COA-Secure Obfuscation and Applications

Ran Canetti
Boston University

Suvradip Chakraborty
ETH Zurich

Manoj Prabhakaran
IIT Bombay

Dakshita Khurana
UIUC

Nishant Kumar*
UIUC

Oxana Poburinnaya
Boston University

EUROCRYPT 2022, Trondheim, Norway, May 31, 2022

International Association for Cryptologic Research

This paper in a nutshell

We provide a **framework** for endowing software obfuscation with "**verifiability**" and "**non-malleability**" guarantees and show **generic constructions satisfying the above guarantees.**

Roadmap

- Motivation
- New Definitions- COA Obfuscation
- New Applications
  - Complete CCA Encryption
  - Stronger (keyless) software watermarking

Roadmap

- Motivation
- New Definitions- COA Obfuscation
- New Applications
  - ☑ Complete CCA Encryption
  - Stronger (keyless) software watermarking

Roadmap

- Motivation
- New Definitions- COA Obfuscation
- New Applications
  - ☑ Complete CCA Encryption
  - Stronger (keyless) software watermarking
- Construction of COA Obfuscation

Program Obfuscation — Boon or Bane for Software Users?

Program Obfuscation— Boon or Bane for Software Users?

- General purpose program obfuscation allows distribution and execution of software without fear of revealing sensitive design secrets/ keys hidden in code.

Program Obfuscation — Boon or Bane for Software Users?

- General purpose program obfuscation allows distribution and execution of software without fear of revealing sensitive design secrets/ keys hidden in code.

Issues:

Program Obfuscation— Boon or Bane for Software Users?

- General purpose program obfuscation allows distribution and execution of software without fear of revealing sensitive design secrets/ keys hidden in code.

Issues:

- **Verifiability**: Verification of properties (structure/functionality) of the underlying program much harder.

Program Obfuscation — Boon or Bane for Software Users?

- General purpose program obfuscation allows distribution and execution of software without fear of revealing sensitive design secrets/ keys hidden in code.

Issues:

- **Verifiability**: Verification of properties (structure/functionality) of the underlying program much harder.
  - Reduced to only *black-box* testing the program.

Program Obfuscation — Boon or Bane for Software Users?

- General purpose program obfuscation allows distribution and execution of software without fear of revealing sensitive design secrets/ keys hidden in code.

Issues:

- **Verifiability**: Verification of properties (structure/functionality) of the underlying program much harder.
    - Reduced to only *black-box* testing the program.
    - The obfuscated program comes from an untrusted source.

Program Obfuscation— Boon or Bane for Software Users?

- General purpose program obfuscation allows distribution and execution of software without fear of revealing sensitive design secrets/ keys hidden in code.

Issues:

- **Verifiability**: Verification of properties (structure/functionality) of the underlying program much harder.
  - Reduced to only *black-box* testing the program.
  - The obfuscated program comes from an untrusted source.
- **Malleability**: A program might depend on other (related) program(s) in "illegitimate" ways.

Program Obfuscation — Boon or Bane for Software Users?

- General purpose program obfuscation allows distribution and execution of software without fear of revealing sensitive design secrets/ keys hidden in code.

Issues:

- **Verifiability**: Verification of properties (structure/functionality) of the underlying program much harder.
  - Reduced to only *black-box* testing the program.
  - The obfuscated program comes from an untrusted source.
- **Malleability**: A program might depend on other (related) program(s) in "illegitimate" ways.
  - Obfuscation might facilitate *software plagiarism* — might hide that program A uses a potentially mauled version of a (proprietary) program B as sub-routine and A's behaviour may depend on B.

Program Obfuscation — Boon or Bane for Software Users?

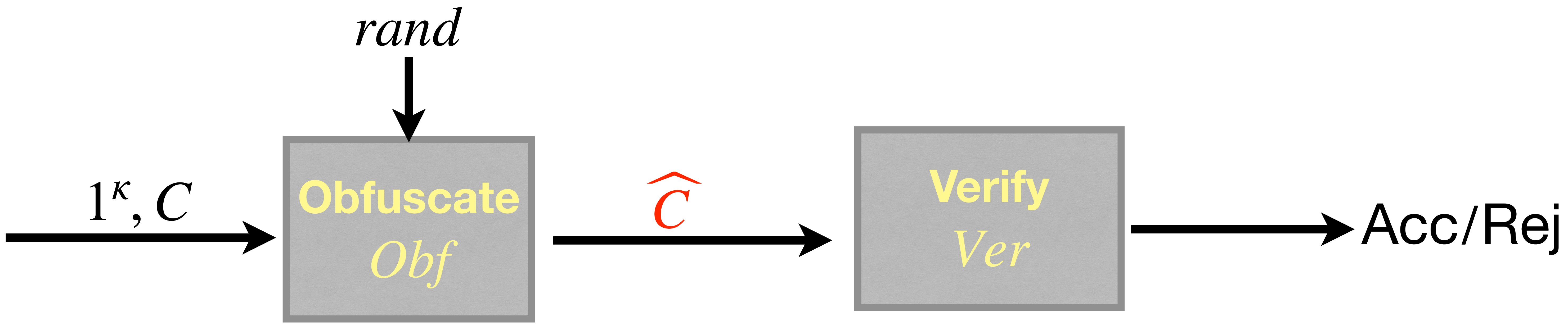
- General purpose program obfuscation allows distribution and execution of software without fear of revealing sensitive design secrets/ keys hidden in code.

Issues:

