$

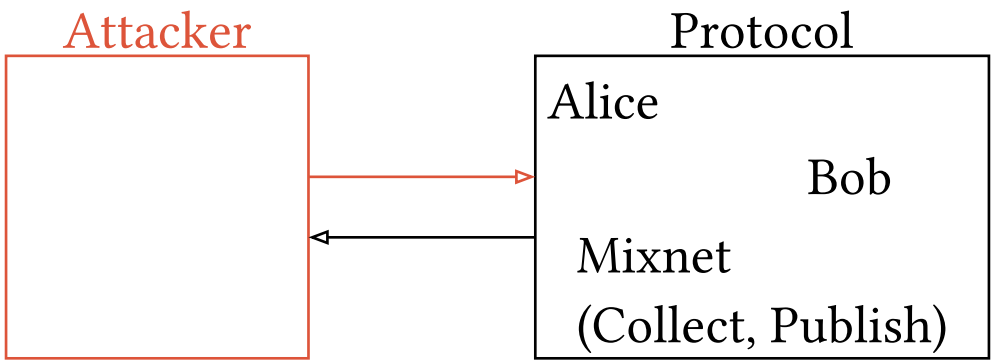
Alice: v_1

Bob: v_0

Running example: proof



Running example: proof



Protocol equivalence: indistinguishability from the attacker's POV

$A : \{ v_0 \}$
$B : \{ v_1 \}$
$P : \text{shuffle}([\ldots, v_0, \ldots, \text{dec}\, v\, \mathsf{sk}, \ldots, v_1, \ldots])$

$\sim$

$A : \{ v_1 \}$
$B : \{ v_0 \}$
$P : \text{shuffle}([\ldots, v_1, \ldots, \text{dec}\, v\, \mathsf{sk}, \ldots, v_0, \ldots])$

Cryptographic proofs

Goal: Formal proof of security

1. **Formal model**: protocol, attackers,...
2. **Cryptographic arguments**: specific proof techniques

Cryptographic proofs

Goal: Formal proof of security

1. **Formal model**: protocol, attackers,...
2. **Cryptographic arguments**: specific proof techniques

Thesis' framework

- Higher-order CCSA logic (Computationally Complete Symbolic Attacker)
  - Terms: computations (messages (ex: $\{v_0\}$, ...))
  - Formula $t_0 \sim t_1$: computational indistinguishability
  - Inference rules: cryptographic arguments

Cryptographic proofs

Goal: Formal proof of security

1. **Formal model**: protocol, attackers,...
2. **Cryptographic arguments**: specific proof techniques

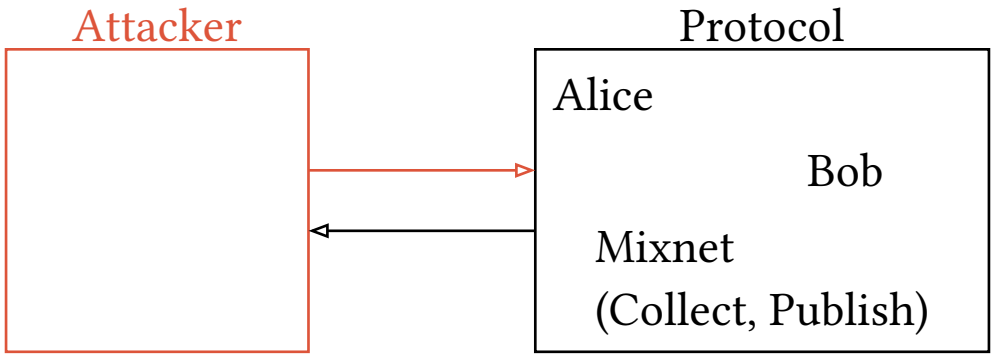
Thesis' framework

- Higher-order CCSA logic (Computationally Complete Symbolic Attacker)
  - Terms: computations (messages (ex: $\{v_0\}$, ...))
  - Formula $t_0 \sim t_1$: computational indistinguishability
  - Inference rules: cryptographic arguments

Computer-aided cryptography

- Mechanized proofs: high confidence
- For CCSA: the Squirrel proof assistant

Running example: proof

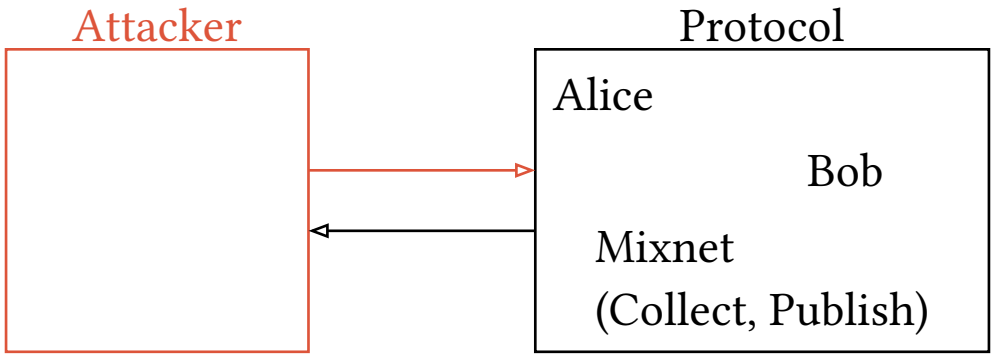


$A : \{v_0\}$
$B : \{v_1\}$
$P : \text{shuffle}([..., v_0, ..., \text{dec}\, v\, \textsf{sk}, ..., v_1, ...])$

$\sim$

$A : \{v_1\}$
$B : \{v_0\}$
$P : \text{shuffle}([..., v_1, ..., \text{dec}\, v\, \textsf{sk}, ..., v_0, ...])$

Running example: proof



$A : \{v_0\}$
$B : \{v_1\}$
$P : \text{shuffle}([\ldots, v_0, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_1, \ldots])$

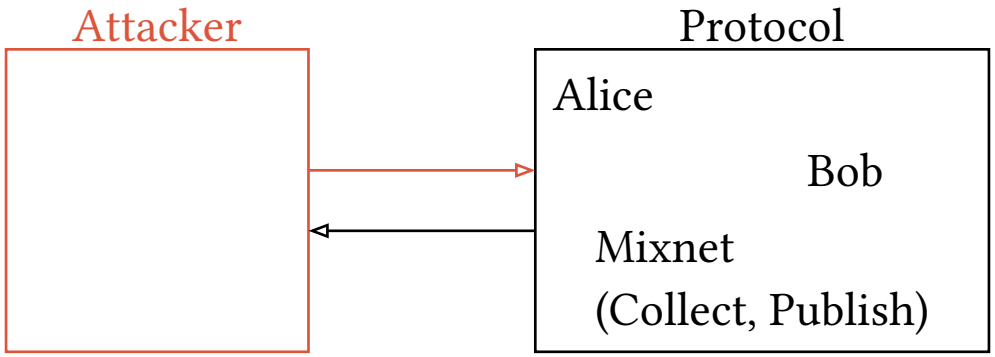
$\sim$

$A : \{v_1\}$
$B : \{v_0\}$
$P : \text{shuffle}([\ldots, v_1, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_0, \ldots])$

1: $\{m\}$ hides m

$A : \{0\}$
$B : \{0\}$
$P : \text{shuffle}([\ldots, v_0, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_1, \ldots])$

Running example: proof



$A : \{v_0\}$
$B : \{v_1\}$
$P : \text{shuffle}([..., v_0, ..., \text{dec}\, v\, \textsf{sk}, ..., v_1, ...])$

$\sim$

$A : \{v_1\}$
$B : \{v_0\}$
$P : \text{shuffle}([..., v_1, ..., \text{dec}\, v\, \textsf{sk}, ..., v_0, ...])$

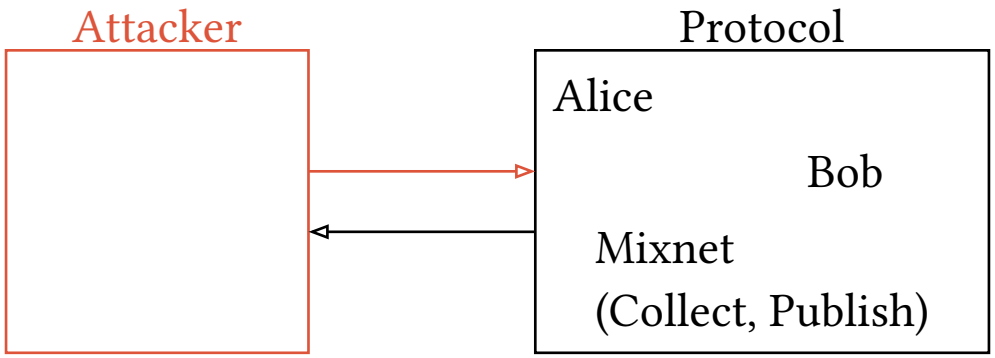
1: $\{m\}$ hides m

2: $\{m\}$ hides m

$A : \{0\}$
$B : \{0\}$
$P : \text{shuffle}([..., v_0, ..., \text{dec}\, v\, \textsf{sk}, ..., v_1, ...])$

$A : \{0\}$
$B : \{0\}$
$P : \text{shuffle}([..., v_1, ..., \text{dec}\, v\, \textsf{sk}, ..., v_0, ...])$

Running example: proof



$A : \{v_0\}$
$B : \{v_1\}$
$P : \text{shuffle}([\ldots, v_0, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_1, \ldots])$

$\sim$

$A : \{v_1\}$
$B : \{v_0\}$
$P : \text{shuffle}([\ldots, v_1, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_0, \ldots])$

1: $\{m\}$ hides m

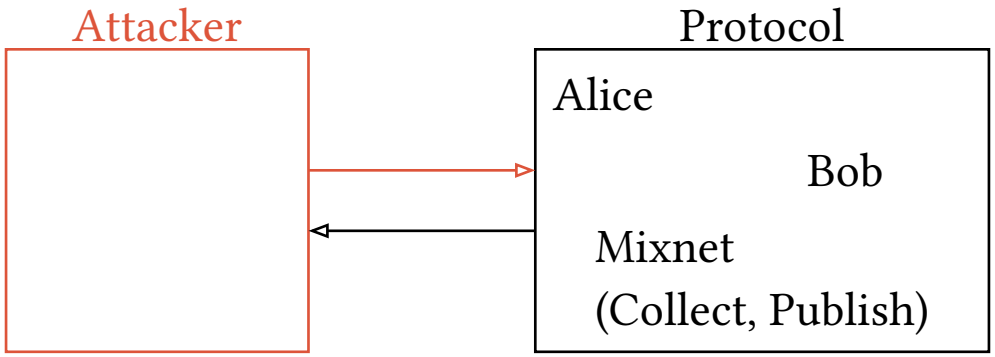
2: $\{m\}$ hides m

$A : \{0\}$
$B : \{0\}$
$P : \text{shuffle}([\ldots, v_0, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_1, \ldots])$

3: shuffle property

$A : \{0\}$
$B : \{0\}$
$P : \text{shuffle}([\ldots, v_1, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_0, \ldots])$

Running example: proof



$A : \{v_0\}$
$B : \{v_1\}$
$P : \text{shuffle}([\ldots, v_0, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_1, \ldots])$

$\sim$

$A : \{v_1\}$
$B : \{v_0\}$
$P : \text{shuffle}([\ldots, v_1, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_0, \ldots])$

Cryptographic reduction

1: $\{m\}$ hides m

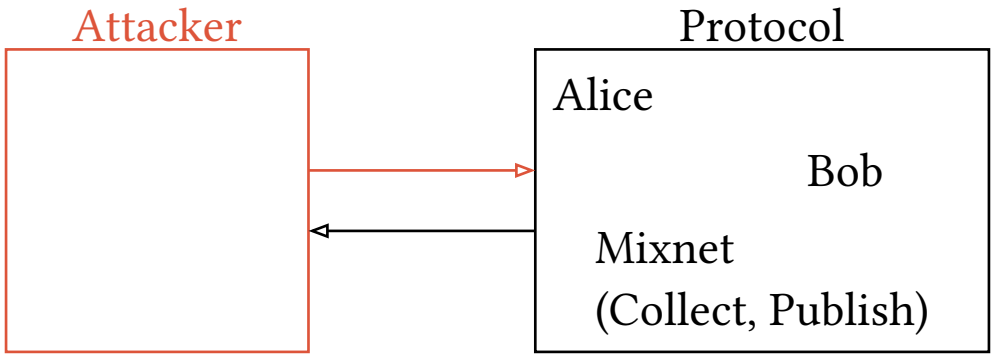
2: $\{m\}$ hides m

$A : \{0\}$
$B : \{0\}$
$P : \text{shuffle}([\ldots, v_0, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_1, \ldots])$

3: shuffle property

$A : \{0\}$
$B : \{0\}$
$P : \text{shuffle}([\ldots, v_1, \ldots, \text{dec}\, v\ \mathsf{sk}, \ldots, v_0, \ldots])$

Running example: proof



$A : \{v_0\}$
$B : \{v_1\}$
$P : \text{shuffle}([\ldots, v_0, \ldots, \text{dec}\, v\, \mathsf{sk}, \ldots, v_1, \ldots])$

$\sim$

$A : \{v_1\}$
$B : \{v_0\}$
$P : \text{shuffle}([\ldots, v_1, \ldots, \text{dec}\, v\, \mathsf{sk}, \ldots, v_0, \ldots])$

Cryptographic reduction

1: $\{m\}$ hides m

2: $\{m\}$ hides m

$A : \{0\}$
$B : \{0\}$
$P : \text{shuffle}([\ldots, v_0, \ldots, \text{dec}\, v\, \mathsf{sk}, \ldots, v_1, \ldots])$

3: shuffle property

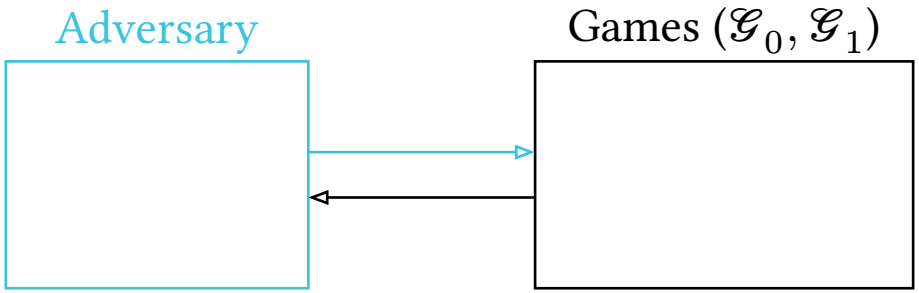
$A : \{0\}$
$B : \{0\}$
$P : \text{shuffle}([\ldots, v_1, \ldots, \text{dec}\, v\, \mathsf{sk}, \ldots, v_0, \ldots])$

Cryptographic reduction

- from a cryptographic assumption: $\{m\}$ hides m
- to a protocol equivalence.