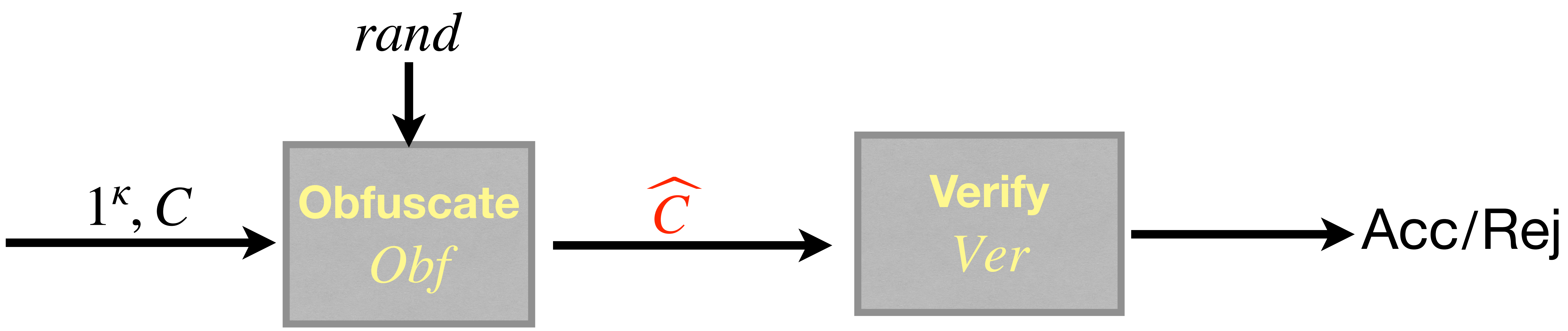
- **Verifiability**: Verification of properties (structure/functionality) of the underlying program much harder.
  - Reduced to only *black-box* testing the program.
  - The obfuscated program comes from an untrusted source.
- **Malleability**: A program might depend on other (related) program(s) in "illegitimate" ways.
  - Obfuscation might facilitate *software plagiarism* — might hide that program A uses a potentially mauled version of a (proprietary) program B as sub-routine and A's behaviour may depend on B.
  - $i\mathcal{O}$ obfuscation of a (puncturable) PRF K might allow an adversary to create an obfuscation of the same function but with key $K+1$

Defining Verifiable and non-malleable obfuscation: Challenges

Defining Verifiable and non-malleable obfuscation: Challenges

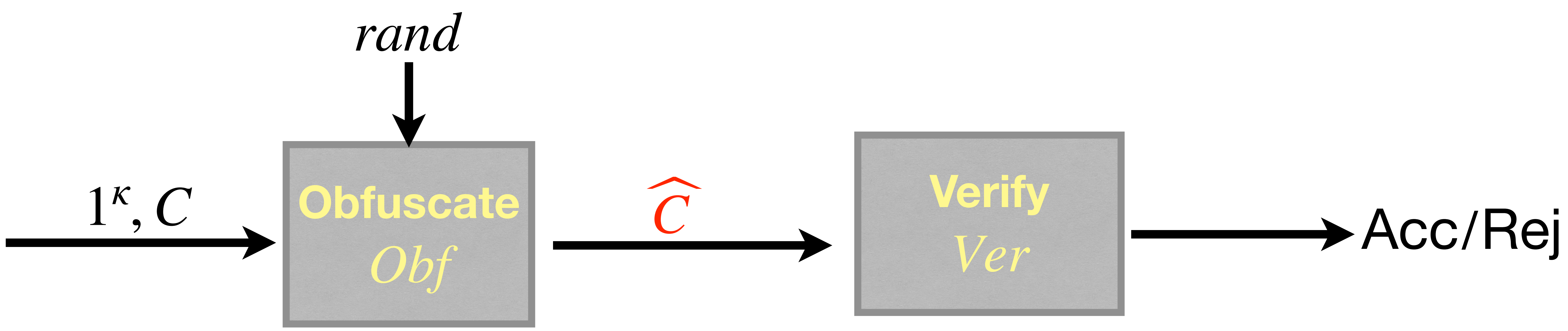


Defining Verifiable and non-malleable obfuscation: Challenges



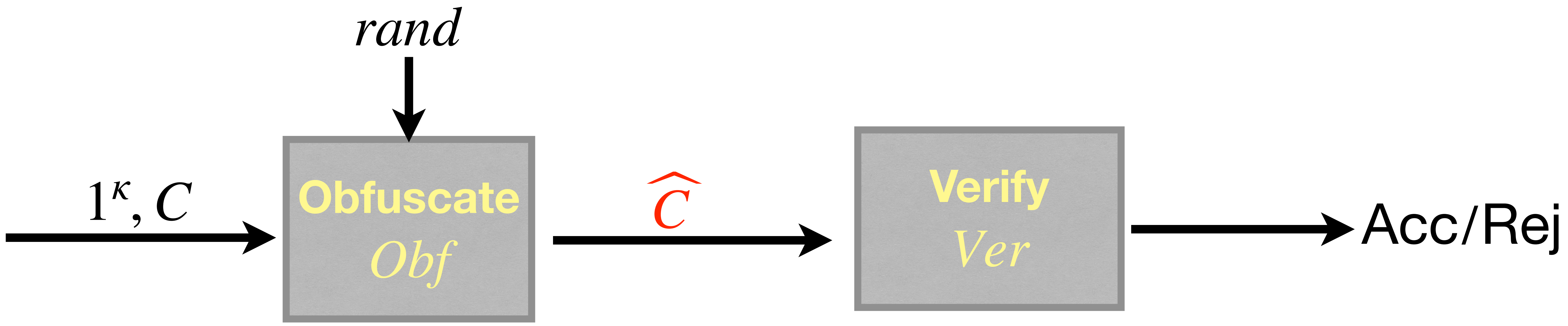
- **Attempt 1**: Whenever Ver accepts, a specific circuit C is being obfuscated

Defining Verifiable and non-malleable obfuscation: Challenges



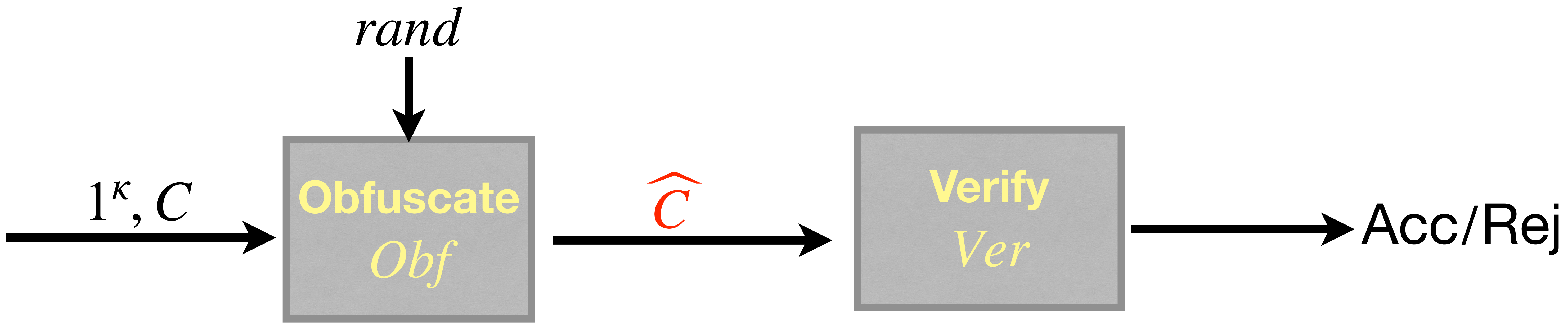
- **Attempt 1**: Whenever *Ver* accepts, a specific circuit C is being obfuscated
- **Attempt 2**: Whenever *Ver* accepts, there exists some circuit C s.t. $\widehat{C} \equiv C$.

Defining Verifiable and non-malleable obfuscation: Challenges



- **Attempt 1**: Whenever *Ver* accepts, a specific circuit C is being obfuscated
- **Attempt 2**: Whenever *Ver* accepts, there exists some circuit C s.t. $\widehat{C} \equiv C$.
- Consider a public predicate ϕ, s.t. if $Ver(\widehat{C}, \phi) = \text{Acc} \implies \exists C$ s.t. $\widehat{C} \equiv C$ and $\phi(C) = 1$.

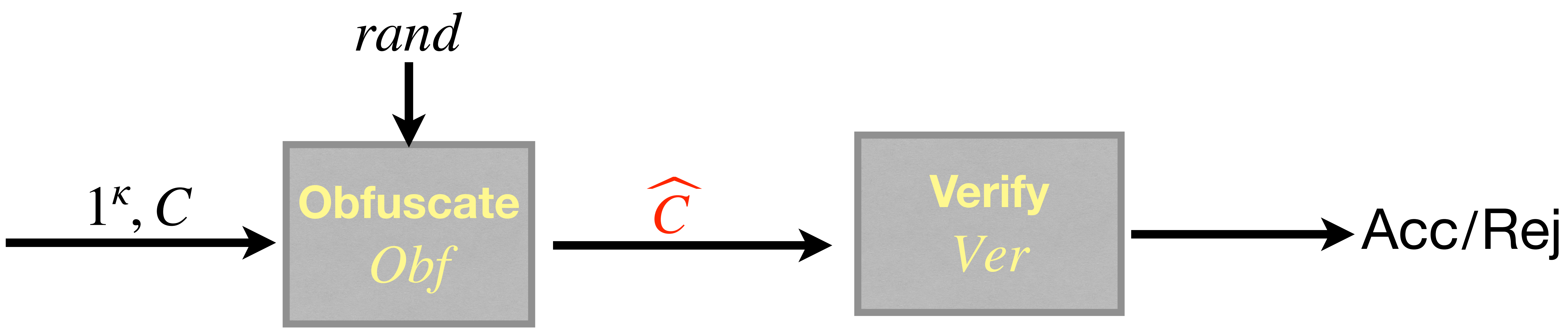
Defining Verifiable and non-malleable obfuscation: Challenges



- **Attempt 1**: Whenever Ver accepts, a specific circuit C is being obfuscated
- **Attempt 2**: Whenever Ver accepts, there exists some circuit C s.t. $\widehat{C} \equiv C$.
- Consider a public predicate ϕ, s.t. if $Ver(\widehat{C}, \phi) = \text{Acc} \implies \exists C$ s.t. $\widehat{C} \equiv C$ and $\phi(C) = 1$.

- Trivial solution: Use NIZK proofs to attest to the structure and functionality of the underlying plaintext program.

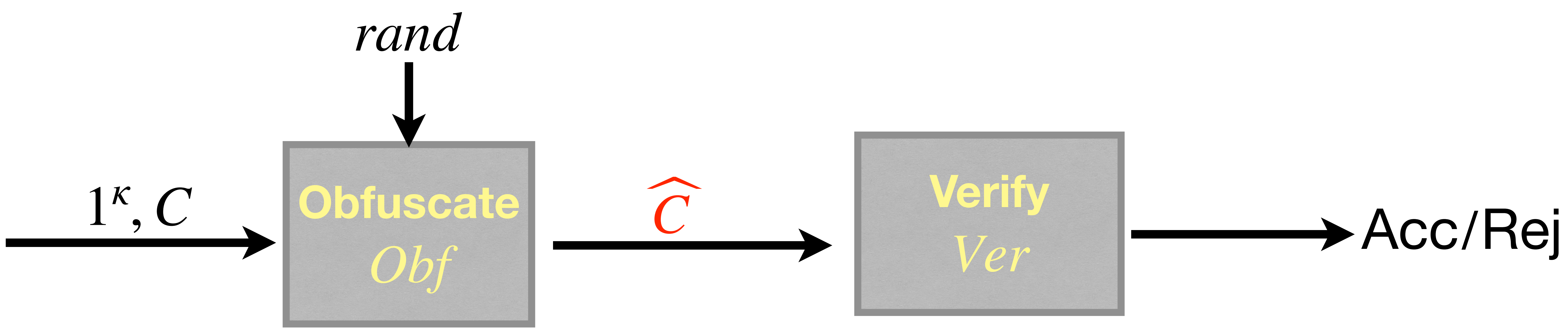
Defining Verifiable and non-malleable obfuscation: Challenges



- **Attempt 1**: Whenever *Ver* accepts, a specific circuit C is being obfuscated
- **Attempt 2**: Whenever *Ver* accepts, there exists some circuit C s.t. $\widehat{C} \equiv C$.
- Consider a public predicate ϕ, s.t. if $Ver(\widehat{C}, \phi) = \text{Acc} \implies \exists C$ s.t. $\widehat{C} \equiv C$ and $\phi(C) = 1$.

- Trivial solution: Use NIZK proofs to attest to the structure and functionality of the underlying plaintext program.
    - Requires trusted setup assumptions (Common random/reference string (CRS))

Defining Verifiable and non-malleable obfuscation: Challenges



- **Attempt 1**: Whenever Ver accepts, a specific circuit C is being obfuscated
- **Attempt 2**: Whenever Ver accepts, there exists some circuit C s.t. $\widehat{C} \equiv C$.
- Consider a public predicate ϕ, s.t. if $Ver(\widehat{C}, \phi) = \text{Acc} \implies \exists C$ s.t. $\widehat{C} \equiv C$ and $\phi(C) = 1$.