IND-CCA2 games

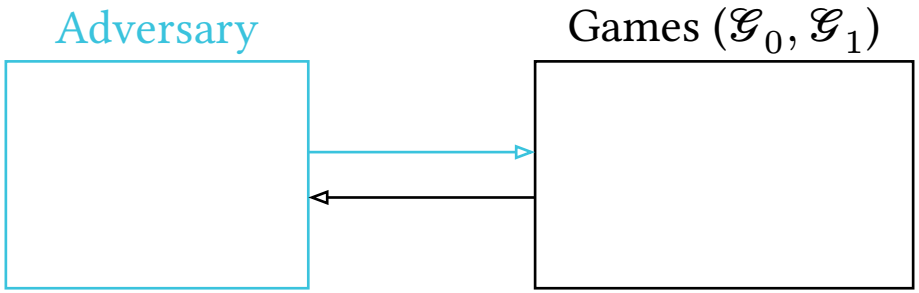
Assumption: the encryption "hides" its content.



[1]For fixed-length plain text

IND-CCA2 games

Assumption: the encryption "hides" its content.



Security parameter: size of the key η

IND-CCA2 assumption

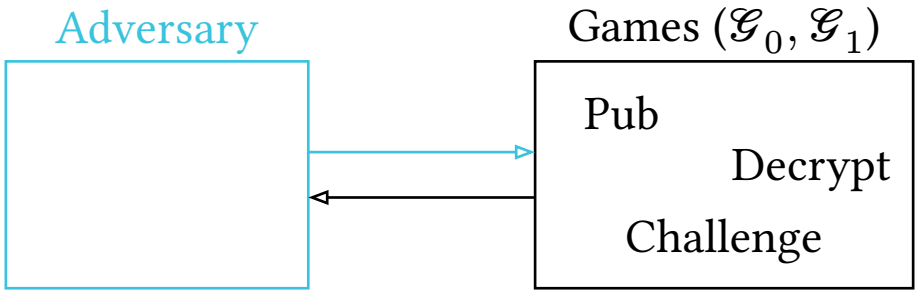
For all **polynomial-time** adversaries $\mathcal{A}$:

$$|\Pr(\mathcal{A}^{\mathcal{G}_0} = 1) - \Pr(\mathcal{A}^{\mathcal{G}_1} = 1)| < \text{negl}(\eta)$$

[1]For fixed-length plain text

IND-CCA2 games

Assumption: the encryption "hides" its content.



Security parameter: size of the key η

IND-CCA2 assumption

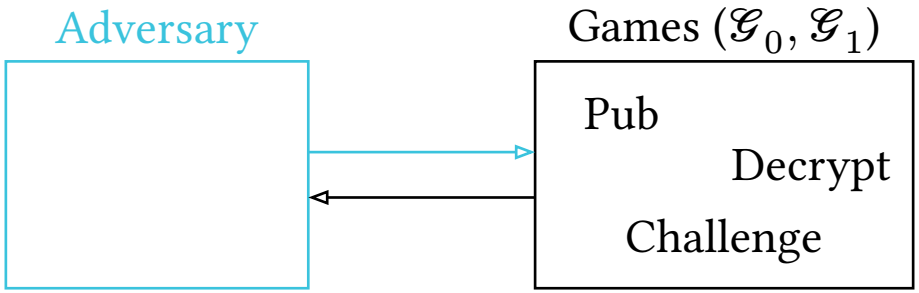
For all **polynomial-time** adversaries $\mathcal{A}$:

$$|\Pr(\mathcal{A}^{\mathcal{G}_0} = 1) - \Pr(\mathcal{A}^{\mathcal{G}_1} = 1)| < \text{negl}(\eta)$$

[1]For fixed-length plain text

IND-CCA2 games

Assumption: the encryption "hides" its content.



Security parameter: size of the key η

IND-CCA2 assumption

For all **polynomial-time** adversaries $\mathcal{A}$:

$$|\Pr(\mathcal{A}^{\mathcal{G}_0} = 1) - \Pr(\mathcal{A}^{\mathcal{G}_1} = 1)| < \text{negl}(\eta)$$

IND-CCA2 games[1]

$\mathcal{G}_0$: key $\xleftarrow{\$}$

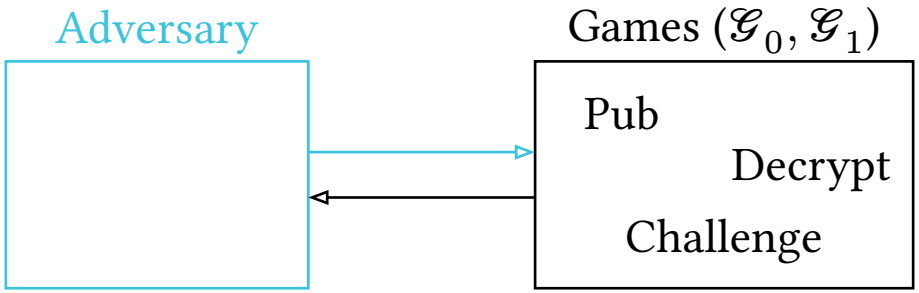
Challenge(m_0, m_1) :=
r $\xleftarrow{\$}$

return $\{ \quad m_0 \quad \}^{\text{r}}_{\text{pk(key)}}$

[1]For fixed-length plain text

IND-CCA2 games

Assumption: the encryption "hides" its content.



Security parameter: size of the key η

IND-CCA2 assumption

For all **polynomial-time** adversaries $\mathcal{A}$:

$$|\Pr(\mathcal{A}^{\mathcal{G}_0} = 1) - \Pr(\mathcal{A}^{\mathcal{G}_1} = 1)| < \text{negl}(\eta)$$

IND-CCA2 games[1]

$\mathcal{G}_1$: key $\xleftarrow{\$}$

Challenge(m_0, m_1) :=

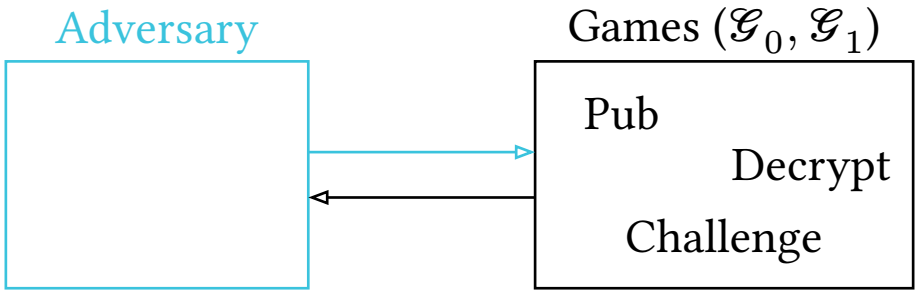
r $\xleftarrow{\$}$

return $\{ \quad m_1 \}^{\text{r}}_{\text{pk(key)}}$

[1]For fixed-length plain text

IND-CCA2 games

Assumption: the encryption "hides" its content.



Security parameter: size of the key η

IND-CCA2 assumption

For all **polynomial-time** adversaries $\mathcal{A}$:

$$|\Pr(\mathcal{A}^{\mathcal{G}_0} = 1) - \Pr(\mathcal{A}^{\mathcal{G}_1} = 1)| < \text{negl}(\eta)$$

IND-CCA2 games[1]

$\#(\mathcal{G}_0, \mathcal{G}_1)$: $\quad \text{key} \xleftarrow{\$}$

Challenge$(m_0, m_1) :=$

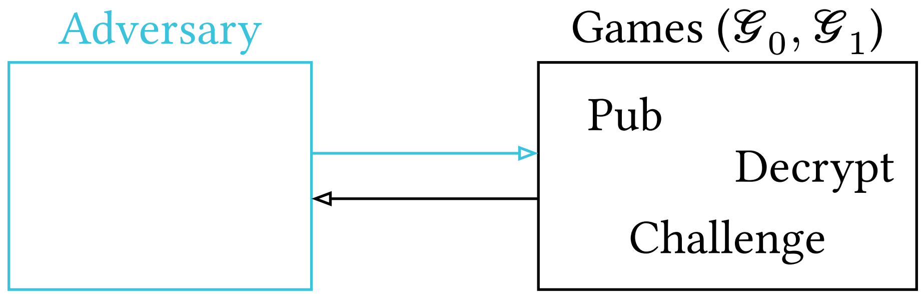
$\text{r} \xleftarrow{\$}$

return $\{\#(m_0, m_1)\}^{\text{r}}_{\text{pk(key)}}$

[1] For fixed-length plain text

IND-CCA2 games

Assumption: the encryption "hides" its content.



Security parameter: size of the key η

IND-CCA2 assumption

For all **polynomial-time** adversaries $\mathcal{A}$:

$$|\Pr(\mathcal{A}^{\mathscr{G}_0} = 1) - \Pr(\mathcal{A}^{\mathscr{G}_1} = 1)| < \mathrm{negl}(\eta)$$

IND-CCA2 games[1]

$\#(\mathscr{G}_0, \mathscr{G}_1)$: $\mathrm{key} \xleftarrow{\$}$

Challenge$(m_0, m_1) :=$

$\mathrm{r} \xleftarrow{\$}$

Pub$() := \mathrm{pk}(\mathrm{key})$

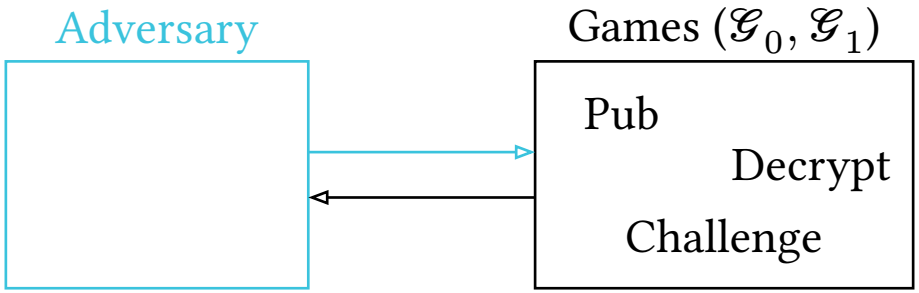
Decrypt$(v) =$ $\mathsf{dec}\, v\, \mathrm{key}$

return $\{\#(m_0, m_1)\}^{\mathrm{r}}_{\mathrm{pk(key)}}$

[1]For fixed-length plain text

IND-CCA2 games

Assumption: the encryption "hides" its content.



Security parameter: size of the key η

IND-CCA2 assumption

For all **polynomial-time** adversaries $\mathcal{A}$:

$$|\Pr(\mathcal{A}^{\mathcal{G}_0} = 1) - \Pr(\mathcal{A}^{\mathcal{G}_1} = 1)| < \text{negl}(\eta)$$

IND-CCA2 games[1]

$\#(\mathcal{G}_0, \mathcal{G}_1)$:

$\text{key} \xleftarrow{\$}$

$\log \leftarrow []$

$\textbf{Pub}() := \text{pk}(\text{key})$

$\textbf{Decrypt}(v) = \texttt{if}\ v \notin \log\ \texttt{then}\ \text{dec}\ v\ \text{key}$

$\textbf{Challenge}(m_0, m_1) :=$

$\text{r} \xleftarrow{\$}$

$\log \leftarrow \log \cup \{\#(m_0, m_1)\}^{\text{r}}_{\text{pk(key)}}$

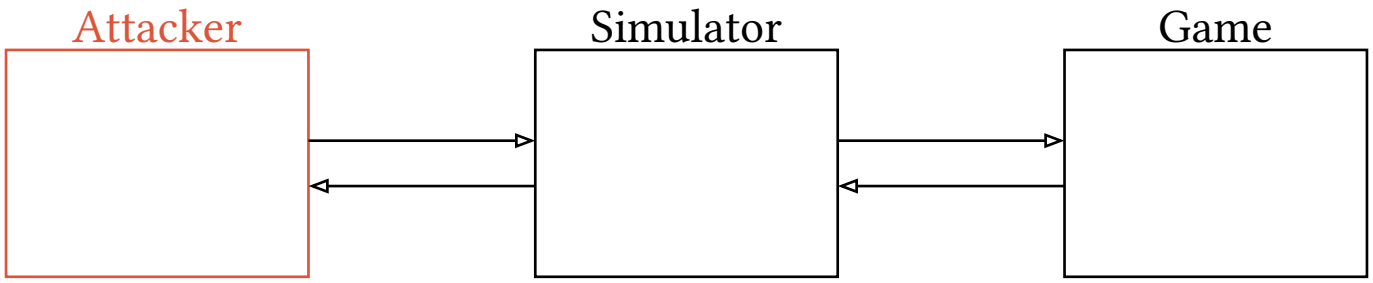
return $\{\#(m_0, m_1)\}^{\text{r}}_{\text{pk(key)}}$