- Trivial solution: Use NIZK proofs to attest to the structure and functionality of the underlying plaintext program.
  - Requires trusted setup assumptions (Common random/reference string (CRS))
- **Goal**: Construct verifiable and non-malleable obfuscation in the "plain" model

Prior works

Prior works

Verifiability compiler for Functional Encryption and Indistinguishability Obfuscation

- Badrinarayanan, Goyal, Jain, Sahai

Prior works

Verifiability compiler for Functional Encryption and Indistinguishability Obfuscation

- Badrinarayanan, Goyal, Jain, Sahai
  Verifiable Functional Encryption, Asiacrypt'16

Prior works

Verifiability compiler for Functional Encryption and Indistinguishability Obfuscation

- Badrinarayanan, Goyal, Jain, Sahai
  Verifiable Functional Encryption, Asiacrypt'16
  - Does not require any trusted setup assumptions.

Prior works

Verifiability compiler for Functional Encryption and Indistinguishability Obfuscation

- Badrinarayanan, Goyal, Jain, Sahai
  Verifiable Functional Encryption, Asiacrypt'16
  - Does not require any trusted setup assumptions.
  - Techniques tailer-made for $i\mathcal{O}$.

Prior works

Verifiability compiler for Functional Encryption and Indistinguishability Obfuscation

- Badrinarayanan, Goyal, Jain, Sahai
  Verifiable Functional Encryption, Asiacrypt'16

  - Does not require any trusted setup assumptions.
  - Techniques tailer-made for $i\mathcal{O}$.
  - *Limited* form of Hiding: Requires a "short proof/witness" of equivalence for the functionally equivalent circuits.

Prior works

Verifiability compiler for Functional Encryption and Indistinguishability Obfuscation

- Badrinarayanan, Goyal, Jain, Sahai
  Verifiable Functional Encryption, Asiacrypt'16
  - Does not require any trusted setup assumptions.
  - Techniques tailer-made for $i\mathcal{O}$.
  - *Limited* form of Hiding: Requires a "short proof/witness" of equivalence for the functionally equivalent circuits.
  - No guarantees against general mauling attacks

Prior works

Verifiability compiler for Functional Encryption and Indistinguishability Obfuscation

- Badrinarayanan, Goyal, Jain, Sahai
  Verifiable Functional Encryption, Asiacrypt'16

  - Does not require any trusted setup assumptions.
  - Techniques tailer-made for $i\mathcal{O}$.
  - *Limited* form of Hiding: Requires a "short proof/witness" of equivalence for the functionally equivalent circuits.
  - No guarantees against general mauling attacks

Non-malleability for VBB obfuscation

- Canetti, Varia
  Non-malleable Obfuscation, TCC'09

Prior works

Verifiability compiler for Functional Encryption and Indistinguishability Obfuscation

- Badrinarayanan, Goyal, Jain, Sahai
  Verifiable Functional Encryption, Asiacrypt'16

  - Does not require any trusted setup assumptions.
  - Techniques tailer-made for $i\mathcal{O}$.
  - *Limited* form of Hiding: Requires a "short proof/witness" of equivalence for the functionally equivalent circuits.
  - No guarantees against general mauling attacks

Non-malleability for VBB obfuscation

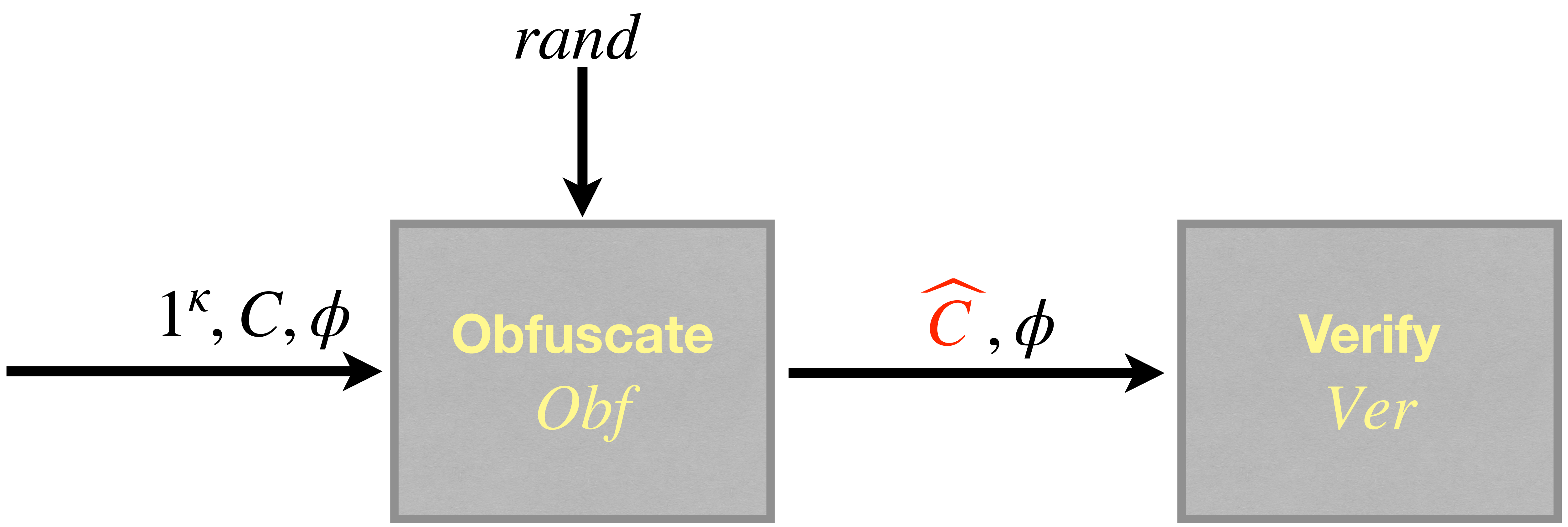
- Canetti, Varia
  Non-malleable Obfuscation, TCC'09

  - Considers obfuscation of point functions and related functionalities

Our notion: Security against Chosen Obfuscation Attacks (COA Obfuscation): Defn. 1

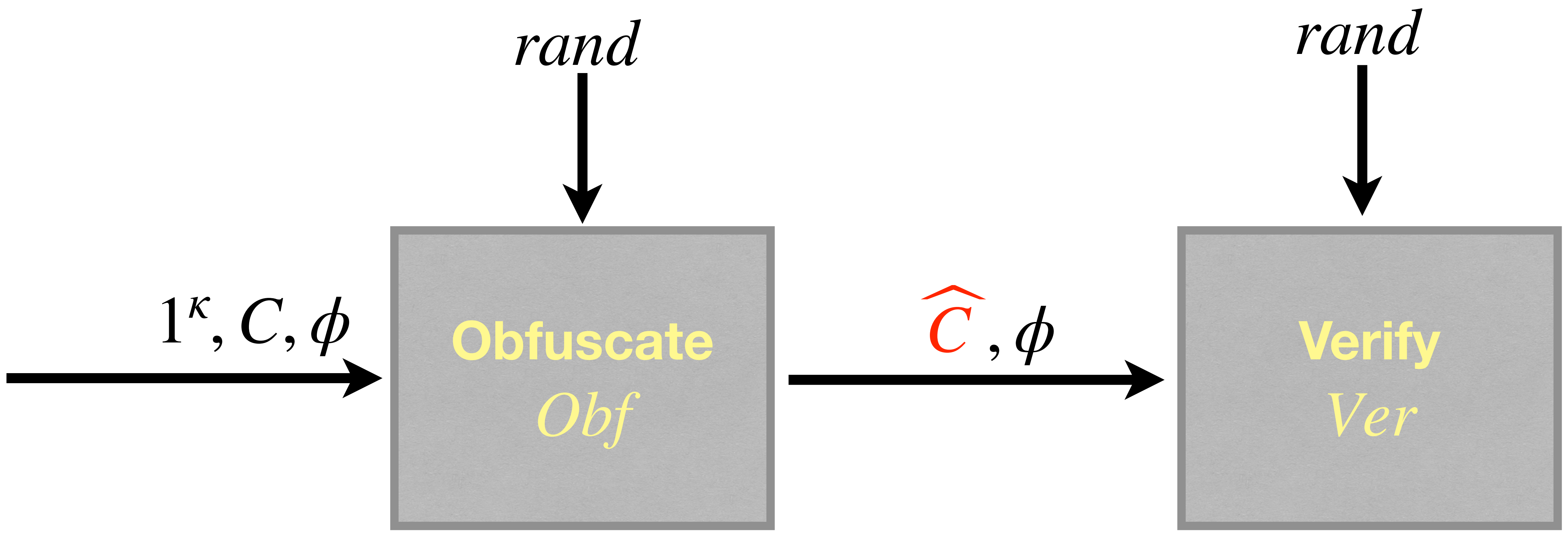
Our notion: Security against Chosen Obfuscation Attacks (COA Obfuscation): Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $C \in \mathscr{C}$.



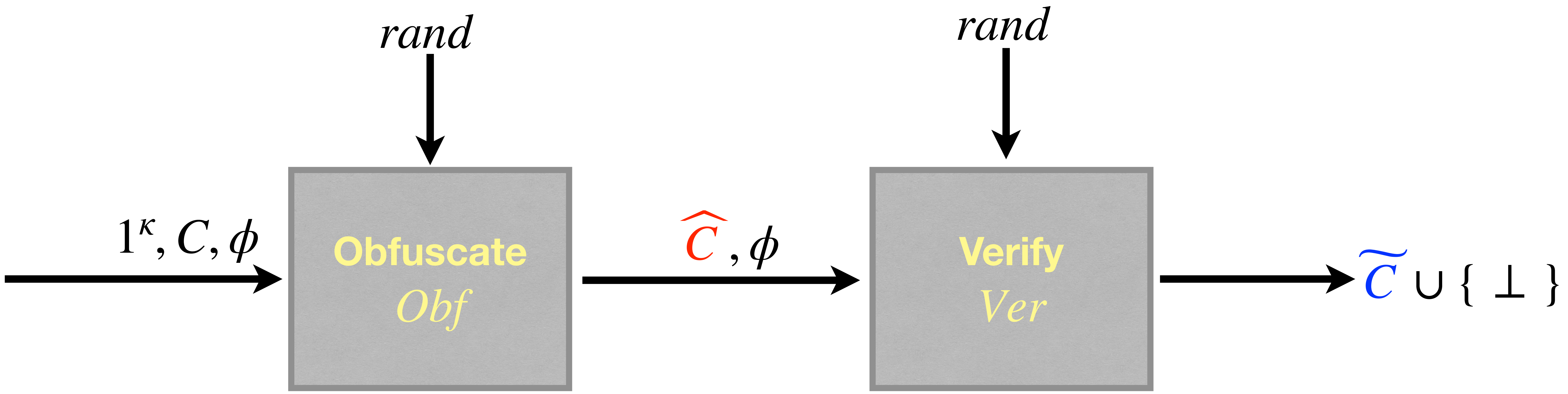
Our notion: Security against Chosen Obfuscation Attacks (COA Obfuscation): Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $C \in \mathscr{C}$.



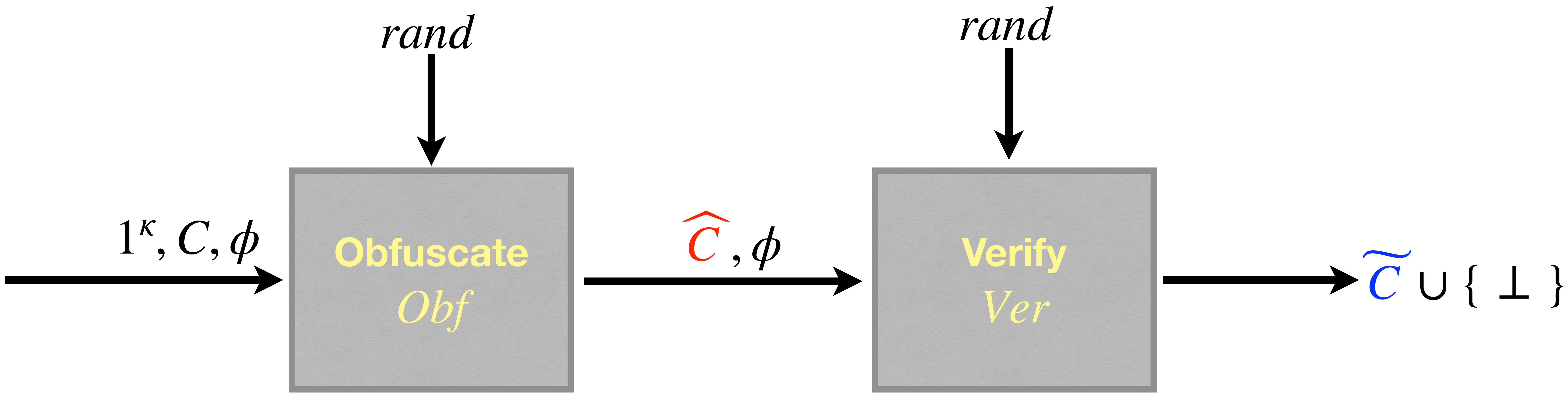
Our notion: Security against Chosen Obfuscation Attacks (COA Obfuscation): Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $C \in \mathscr{C}$.