[1] For fixed-length plain text

Cryptographic reduction

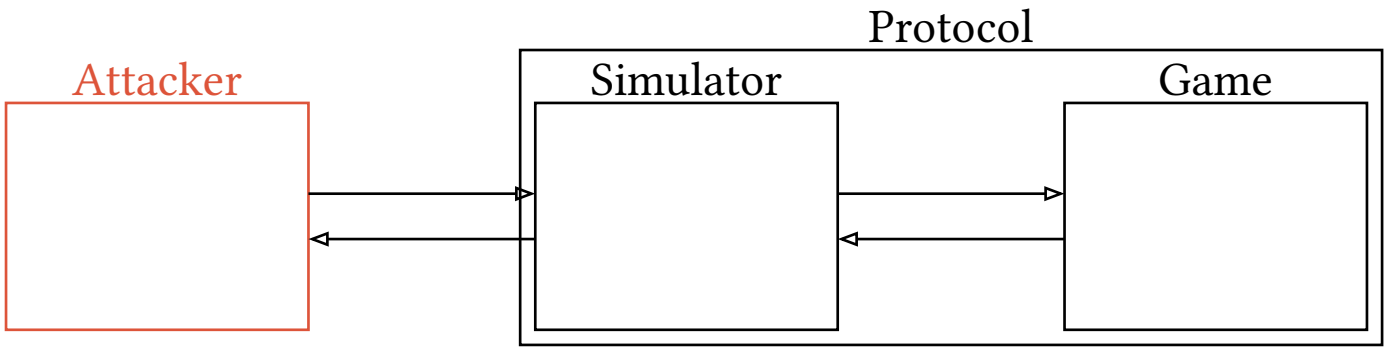
Cryptographic reduction

Step 1: Build a simulator



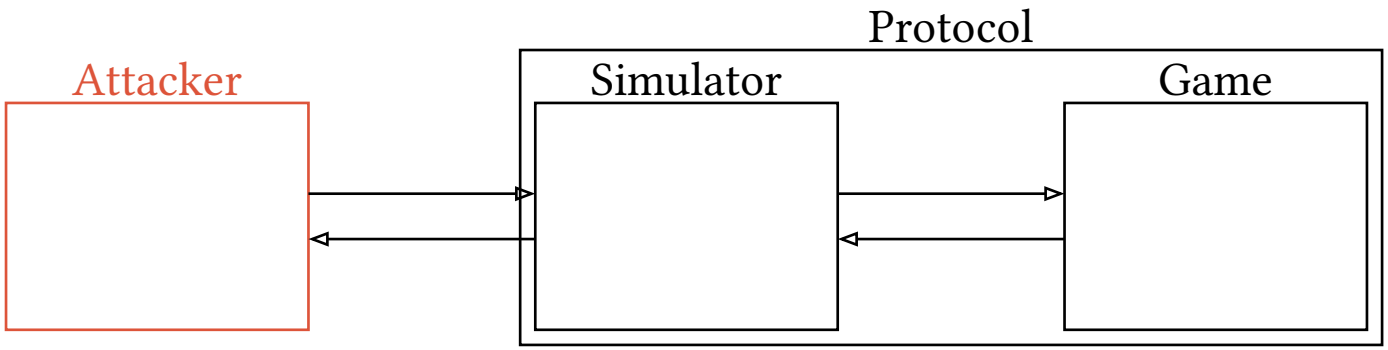
Cryptographic reduction

Step 1: Build a simulator

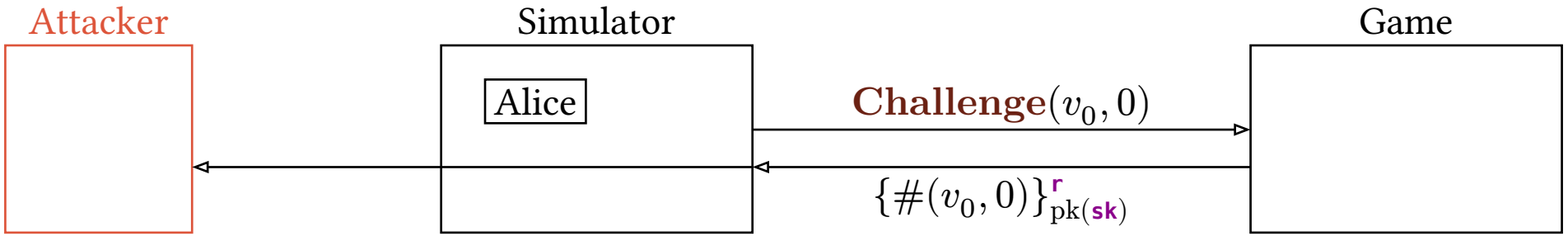


Cryptographic reduction

Step 1: Build a simulator

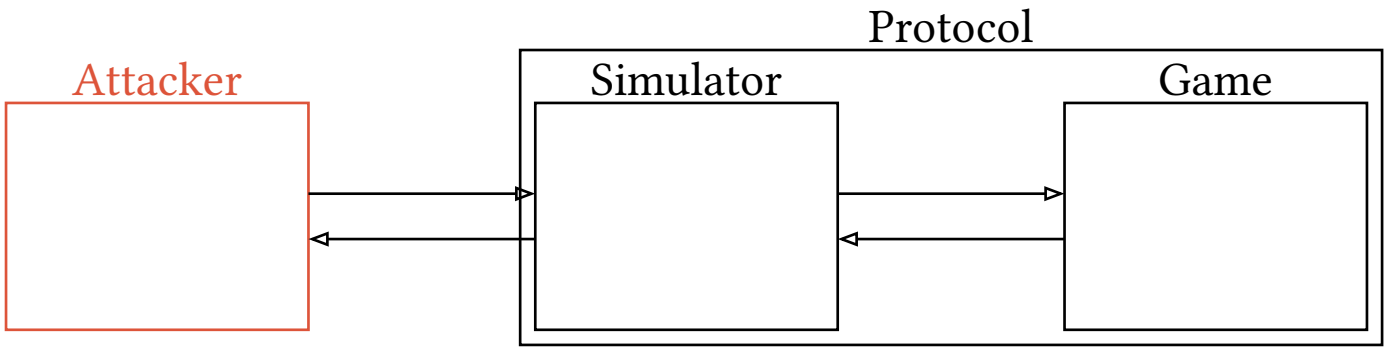


For example:

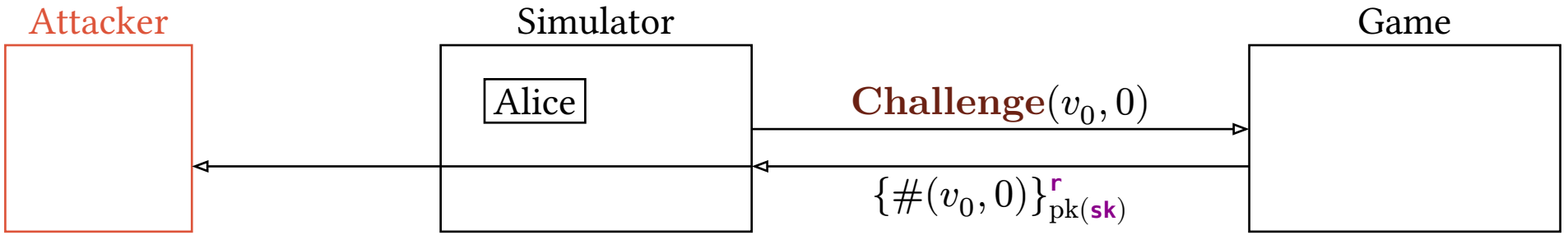


Cryptographic reduction

Step 1: Build a simulator



For example:

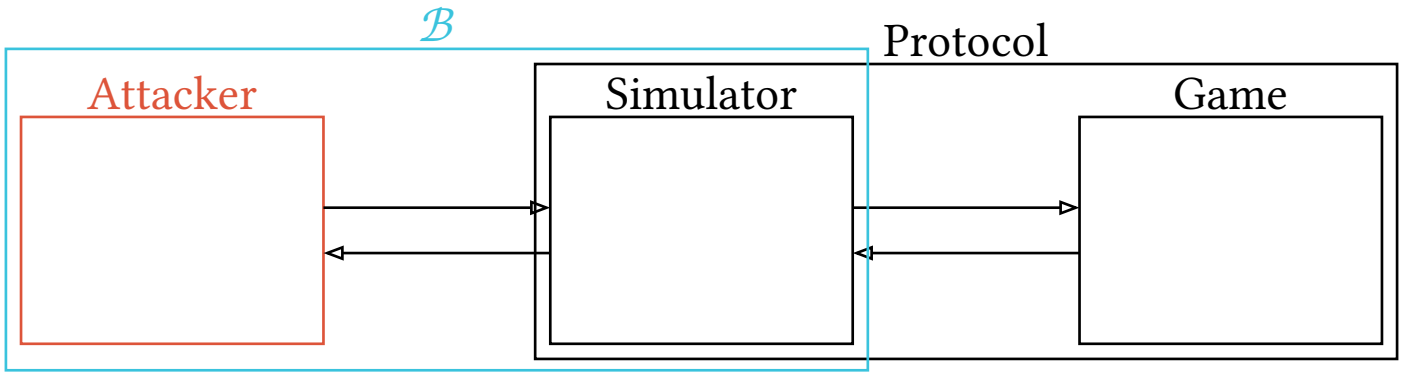


Step 2: Proof by contradiction

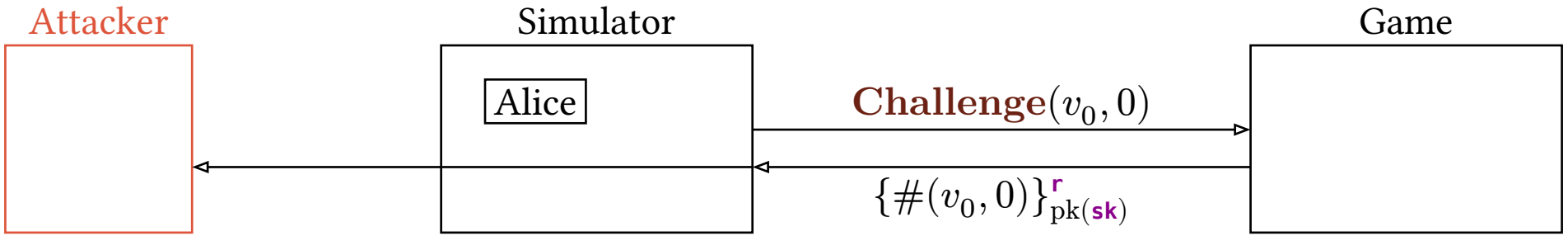
Attacker breaches security of protocol

Cryptographic reduction

Step 1: Build a simulator



For example:

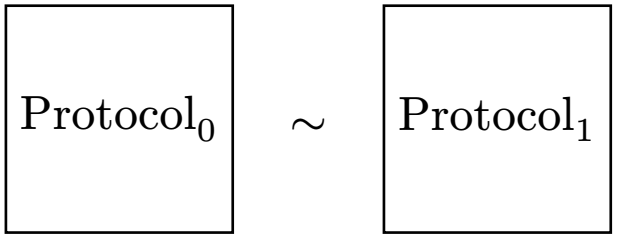


Step 2: Proof by contradiction

Attacker breaches security of protocol $\Rightarrow$ $\mathcal{B}$ breaches security of IND-CCA2.

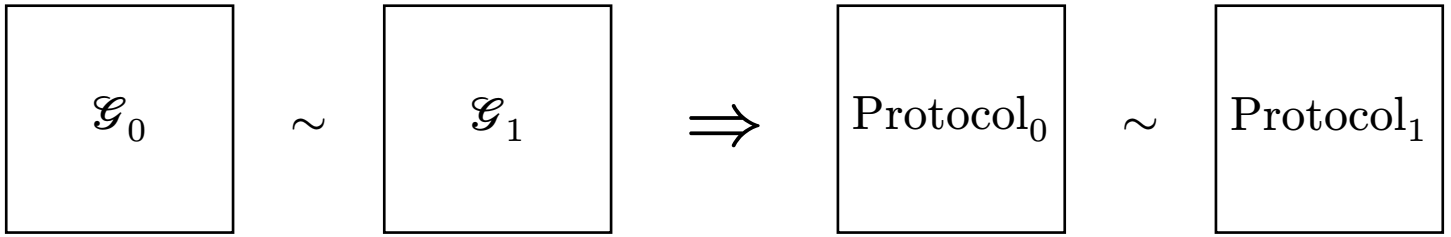
Simulator synthesis

In summary:



Simulator synthesis

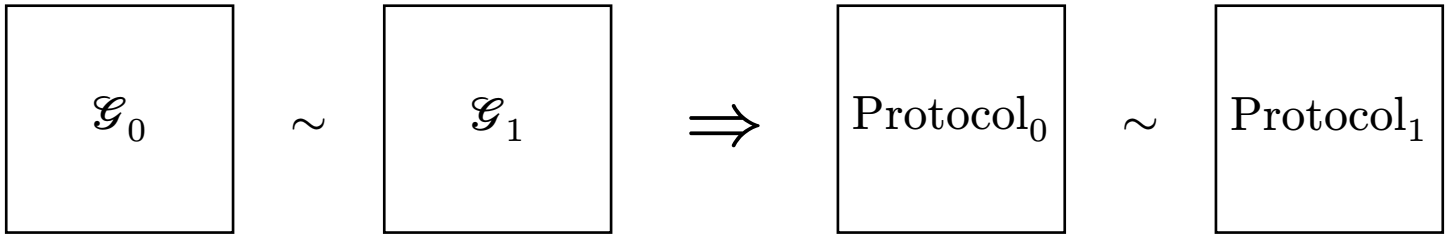
In summary:



Establishing a cryptographic reduction $\rightarrow$ solving a **simulator synthesis** problem

Simulator synthesis

In summary:



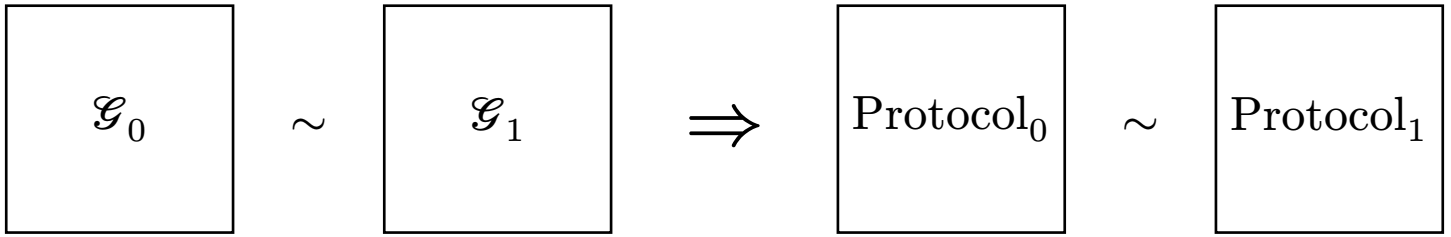
Establishing a cryptographic reduction $\rightarrow$ solving a **simulator synthesis** problem

Simulator synthesis problem

There exists $\mathcal{S}$ such that $\mathcal{S} \circ \mathcal{G} = \text{Protocol}$

Simulator synthesis

In summary:



Establishing a cryptographic reduction $\rightarrow$ solving a **simulator synthesis** problem

Simulator synthesis problem

There exists $\mathcal{S}$ such that $\mathcal{S} \circ \mathcal{G} = \text{Protocol}$

Reductions in CCSA and Squirrel

Logical rules

- Capture cryptographic assumptions as rules
- Use syntactic checks that ensure the existence of a simulator.

Example:

$$\frac{}{m, \{v\}^{\mathsf{r}}_{\mathrm{pk}(\mathsf{sk})} \sim m, \{0\}^{\mathsf{r}}_{\mathrm{pk}(\mathsf{sk})}} \text{“}\mathsf{sk}\text{ correctly used as key and }\mathsf{r}\text{ fresh in }m\text{ an }v\text{”}$$

Reductions in CCSA and Squirrel

Logical rules

- Capture cryptographic assumptions as rules
- Use syntactic checks that ensure the existence of a simulator.