Our notion: Security against Chosen Obfuscation Attacks (COA Obfuscation): Defn. 1

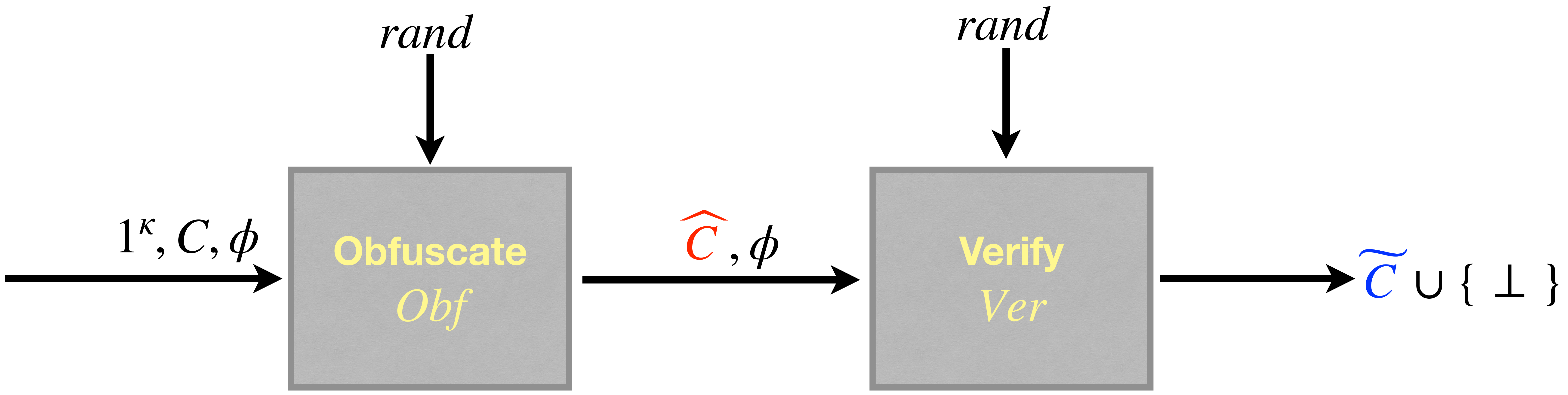
Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $C \in \mathscr{C}$.



☑ **Correctness**: For a legitimate C, i.e., $\phi(C) = 1$, $\widetilde{C}$ is functionally equivalent to C.

Our notion: Security against Chosen Obfuscation Attacks (COA Obfuscation): Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $C \in \mathscr{C}$.

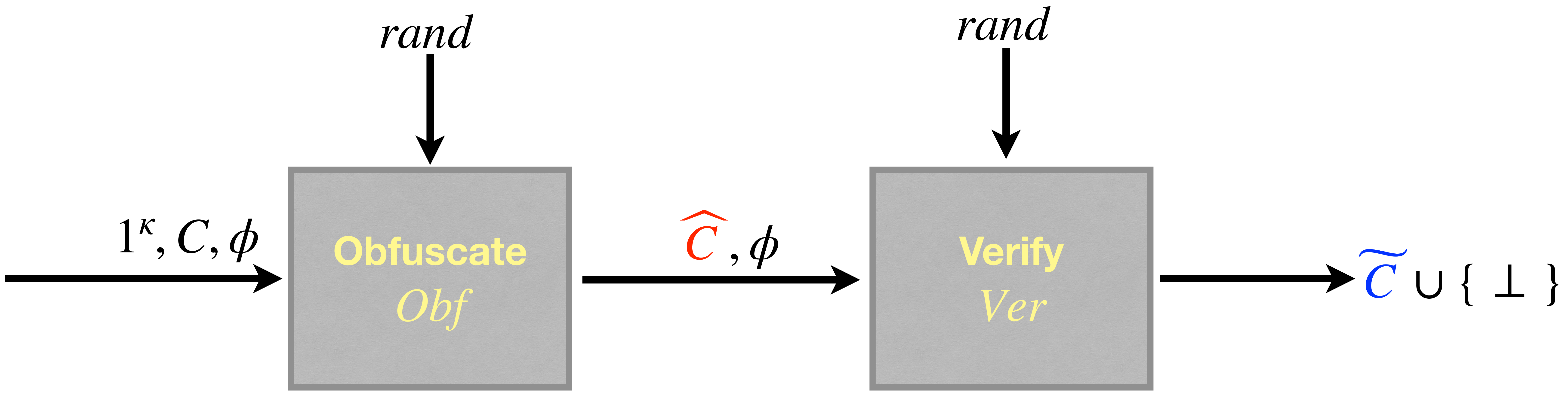


☑ **Correctness**: For a legitimate C, i.e., $\phi(C) = 1$, $\widetilde{C}$ is functionally equivalent to C.

☑ **Soundness**: For any string $\widehat{C}$, if Verify outputs some $\widetilde{C}$, i.e., $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$ s.t $\widetilde{C} \neq \perp$ => there exists an underlying circuit $C \in \mathscr{C}$ such that $\widetilde{C} \equiv C$ and $\phi(C) = 1$.

Our notion: Security against Chosen Obfuscation Attacks (COA Obfuscation): Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $C \in \mathscr{C}$.



☑ **Correctness**: For a legitimate C, i.e., $\phi(C) = 1$, $\widetilde{C}$ is functionally equivalent to C.

☑ **Soundness**: For any string $\widehat{C}$, if Verify outputs some $\widetilde{C}$, i.e., $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$ s.t $\widetilde{C} \neq \perp$ **=>** there exists an underlying circuit $C \in \mathscr{C}$ such that $\widetilde{C} \equiv C$ and $\phi(C) = 1$.

☑ **COA security**: For "sufficiently similar" circuits C_0 and C_1, $Obf(C_0, \phi) \approx_c Obf(C_1, \phi)$, even given access to a *"de-obfuscation oracle"* $\mathbb{O}(\cdot, \phi)$.

Defining COA Security: Defn. 1

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Adversary

Challenger

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Adversary

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$

$\xrightarrow{(C_0, C_1, z)}$

Challenger

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Adversary

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$

Samp must be "admissible ϕ-satisfying "sampler:

$C_0, C_1 \in \mathscr{C}$ and $\phi(C_0) = \phi(C_1) = 1$

2. $C_0 \approx_c C_1$ for any black-box PPT distinguisher

$$(C_0, C_1, z) \longrightarrow$$

Challenger

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Adversary

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$

Samp must be "admissible ϕ-satisfying "sampler:

$C_0, C_1 \in \mathscr{C}$ and $\phi(C_0) = \phi(C_1) = 1$

2. $C_0 \approx_c C_1$ for any black-box PPT distinguisher

(C_0, C_1, z) $\longrightarrow$

$\longleftarrow$ $\widehat{C}$

Challenger

$b \xleftarrow{\$} \{0,1\}$

$\widehat{C} \leftarrow Obf(C_b, \phi)$

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Adversary

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$

Challenger

$b \xleftarrow{\$} \{0,1\}$

(C_0, C_1, z) →

$\color{blue}{\widehat{C}}$ ←

$\color{blue}{\widehat{C}} \leftarrow Obf(C_b, \phi)$

$\widehat{C}$ →

Samp must be “admissible ϕ-satisfying “sampler:

$C_0, C_1 \in \mathscr{C}$ and $\phi(C_0) = \phi(C_1) = 1$

2. $C_0 \approx_c C_1$ for any black-box PPT distinguisher

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Adversary

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$

Samp must be "admissible ϕ-satisfying "sampler:

$C_0, C_1 \in \mathscr{C}$ and $\phi(C_0) = \phi(C_1) = 1$

2. $C_0 \approx_c C_1$ for any black-box PPT distinguisher

(C_0, C_1, z) $\longrightarrow$

$\longleftarrow$ $\widehat{C}$

$\widehat{C}$ $\longrightarrow$

Challenger

$b \xleftarrow{\$} \{0,1\}$

$\widehat{C} \leftarrow Obf(C_b, \phi)$

1. If $\widehat{C} = \widehat{C}$, output $\perp$.
2. Run $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$
3. If $\widetilde{C} \neq \perp$, return C such that $\widetilde{C} = \mathcal{O}(C; r)$ for some r

The De-obfuscation oracle $\mathbb{O}(\cdot, \phi)$

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Adversary

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$

Samp must be "admissible ϕ-satisfying "sampler:

$C_0, C_1 \in \mathscr{C}$ and $\phi(C_0) = \phi(C_1) = 1$

2. $C_0 \approx_c C_1$ for any black-box PPT distinguisher

Challenger

$b \xleftarrow{\$} \{0,1\}$

$\xrightarrow{(C_0, C_1, z)}$

$\widehat{C} \leftarrow Obf(C_b, \phi)$

$\xleftarrow{\widehat{C}}$

$\xrightarrow{\widehat{C}}$

1. If $\widehat{C} = \widehat{C}$, output $\perp$.

2. Run $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$

3. If $\widetilde{C} \neq \perp$, return C such that $\widetilde{C} = \mathscr{O}(C; r)$ for some r

$\xleftarrow{C}$

The De-obfuscation oracle $\mathbb{O}(\cdot, \phi)$

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Adversary

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$

Challenger

$b \xleftarrow{\$} \{0,1\}$

(C_0, C_1, z)

$\widehat{C}$

$\widehat{C} \leftarrow Obf(C_b, \phi)$

$\widehat{C}$

C

Samp must be "admissible ϕ-satisfying "sampler:

$C_0, C_1 \in \mathscr{C}$ and $\phi(C_0) = \phi(C_1) = 1$

2. $C_0 \approx_c C_1$ for any black-box PPT distinguisher

1. If $\widehat{C} = \widehat{C}$, output $\perp$.

2. Run $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$

3. If $\widetilde{C} \neq \perp$, return C such that $\widetilde{C} = \mathcal{O}(C; r)$ for some r

The De-obfuscation oracle $\mathbb{O}(\cdot, \phi)$

De-obfuscation queries can be made adaptively and in arbitrary order

Defining COA Security: Defn. 1

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate.

Adversary

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$

Samp must be "admissible ϕ-satisfying "sampler:

$C_0, C_1 \in \mathscr{C}$ and $\phi(C_0) = \phi(C_1) = 1$

2. $C_0 \approx_c C_1$ for any black-box PPT distinguisher

De-obfuscation queries can be made adaptively and in arbitrary order

Challenger

$b \xleftarrow{\$} \{0,1\}$

$\xrightarrow{(C_0, C_1, z)}$

$\widehat{C} \leftarrow Obf(C_b, \phi)$

$\xleftarrow{\widehat{C}}$

$\xrightarrow{\widehat{C}}$

1. If $\widehat{C} = \widehat{C}$, output $\perp$.
2. Run $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$
3. If $\widetilde{C} \neq \perp$, return C such that $\widetilde{C} = \mathscr{O}(C; r)$ for some r

The De-obfuscation oracle $\mathbb{O}(\cdot, \phi)$

$\xleftarrow{C}$

$\xrightarrow{b'}$

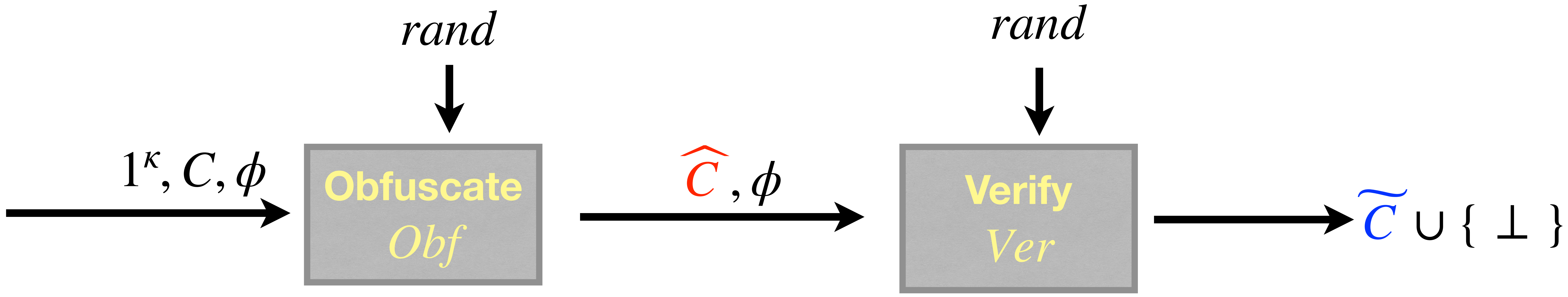
COA Fortification for Obfuscation: Defn. 2

COA Fortification for Obfuscation: Defn. 2

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $\mathscr{O}$ be an *injective* obfuscation scheme.