Example:

$$\frac{}{m, \{v\}^{\mathsf{r}}_{\mathrm{pk}(\mathsf{sk})} \sim m, \{0\}^{\mathsf{r}}_{\mathrm{pk}(\mathsf{sk})}} \text{ “}\mathsf{sk}\text{ correctly used as key and }\mathsf{r}\text{ fresh in } m \text{ an } v\text{”}$$

Squirrel

Implement crypto rules as domain-specific tactic (CPA, EUF-CMA, etc)

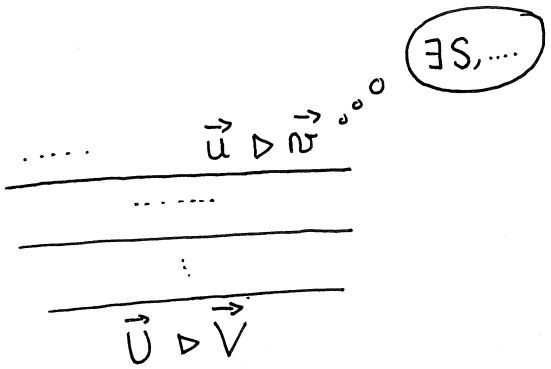
Limit generality and extensibility of Squirrel:

- manually design and prove each new rule
- implement each new rule in **Squirrel**

$\Longrightarrow$ out of reach for standard users and error-prone.

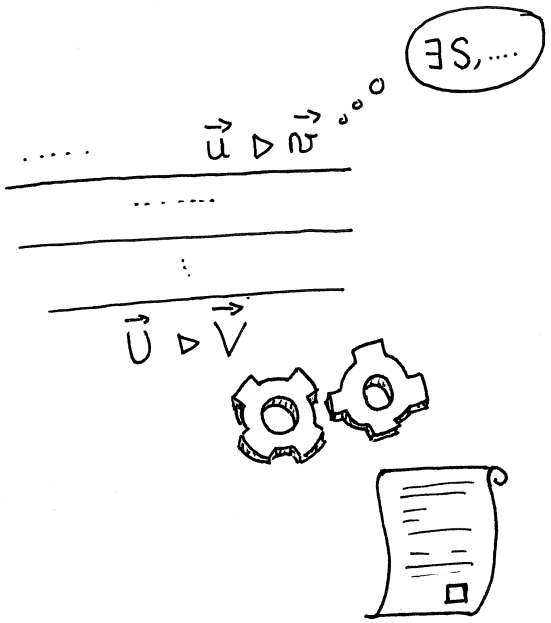
Contributions

- Logical framework for **simulator synthesis**
  - *For arbitrary games*
  - Language for adversaries, games, etc.
  - Bideduction predicate
  - Proof system
-
-
  -



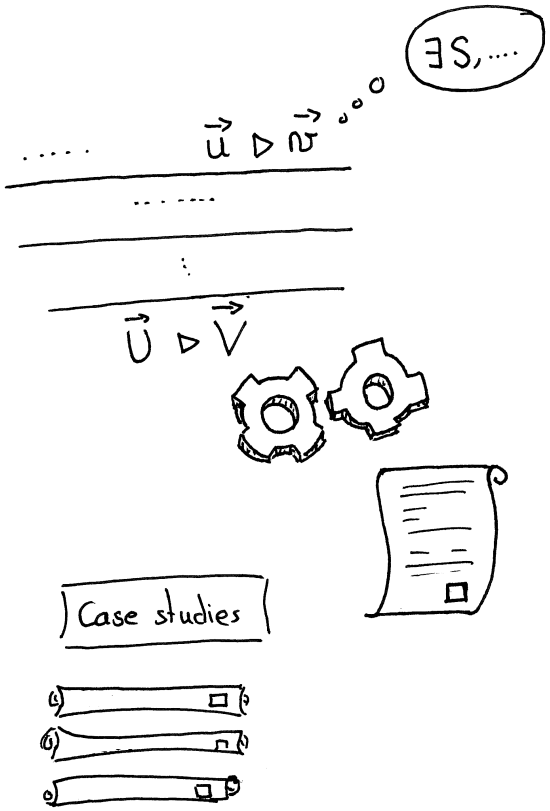
Contributions

- Logical framework for **simulator synthesis**
  - *For arbitrary games*
  - Language for adversaries, games, etc.
  - Bideduction predicate
  - Proof system
- Proof mechanization.
- ▸
  - ▸



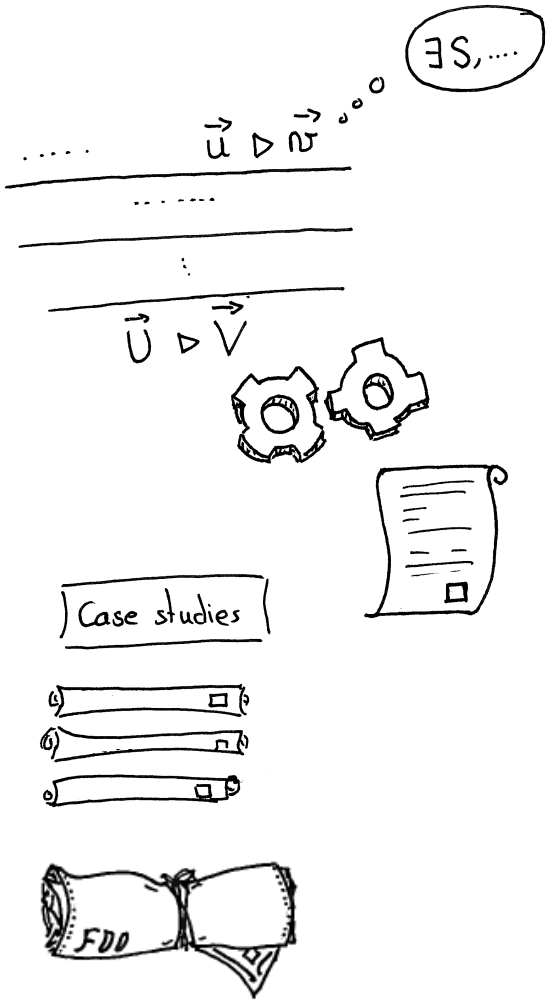
Contributions

- Logical framework for **simulator synthesis**
  - *For arbitrary games*
  - Language for adversaries, games, etc.
  - Bideduction predicate
  - Proof system
- Proof mechanization.
- Implementation in Squirrel.
  - Validation through case studies
  -



Contributions

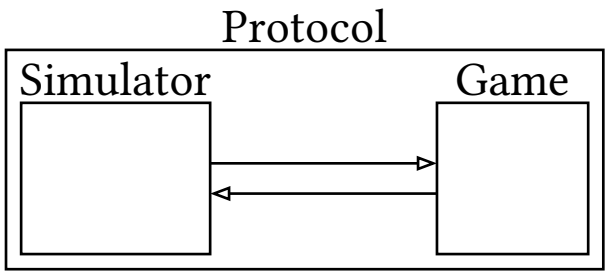
- Logical framework for **simulator synthesis**
  - *For arbitrary games*
  - Language for adversaries, games, etc.
  - Bideduction predicate
  - Proof system
- Proof mechanization.
- Implementation in Squirrel.
  - Validation through case studies
  - Large case study: the proof of vote privacy for FOO e-voting protocol.



3. Bideduction

Starting point

Objectives

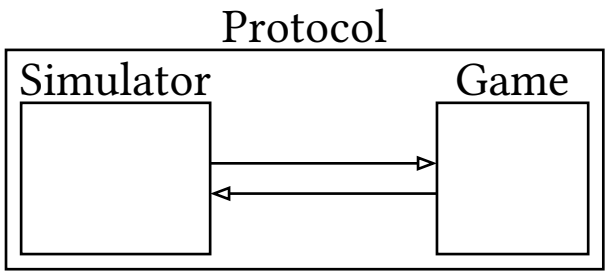


$$\exists \mathcal{S} \text{ s.t. } \mathcal{S} \circ \mathcal{G} = P$$

Starting point

Objectives

- Predicate to capture the existence of a simulator (**step 1**)



$$\exists \mathcal{S} \text{ s.t. } \mathcal{S} \circ \mathcal{G} = P$$

Starting predicate

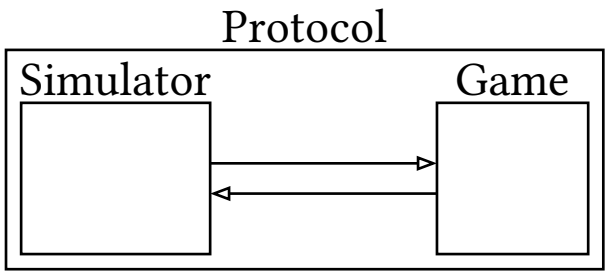
$$\vdash \#(\vec{u_0}, \vec{u_1}) \rhd \#(\vec{v_0}, \vec{v_1})$$

There exists a (polynomial-time) simulator $\mathcal{S}$ such that: $\mathcal{S}^{\mathcal{G}_0}(\vec{u_0}) = \vec{v_0}$ and $\mathcal{S}^{\mathcal{G}_1}(\vec{u_1}) = \vec{v_1}$.

Starting point

Objectives

- Predicate to capture the existence of a simulator (**step 1**)



$$\exists \mathcal{S} \text{ s.t. } \mathcal{S} \circ \mathcal{G} = P$$

Starting predicate

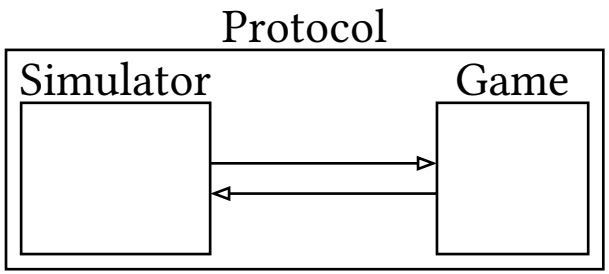
$$\vdash \#(\vec{u_0}, \vec{u_1}) \rhd \#(\vec{v_0}, \vec{v_1})$$

There exists a (polynomial-time) simulator $\mathcal{S}$ such that: $\mathcal{S}^{\mathcal{G}_0}(\vec{u_0}) = \vec{v_0}$ and $\mathcal{S}^{\mathcal{G}_1}(\vec{u_1}) = \vec{v_1}$.

Starting point

Objectives

- Predicate to capture the existence of a simulator (**step 1**)
- Proof system: proof rule = simulator building block



$$\exists \mathcal{S} \text{ s.t. } \mathcal{S} \circ \mathcal{G} = P$$

Starting predicate

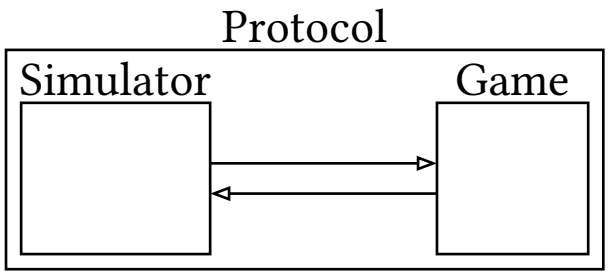
$$\vdash \#(\vec{u_0}, \vec{u_1}) \triangleright \#(\vec{v_0}, \vec{v_1})$$

There exists a (polynomial-time) simulator $\mathcal{S}$ such that: $\mathcal{S}^{\mathcal{G}_0}(\vec{u_0}) = \vec{v_0}$ and $\mathcal{S}^{\mathcal{G}_1}(\vec{u_1}) = \vec{v_1}$.

Starting point

Objectives

- Predicate to capture the existence of a simulator (**step 1**)
- Proof system: proof rule = simulator building block
- Rule to capture cryptographic reduction argument (**step 2**)



$$\exists \mathcal{S} \text{ s.t. } \mathcal{S} \circ \mathcal{G} = P$$

Starting predicate

$$\vdash \#(\vec{u_0}, \vec{u_1}) \triangleright \#(\vec{v_0}, \vec{v_1})$$

There exists a (polynomial-time) simulator $\mathcal{S}$ such that: $\mathcal{S}^{\mathcal{G}_0}(\vec{u_0}) = \vec{v_0}$ and $\mathcal{S}^{\mathcal{G}_1}(\vec{u_1}) = \vec{v_1}$.

Bideduce

$$\frac{\vdash \emptyset \triangleright \#(\vec{v_0}, \vec{v_1})}{v_0 \sim v_1}$$

Intuition: "If there exists such a simulator then v_0 and v_1 are indistinguishable **because $\mathcal{G}_0$ and $\mathcal{G}_1$ are indistinguishable**"

Bideduction rules

What can a simulator do ?

Simulator = probabilistic program with oracle calls.

Bideduction rules

What can a simulator do ?

Simulator = probabilistic program with oracle calls.

Function application

Bideduction rules

What can a simulator do ?

Simulator = probabilistic program with oracle calls.

Function application

If:

- $\mathcal{S}'$ computes v from $\vec{u}$
- f available to simulators

Function application rule

$$\frac{\vdash \vec{u} \rhd v \qquad f(\_) \in \mathrm{Lib}}{}$$

Bideduction rules

What can a simulator do ?

Simulator = probabilistic program with oracle calls.

Function application

If:

- $\mathcal{S}'$ computes v from $\vec{u}$
- f available to simulators

Then:

- $\mathcal{S}$ computes $f(v)$ from $\vec{u}$

Function application rule

$$\frac{\vdash \vec{u} \rhd v \qquad f(\_) \in \mathrm{Lib}}{\vdash \vec{u} \rhd f(v)}$$

Bideduction rules

What can a simulator do ?

Simulator = probabilistic program with oracle calls.

Function application

If:

- $\mathcal{S}'$ computes v from $\vec{u}$
- f available to simulators

Then:

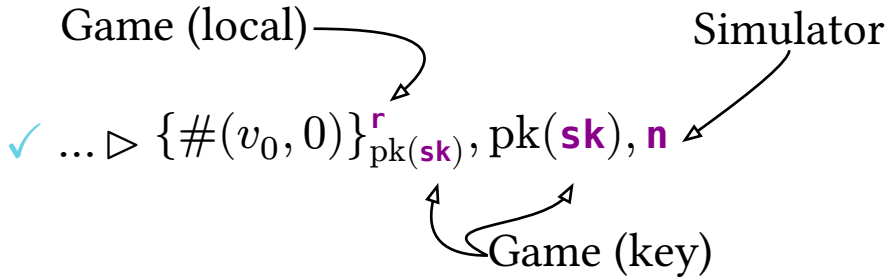
- $\mathcal{S}$ computes $f(v)$ from $\vec{u}$

Function application rule

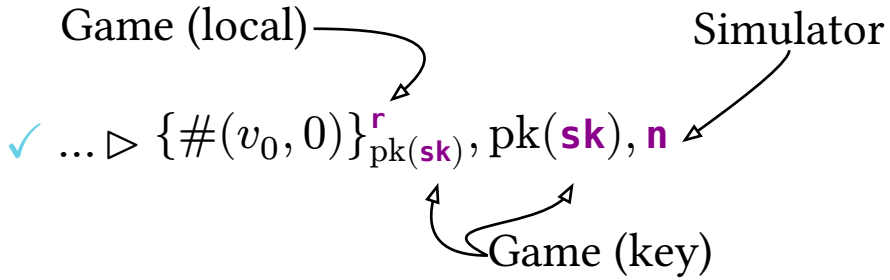
$$\frac{\vdash \vec{u} \rhd v \qquad f(\_) \in \text{Lib}}{\vdash \vec{u} \rhd f(v)}$$

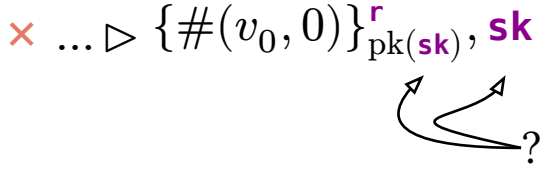
```
S(u):= v <- S'(u)
       return f(v)
```

Bideduction rules

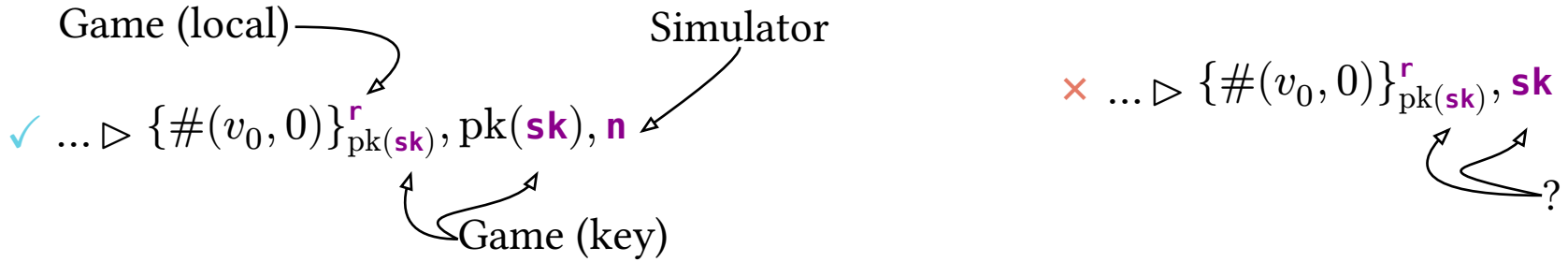


Bideduction rules





Bideduction rules



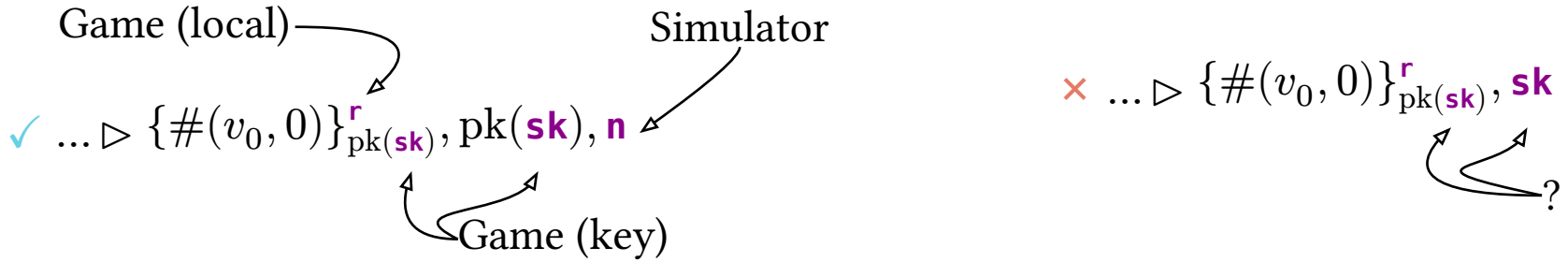
Constraint system

Register randomness ownership.

Tags: S, G.local, G.key,...

Constraint: (n, T)

Bideduction rules



Constraint system

Register randomness ownership.

Tags: S, G.local, G.key,...

Constraint: (n, T)

Random sampling

Sampling rule on example

$$\frac{}{(\mathsf{n}, \mathrm{S}) \vdash \vec{u} \rhd \mathsf{n}}$$

```
S := sample()
```

Bideduction rules

Oracle calls: challenge

Bideduction rules

Oracle calls: challenge

$$\textbf{Challenge}(m_0, m_1) :=$$
$$\mathrm{r} \xleftarrow{\$}$$
$$\log += \{\#(m_0, m_1)\}^{\mathrm{r}}_{\mathrm{pk(key)}}$$
$$\text{return } \{\#(m_0, m_1)\}^{\mathrm{r}}_{\mathrm{pk(key)}}$$

Bideduction rules

Oracle calls: challenge

$$\begin{array}{l}\textbf{Challenge}(m_0, m_1) := \\ \quad \mathrm{r} \xleftarrow{\$} \\ \quad \mathrm{log} += \{\#(m_0, m_1)\}^{\mathrm{r}}_{\mathrm{pk(key)}} \\ \quad \text{return } \{\#(m_0, m_1)\}^{\mathrm{r}}_{\mathrm{pk(key)}}\end{array}$$

Oracle call on encryption

$$\frac{\mathcal{C} \vdash \vec{u} \rhd v_0, 0}{\mathcal{C}.(\mathtt{r} : \mathrm{G.local}).(\mathtt{sk} : \mathrm{G.key}) \vdash \vec{u} \rhd \{\#(v_0, 0)\}^{\mathtt{r}}_{\mathrm{pk}(\mathtt{sk})}}$$

```
S := v0,z <- S'
     (*oracle call*)
     o <- Challenge v0 z
     return o
```

Semantical challenges

Restrictions on randomness usage:

Semantical challenges

Restrictions on randomness usage:

- respect randomness ownership (games v.s. simulator)
- single (`sk`, G.key) for the game's key
- `r` is local
- ...

Semantical challenges

Restrictions on randomness usage:

- respect randomness ownership (games v.s. simulator)
- single (`sk`, G.key) for the game's key
- `r` is local
- ...

Constraint validity

Logical formula $\mathrm{Valid}(\mathcal{C})$ that ensures correct randomness handling

Semantical challenges

Restrictions on randomness usage:

- respect randomness ownership (games v.s. simulator)
- single (sk, G.key) for the game's key
- r is local
- ...

Constraint validity

Logical formula $\mathbf{Valid}(\mathcal{C})$ that ensures correct randomness handling

Bideduce

$$\frac{\vdash \emptyset \rhd \#(\vec{v_0}, \vec{v_1}) \qquad \mathbf{Valid}(\mathcal{C})}{v_0 \sim v_1}$$

Bideduction rules

Oracle call: decryption

$$\mathbf{Decrypt}(v) = \texttt{if}\ v \notin \log\ \texttt{then}\ \mathsf{dec}\, v\, \mathrm{key}$$

Bideduction rules

Oracle call: decryption

$$\mathbf{Decrypt}(v) = \texttt{if}\ v \notin \log\ \texttt{then}\ \mathsf{dec}\, v\, \mathrm{key}$$

$$\frac{\mathcal{C} \vdash \vec{u} \rhd v}{\mathcal{C}.(\texttt{sk} : \mathrm{G.key}) \vdash \vec{u} \rhd \mathsf{dec}\, v\, \texttt{sk}}$$

Bideduction rules

Oracle call: decryption

$$\textbf{Decrypt}(v) = \texttt{if}\ v \notin \log\ \texttt{then}\ \mathsf{dec}\, v\, \mathrm{key}$$

$$\frac{\mathcal{C} \vdash \vec{u} \rhd v}{\mathcal{C}.(\texttt{sk} : \mathrm{G.key}) \vdash \vec{u} \rhd \mathsf{dec}\, v\, \texttt{sk}}$$

How do we verify $v \notin \log$?

Solution: Hoare-style pre and post conditions on **game**'s memory.

Tracking memory

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

Pre-condition

Post-condition

Tracking memory

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

Pre-condition — Post-condition

Example : $\psi := \log \subseteq \{\{\#(v_0, 0)\}^{\mathsf{r}}_{\mathrm{pk}(\mathsf{sk})}\}$

Oracle triples

- Predicate to track game's memory in oracle calls
- Automated verification for sets

Tracking memory

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

Pre-condition — Post-condition

Example : $\psi := \log \subseteq \{\{\#(v_0, 0)\}^{\mathsf{r}}_{\mathrm{pk}(\mathsf{sk})}\}$

Oracle triples

- Predicate to track game's memory in oracle calls
- Automated verification for sets

Oracle call on decryption

$$\frac{\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd v}{\mathcal{C}.(\mathsf{sk} : \mathrm{G.key}), (\varphi, \psi) \vdash \vec{u} \rhd \mathsf{dec}\, v\, \mathsf{sk}}$$

Tracking memory

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

Pre-condition — Post-condition

Example : $\psi := \log \subseteq \{\{\#(v_0, 0)\}^{\mathsf{r}}_{\mathrm{pk}(\mathsf{sk})}\}$

Oracle triples

- Predicate to track game's memory in oracle calls
- Automated verification for sets

Oracle call on decryption

$$\frac{\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd v \quad \{\psi\}\, \mathsf{dec}\, v\, \mathsf{sk} \leftarrow O_{\mathsf{dec}}[\mathrm{key} \mapsto \mathsf{sk}](v)\{\psi\}}{\mathcal{C}\,.(\mathsf{sk} : \mathrm{G.key}), (\varphi, \psi) \vdash \vec{u} \rhd \mathsf{dec}\, v\, \mathsf{sk}}$$

Tracking memory

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

Pre-condition — Post-condition

Example : $\psi := \log \subseteq \{\{\#(v_0, 0)\}^{\mathsf{r}}_{\mathrm{pk}(\mathsf{sk})}\}$

Oracle triples

- Predicate to track game's memory in oracle calls
- Automated verification for sets

Oracle call on decryption

$$\frac{\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd v \quad \{\psi\}\, \mathsf{dec}\, v\, \mathsf{sk} \leftarrow O_{\mathsf{dec}}[\mathrm{key} \mapsto \mathsf{sk}](v)\{\psi\}}{\mathcal{C}\,.(\mathsf{sk} : \mathrm{G.key}), (\varphi, \psi) \vdash \vec{u} \rhd \mathsf{dec}\, v\, \mathsf{sk}} \text{ when } \psi \Rightarrow v \notin \log$$

Recursive terms

Squirrel terms: recursive definitions

Recursive terms

Squirrel terms: recursive definitions

Protocol modelling

- Protocol: a sequence of messages (a `frame`)

Recursive terms

Squirrel terms: recursive definitions

Protocol modelling

- Protocol: a sequence of messages (a `frame`)
- Mutual definition of
  - `frame`: sequence of messages
  - `output`: protocol output
  - `input`: protocol input made by the attacker

Recursive terms

Squirrel terms: recursive definitions

Protocol modelling

- Protocol: a sequence of messages (a `frame`)
- Mutual definition of
  - `frame`: sequence of messages
  - `output`: protocol output
  - `input`: protocol input made by the attacker

$\texttt{frame}(t) := \langle \texttt{output}(t, \texttt{input}(t)), \texttt{frame}(\texttt{pred}(t)) \rangle$

Proving protocol equivalence: $\forall t, \ldots \vdash \emptyset \rhd \texttt{frame}(t)$

Bideduction: induction

Example:

$$f(x) := t(f(x-1))$$
$$f(0) := t_0$$

If:

- $\mathcal{S}'$ computes $f(x)$ from $f(x-1)$ and x
- $\mathcal{S}''$ computes t_0

Bideduction: induction

Example:

$$f(x) := t(f(x-1))$$
$$f(0) := t_0$$

If:

- $\mathcal{S}'$ computes $f(x)$ from $f(x-1)$ and x
- $\mathcal{S}''$ computes t_0

Induction rule

$$\frac{f(x-1), x \rhd f(x) \qquad\qquad \rhd t_0}{}$$

Bideduction: induction

Example:

$$f(x) := t(f(x-1))$$
$$f(0) := t_0$$

If:

- $\mathcal{S}'$ computes $f(x)$ from $f(x-1)$ and x
- $\mathcal{S}''$ computes t_0

Then:

- $\mathcal{S}$ computes $f(X)$ from X

Induction rule

$$\frac{f(x-1), x \rhd f(x) \qquad\qquad \rhd t_0}{}$$

Bideduction: induction

Example:

$$f(x) := t(f(x-1))$$
$$f(0) := t_0$$

If:

- $\mathcal{S}'$ computes $f(x)$ from $f(x-1)$ and x
- $\mathcal{S}''$ computes t_0

Then:

- $\mathcal{S}$ computes $f(X)$ from X

Induction rule

$$\frac{f(x-1), x \rhd f(x) \qquad\qquad \rhd t_0}{X \rhd f(X)}$$

Bideduction: induction

Example:

$$f(x) := t(f(x-1))$$
$$f(0) := t_0$$

If:

- $\mathcal{S}'$ computes $f(x)$ from $f(x-1)$ and x
- $\mathcal{S}''$ computes t_0

Then:

- $\mathcal{S}$ computes $f(X)$ from X

Induction rule

$$\frac{f(x-1), x \rhd f(x) \qquad\qquad \rhd t_0}{X \rhd f(X)}$$

```
S := f <- S''
     for x in [0,X[:
       f <- S'(f,x)
     return f
```

Bideduction: induction

Example:

$$f(x) := t(f(x-1))$$
$$f(0) := t_0$$

If:

- $\mathcal{S}'$ computes $f(x)$ from $f(x-1)$ and x
- $\mathcal{S}''$ computes t_0

Then:

- $\mathcal{S}$ computes $f(X)$ from X

Induction rule

$$\frac{f(x-1), x \rhd f(x) \qquad\qquad \rhd t_0}{X \rhd f(X)}$$

```
S := f <- S''
     for x in [0,X[:
       f <- S'(f,x)
    return f
```

Challenges

- Pre and post conditions: **memory invariant $\varphi(x)$**
- Complexity: **iteration over constant-size domain**

Bideduction summary

Bideduction predicate

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

There exists a (polynomial-time) simulator $\mathcal{S}$ such that when $\mathrm{Valid}(\mathcal{C})$:

- $\mathcal{S}$ computes $\vec{v}$ from $\vec{u}$

Bideduction summary

Bideduction predicate

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

There exists a (polynomial-time) simulator $\mathcal{S}$ such that when $\mathrm{Valid}(\mathcal{C})$:

- $\mathcal{S}$ computes $\vec{v}$ from $\vec{u}$
- the game's memory evolves from φ to ψ

Bideduction summary

Bideduction predicate

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

There exists a (polynomial-time) simulator $\mathcal{S}$ such that when $\mathrm{Valid}(\mathcal{C})$:

- $\mathcal{S}$ computes $\vec{v}$ from $\vec{u}$
- the game's memory evolves from φ to ψ

Omitted complexity

- Early-sampling semantics: simulator controls full randomness usage.

Bideduction summary

Bideduction predicate

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

There exists a (polynomial-time) simulator $\mathcal{S}$ such that when $\mathrm{Valid}(\mathcal{C})$:

- $\mathcal{S}$ computes $\vec{v}$ from $\vec{u}$
- the game's memory evolves from φ to ψ
- $\mathcal{S}$ controls randomness correctly according to $\mathcal{C}$

Omitted complexity

- Early-sampling semantics: simulator controls full randomness usage.

Bideduction summary

Bideduction predicate

$$\mathcal{C}, (\varphi, \psi) \vdash \vec{u} \rhd \vec{v}$$

There exists a (polynomial-time) simulator $\mathcal{S}$ such that when $\mathrm{Valid}(\mathcal{C})$:

- $\mathcal{S}$ computes $\vec{v}$ from $\vec{u}$
- the game's memory evolves from φ to ψ
- $\mathcal{S}$ controls randomness correctly according to $\mathcal{C}$
- $\mathcal{C}$ is well-formned

Omitted complexity

- Early-sampling semantics: simulator controls full randomness usage.
- Lifting simulator computations to probabilities
  - probabilistic couplings
  - well-formedness condition on $\mathcal{C}$

4. Automation

Proof mechanization

Proof mechanization

Objective: Automate proof search

$$\ldots \vdash \emptyset \triangleright \texttt{frame}(t)$$

Proof mechanization

Objective: Automate proof search

$$\frac{\ldots \vdash \mathtt{frame}\,\mathtt{pred}(t) \rhd \mathtt{frame}(t)}{\ldots \vdash \emptyset \rhd \mathtt{frame}(t)}\text{Induction}$$

Proof mechanization

Objective: Automate proof search

$$\dfrac{\dfrac{\dots \vdash \mathtt{frame}\,\mathtt{pred}(t) \rhd \mathtt{frame}(t)}{\dots \vdash \emptyset \rhd \mathtt{frame}(t)}\text{Induction} \qquad \dots}{\dots \vdash \mathtt{frame}\,\mathtt{pred}(t) \rhd \mathtt{frame}(A)}$$

Proof mechanization

Objective: Automate proof search

$$\frac{}{\ldots \vdash u \rhd v}$$

$$\ldots$$

$$\frac{\ldots \vdash \texttt{frame}\,\texttt{pred}(t) \rhd \texttt{frame}(A) \qquad \ldots}{\dfrac{\ldots \vdash \texttt{frame}\,\texttt{pred}(t) \rhd \texttt{frame}(t)}{\ldots \vdash \emptyset \rhd \texttt{frame}(t)}\text{Induction}}$$

Proof mechanization

Objective: Automate proof search

$$\frac{}{\ldots \vdash u \rhd v}$$

$$\ldots$$

$$\frac{\ldots \vdash \texttt{frame}\,\texttt{pred}(t) \rhd \texttt{frame}(A) \qquad \ldots}{\dfrac{\ldots \vdash \texttt{frame}\,\texttt{pred}(t) \rhd \texttt{frame}(t)}{\ldots \vdash \emptyset \rhd \texttt{frame}(t)}\text{Induction}}$$

Objectives

- Heuristic to choose proof rules to apply
- Fill the "..."
  - Propagate downward constraint and post conditions
  - Has to be guessed (induction invariants)

Proof search overview

Synthesis algorithm:

$\mathcal{C}.\square, (\varphi, \square) \vdash \vec{u} \triangleright \vec{v}, \square$

Proof search overview

Synthesis algorithm:

$$\mathcal{C}.\ ,\ (\varphi,\ )\vdash \vec{u}\triangleright\vec{v},\ \rightsquigarrow \Theta,\mathcal{C}',\psi,\vec{w}$$

Proof search overview

Synthesis algorithm:

$$\mathcal{C}, \_, (\varphi, \_) \vdash \vec{u} \triangleright \vec{v}, \_ \rightsquigarrow \Theta, \mathcal{C}', \psi, \vec{w}$$

such that

$\mathcal{C}, \mathcal{C}', (\varphi, \psi) \vdash \vec{u} \triangleright \vec{v}, \vec{w}$ holds assuming Θ holds.

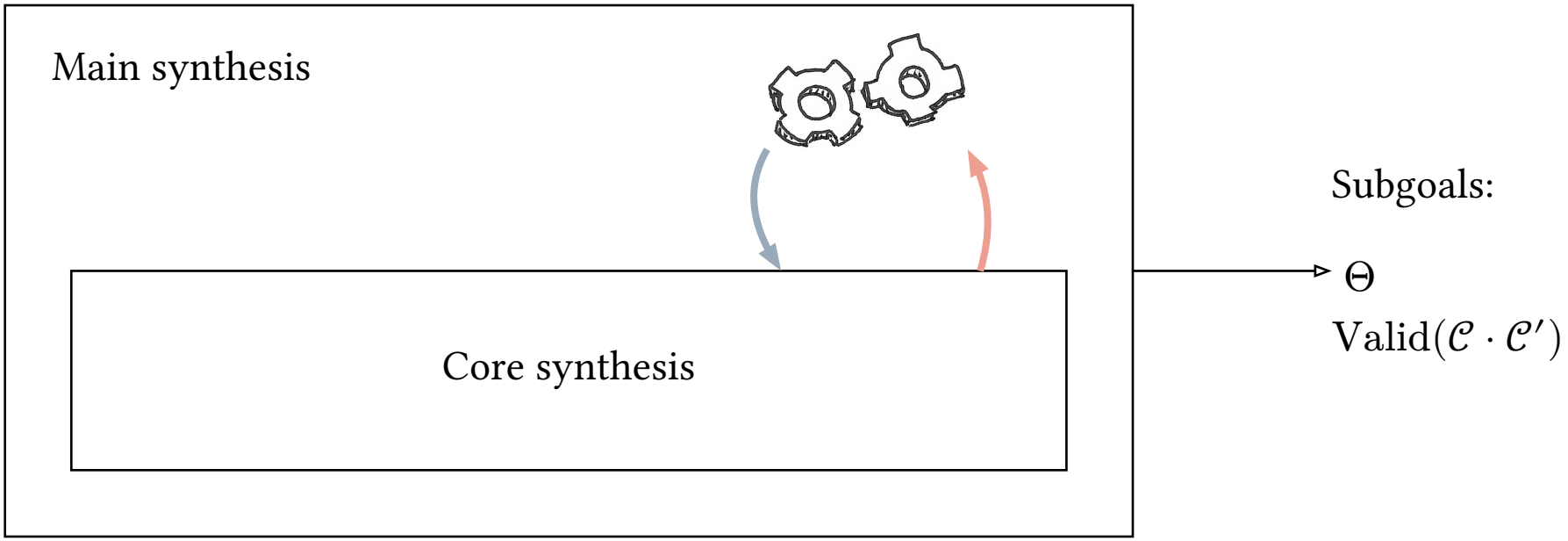
Proof search overview

Synthesis algorithm:

$\mathcal{C}, \_, (\varphi, \_) \vdash \vec{u} \triangleright \vec{v}, \_ \rightsquigarrow \Theta, \mathcal{C}', \psi, \vec{w}$

such that

$\mathcal{C}, \mathcal{C}', (\varphi, \psi) \vdash \vec{u} \triangleright \vec{v}, \vec{w}$ holds assuming Θ holds.



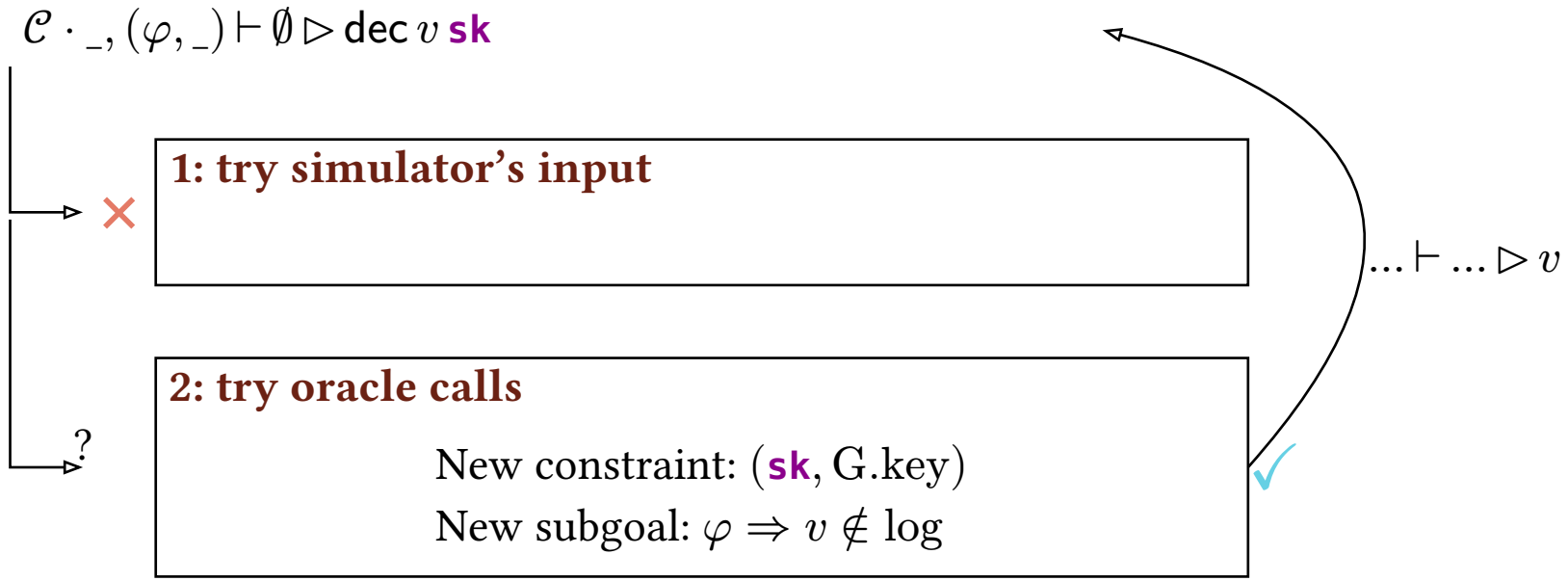
Core synthesis

$$\mathcal{C} \cdot \_, (\varphi, \_) \vdash u \rhd u$$

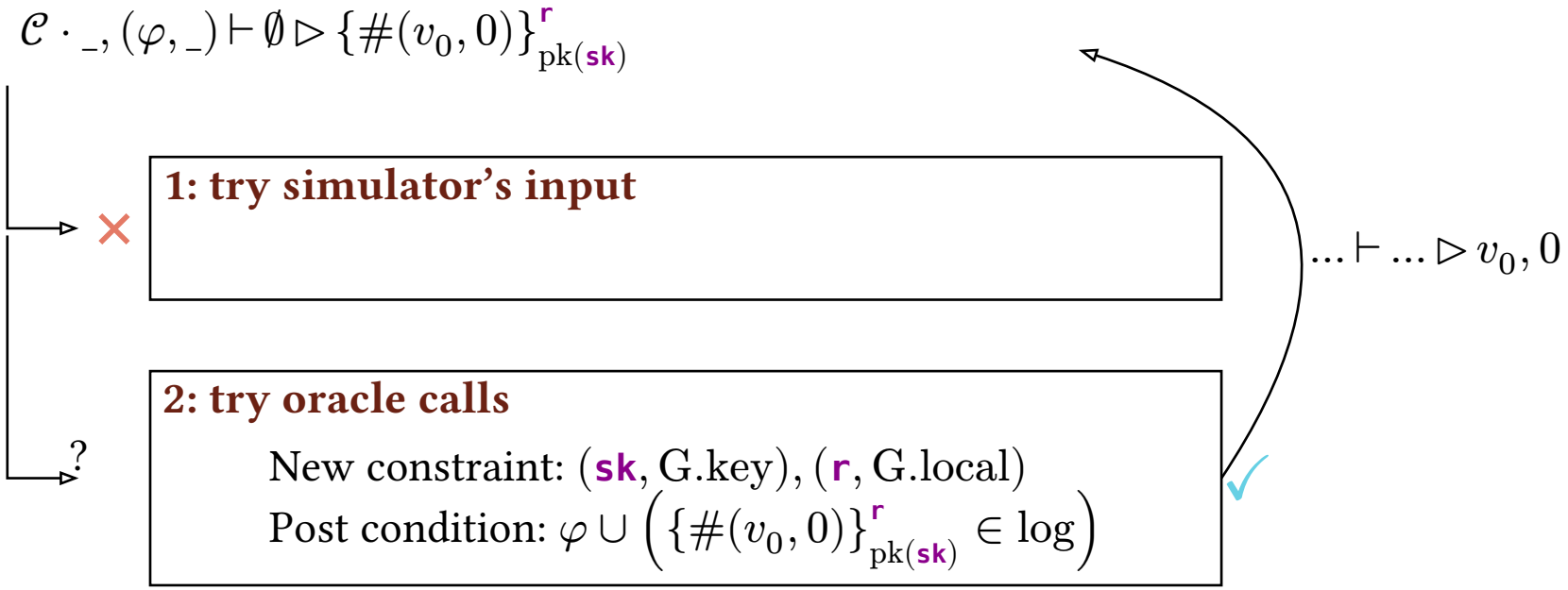
?

1: try simulator's input

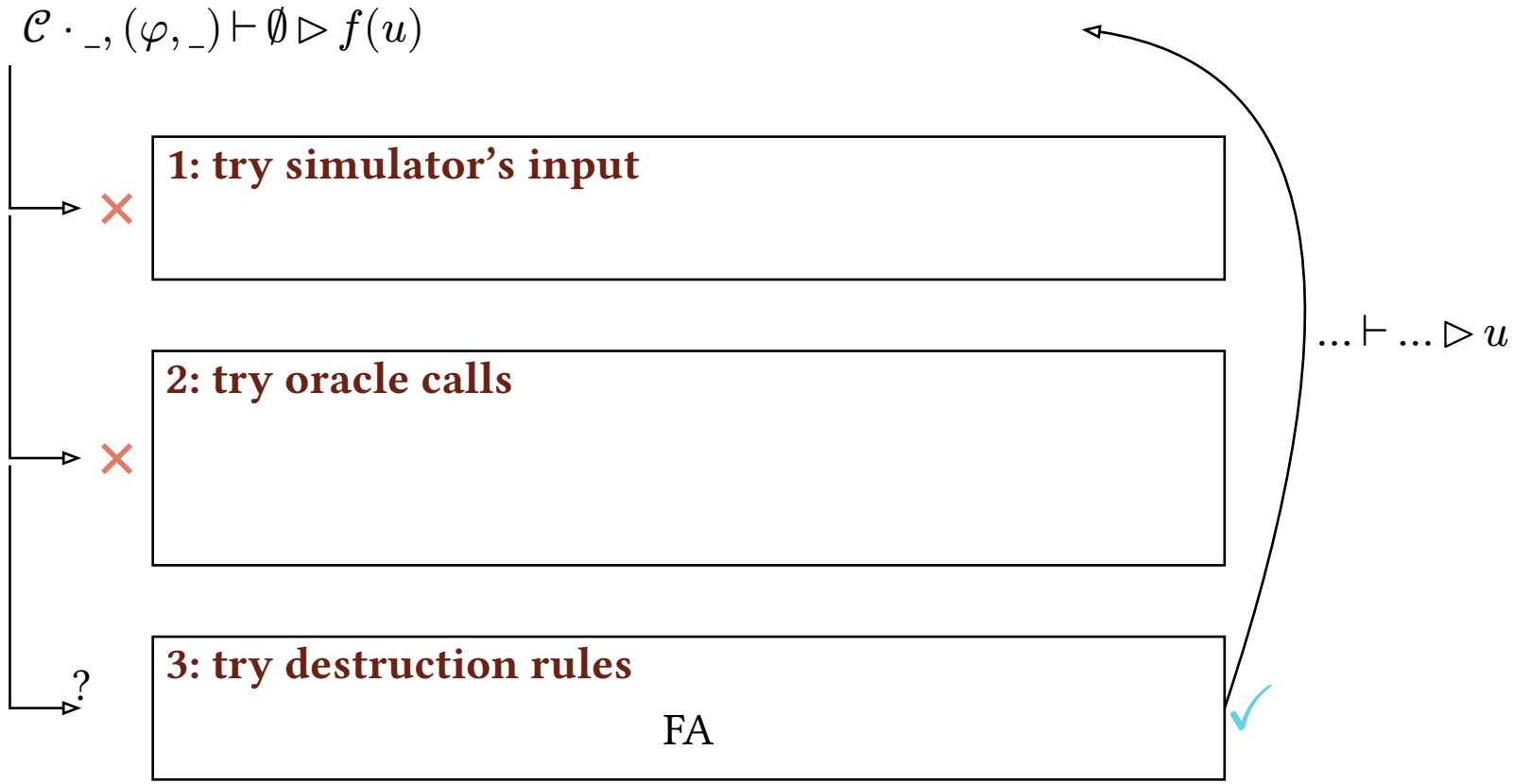
Core synthesis



Core synthesis

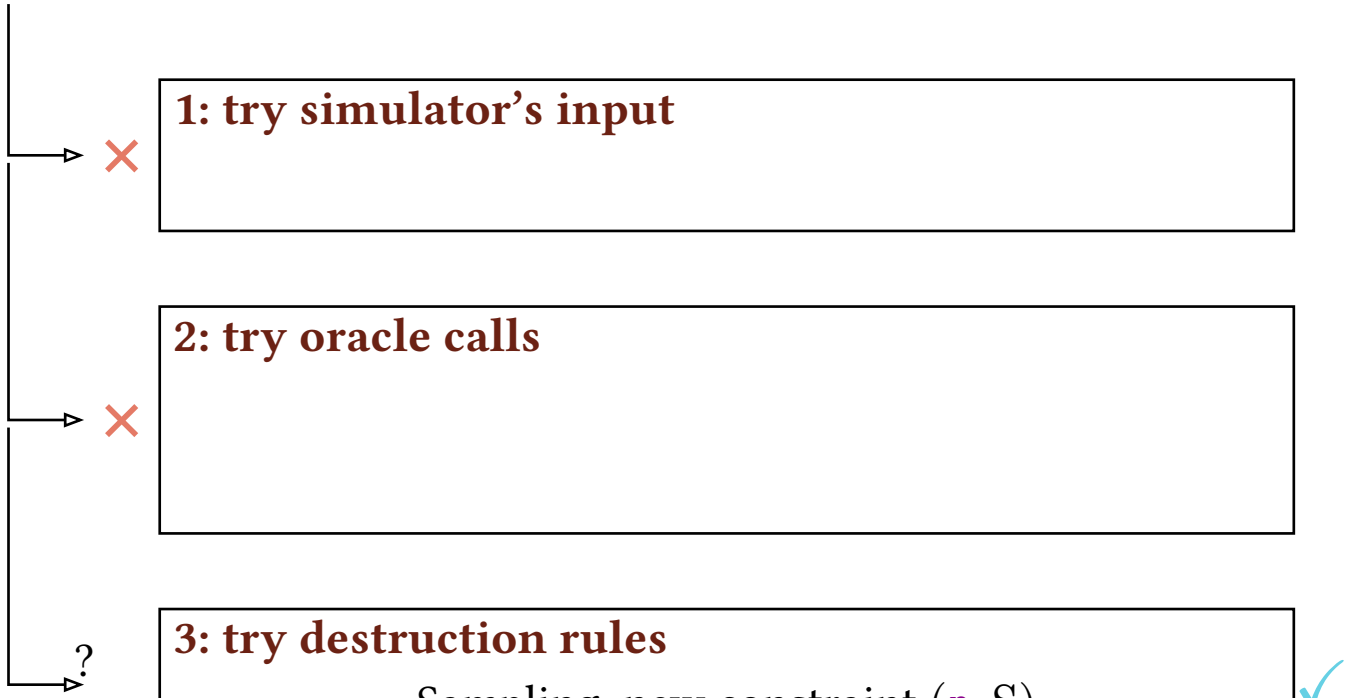


Core synthesis



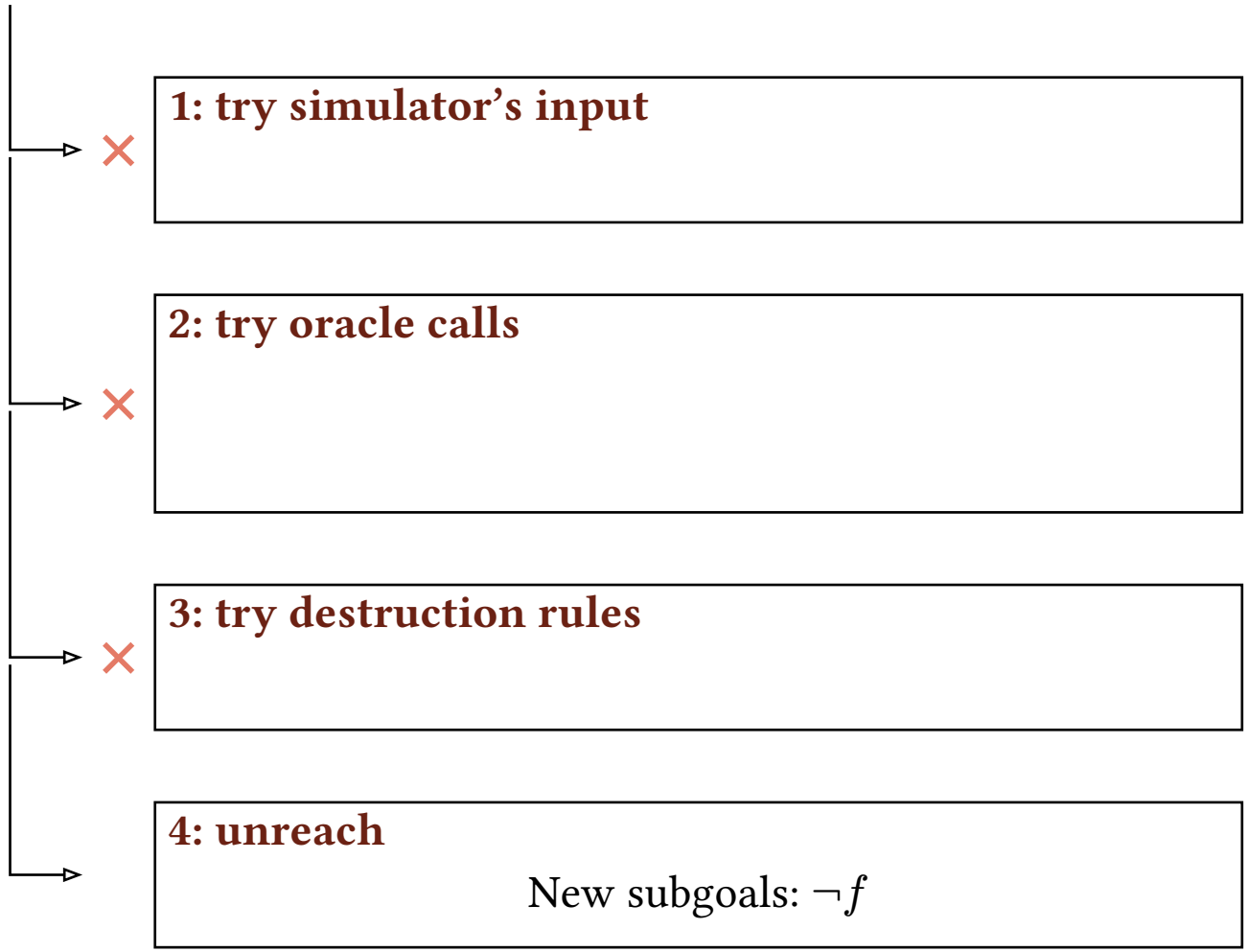
Core synthesis

$\mathcal{C} \cdot \_, (\varphi, \_) \vdash \emptyset \rhd \mathbf{n}$

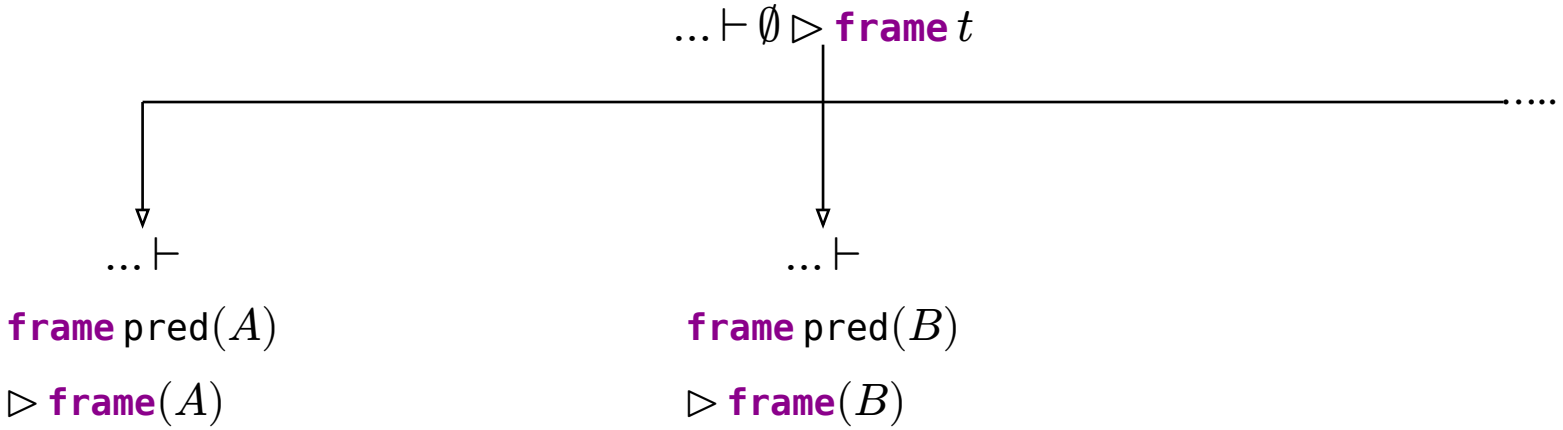


Core synthesis

$\mathcal{C} \cdot \_, (\varphi, \_) \vdash \emptyset \rhd t$ in branching f



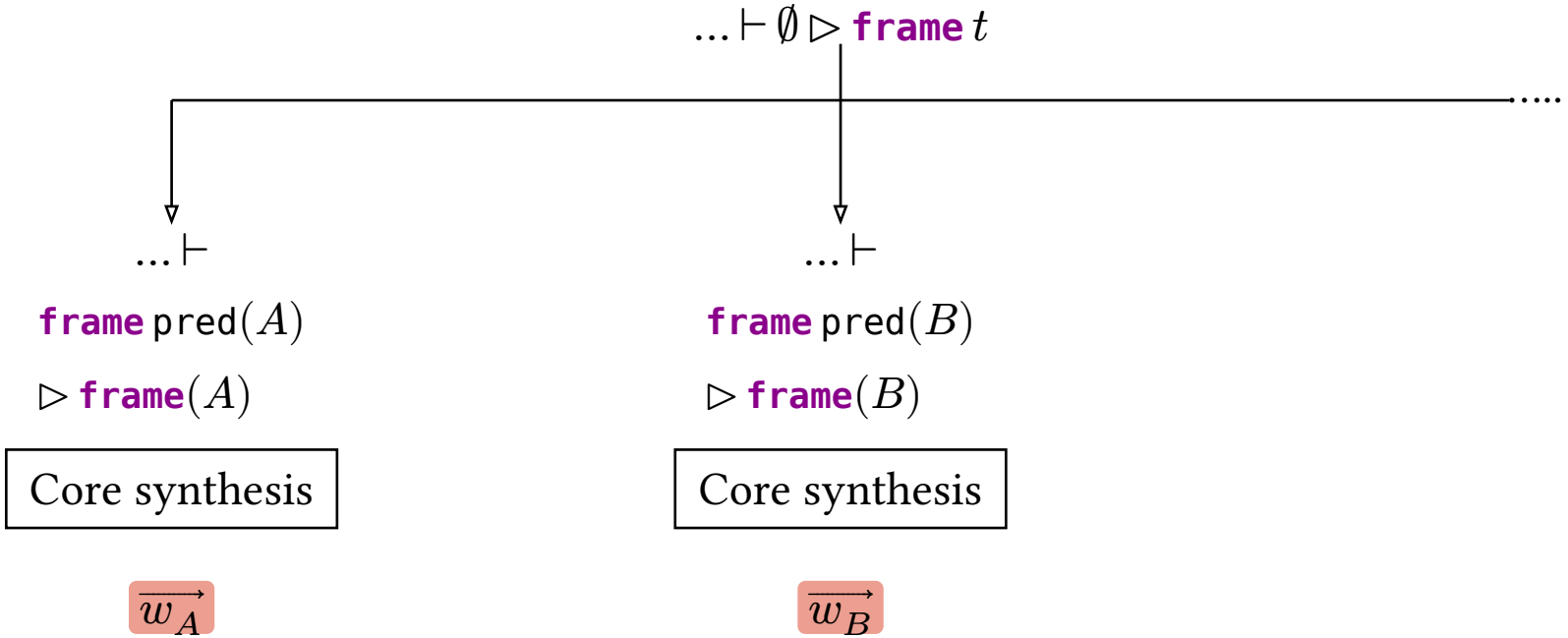
Main synthesis



Init: **Pre-process frame inductions**

- Apply the induction rule for frames
- Case analysis on time points

Main synthesis

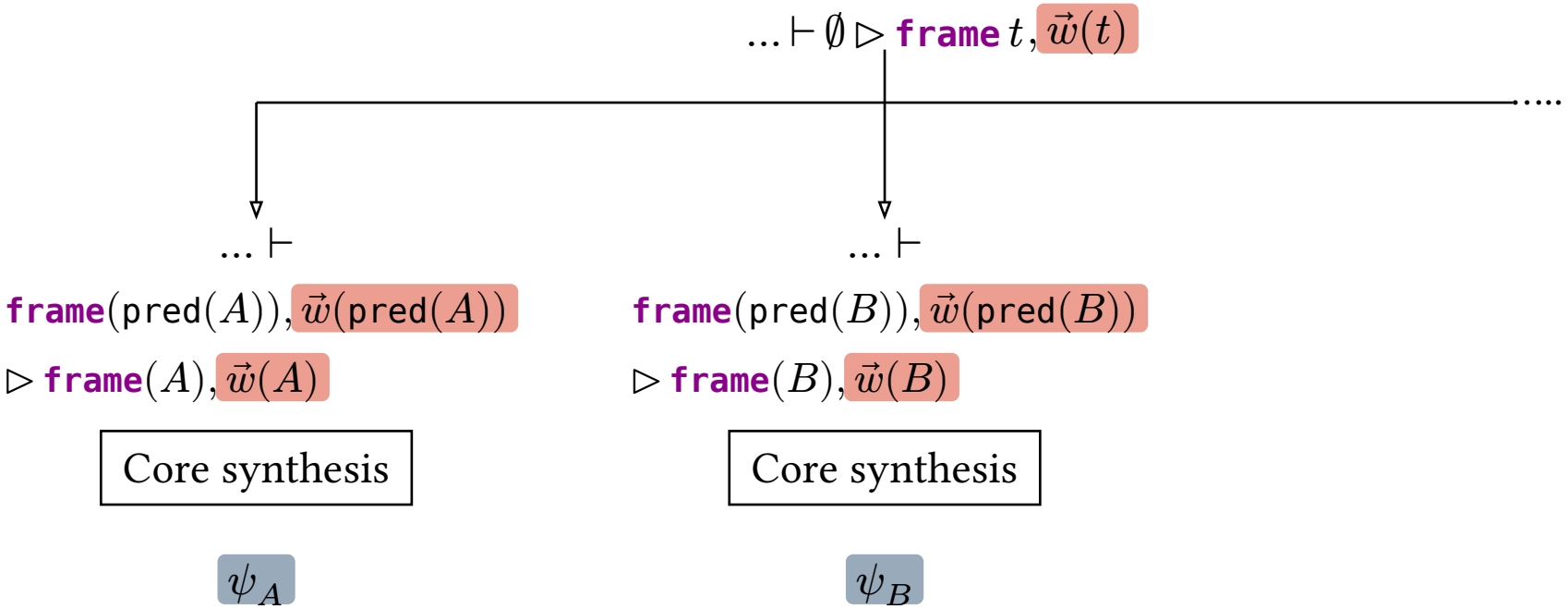


Init: **Pre-process frame inductions**

Run 1: **Memoïzed oracle calls**

- Identifies oracle calls $(\vec{w}_A, \ldots)$
- Terms shared between simulators: $\vec{w}(t) := (\texttt{if}\ A \leq t\ \texttt{then}\ \vec{w}_A, \ldots)$

Main synthesis



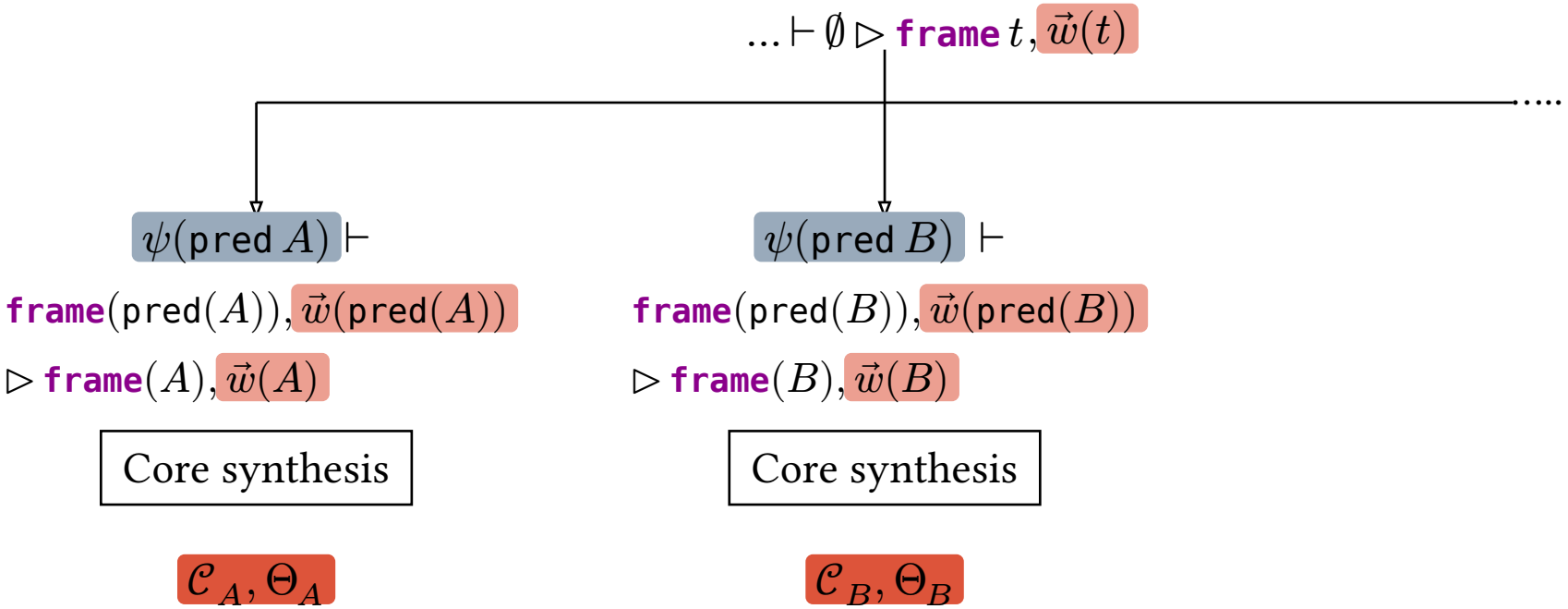
Init: **Pre-process frame inductions**

Run 1: **Memoïzed oracle calls**

Run 2: **Memory invariants**

- Memory invariants $\psi(t) := \texttt{if}\ A \leq t\ \texttt{then}\ \psi_A, \texttt{if}\ \ldots$

Main synthesis



Init: **Pre-process frame inductions**

Run 1: **Memoïzed oracle calls**

Run 2: **Memory invariants**

Run 3: **Constraints and subgoals**

5. Implementation and case studies

Squirrel implementation

What we have:

- Language for games declarations
- Synthesis implementation: `crypto` tactic.

Example:

```
game IND-CCA2 =

  rnd sk;                    oracle challenge m0 m1 = {
  var log = empty_set;         rnd r;
                               var e = enc (#(m0,m1)) r (pk sk);
  oracle pub = {               log:= add e log;
     return pk sk              return e;
  }                          }

                             oracle decrypt c = {
                               return (if not (mem c log) then dec c sk);
                             }
```

Cases studies summary

Protocol	Hypotheses	LoC	Exec time
Hash Lock	PRF	<200	<1s
Basic Hash	EUF-CAM and PRF	<200	<1s
Global CPA	CPA	<200	<1s
Private Authentication	$\text{CPA}_{\$}$	<200	<1s
Partial NSL	IND-CCA2	<200	<1s
NSL	IND-CCA2	$\sim 2.10^3$	$\sim$ 1 min 30

- Replace existing usage of legacy cryptographic tactics
- Execution time kept low in usual cases
- Support new hypotheses

Focus on FOO

Protocol	Hypotheses	LoC	Exec time
FOO	IND-CCA2, Blinding Commitment Hiding	$\sim 10^4$	~ 4 min 20

- **FOO**, largest proof in Squirrel to date

Focus on FOO

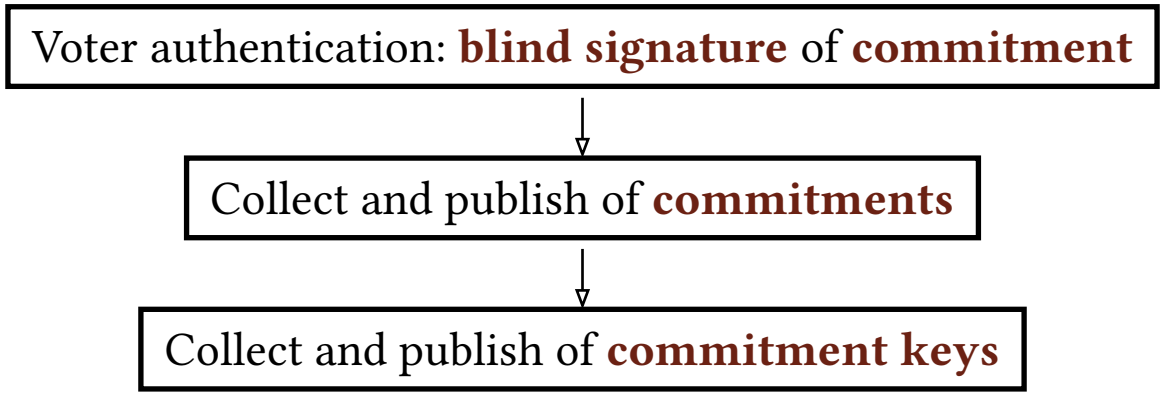
Protocol	Hypotheses	LoC	Exec time
FOO	IND-CCA2, Blinding Commitment Hiding	$\sim 10^4$	~ 4 min 20

- **FOO**, largest proof in Squirrel to date
- Adapted from existing *pen and paper* proof on three voters
- First computational proof for an **unbounded** number of voters

Focus on FOO

Protocol	Hypotheses	LoC	Exec time
FOO	IND-CCA2, Blinding Commitment Hiding	$\sim 10^4$	$\sim$ 4 min 20

- **FOO**, largest proof in Squirrel to date
- Adapted from existing *pen and paper* proof on three voters
- First computational proof for an **unbounded** number of voters

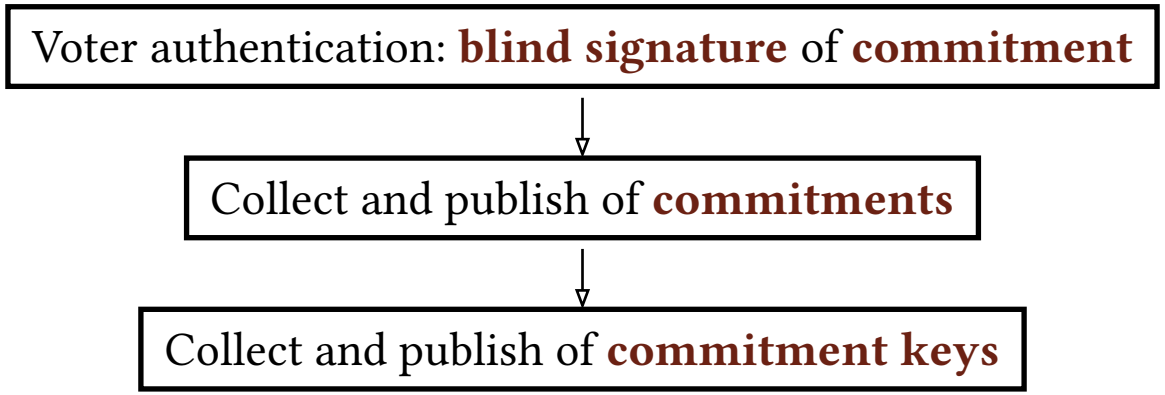


Proof structure:
1. IND-CCA2
2. Shuffle + **deductions** to reduce the protocol
3. Cryptographic reduction on **commitments** and **blind signatures**

Focus on FOO

Protocol	Hypotheses	LoC	Exec time
FOO	IND-CCA2, Blinding Commitment Hiding	$\sim 10^4$	$\sim$ 4 min 20

- **FOO**, largest proof in Squirrel to date
- Adapted from existing *pen and paper* proof on three voters
- First computational proof for an **unbounded** number of voters



Proof structure:
1. IND-CCA2
2. Shuffle + **deductions** to reduce the protocol
3. Cryptographic reduction on **commitments** and **blind signatures**

Execution time: realistic use of the tactics:

- Most calls are $< 0.5s$
- Maximum: 11s

6. Conclusion

What has been done

- Framework for simulator synthesis
  - Bideduction predicate and proof system
  - Structure to support composability:
    - constraint system
    - pre and post conditions

What has been done

- Framework for simulator synthesis
  - Bideduction predicate and proof system
  - Structure to support composability:
    - constraint system
    - pre and post conditions
- Automation
  - Goal- and constraints-directed
  - Infer memoïzing simulators

What has been done

- Framework for simulator synthesis
  - Bideduction predicate and proof system
  - Structure to support composability:
    - constraint system
    - pre and post conditions
- Automation
  - Goal- and constraints-directed
  - Infer memoïzing simulators
- Implementation
  - Integrated in Squirrel's stream
  - Application to simple protocols
    - Replace existing axioms, and support new assumptions
  - Large-scale validation with proof of FOO

Limitations and future work

Limitations and future work

Limitations and immediate follow-up

- More permissive semantics (approximate couplings)
- Enrich pre and post conditions logics (maps)

Limitations and future work

Limitations and immediate follow-up

- More permissive semantics (approximate couplings)
- Enrich pre and post conditions logics (maps)

General new directions

- Inferring game hops
- Debate on full automation: strong constraints on running time, limiting improvements
- Transfer to other frameworks (ex: EasyCrypt)