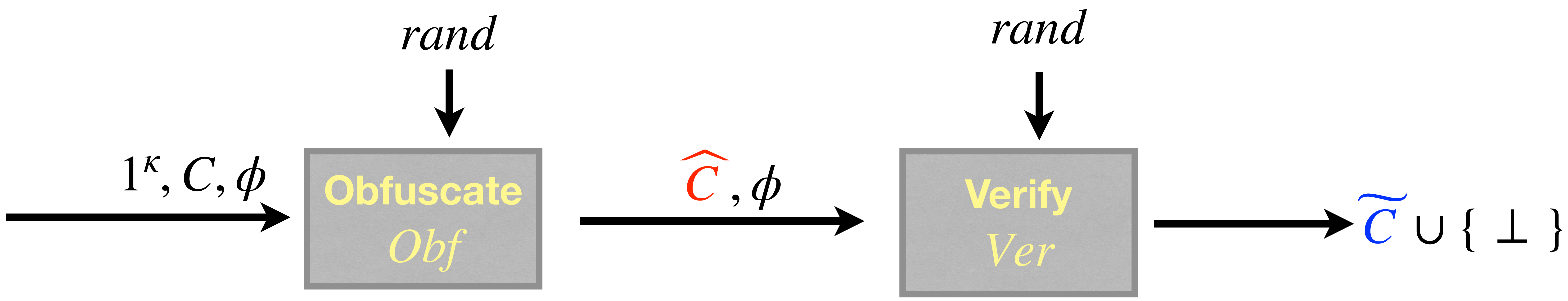
COA Fortification for Obfuscation: Defn. 2

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $\mathscr{O}$ be an *injective* obfuscation scheme.



COA Fortification for Obfuscation: Defn. 2

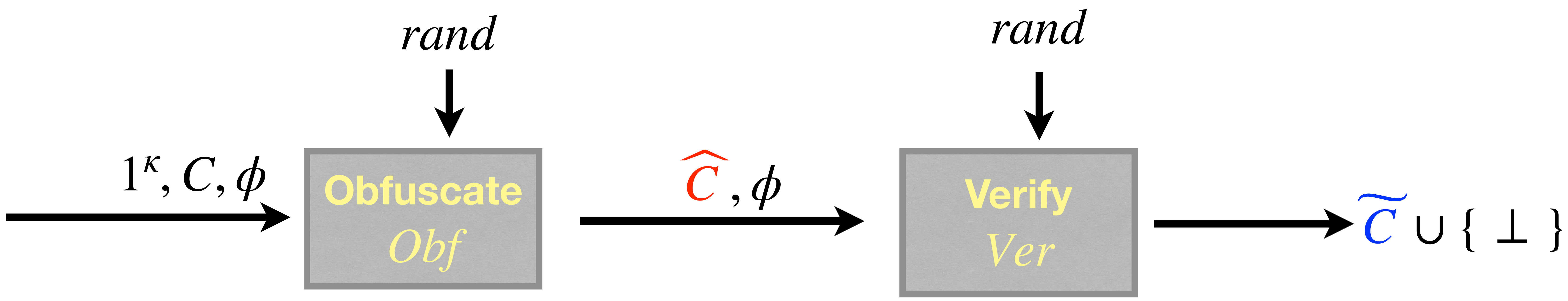
Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $\mathcal{O}$ be an *injective* obfuscation scheme.



☑ **Correctness**: For a legitimate C, i.e., $\phi(C) = 1$, $\widetilde{C} = \mathcal{O}(C; r)$.

COA Fortification for Obfuscation: Defn. 2

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $\mathscr{O}$ be an *injective* obfuscation scheme.

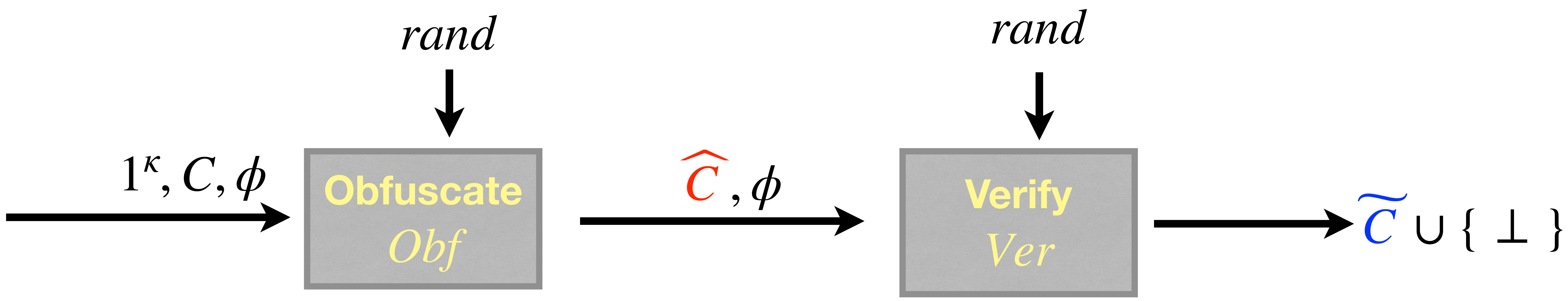


☑ **Correctness**: For a legitimate C, i.e., $\phi(C) = 1$, $\widetilde{C} = \mathscr{O}(C; r)$.

☑ **Soundness**: For any string $\widehat{C}$, if Verify outputs some $\widetilde{C}$, i.e., $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$ s.t $\widetilde{C} \neq \perp$ => there exists an underlying circuit $C \in \mathscr{C}$ such that $\phi(C) = 1$ and $\widetilde{C} = \mathscr{O}(C; r)$.

COA Fortification for Obfuscation: Defn. 2

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $\mathcal{O}$ be an *injective* obfuscation scheme.



☑ **Correctness**: For a legitimate C, i.e., $\phi(C) = 1$, $\widetilde{C} = \mathcal{O}(C; r)$.

☑ **Soundness**: For any string $\widehat{C}$, if Verify outputs some $\widetilde{C}$, i.e., $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$ s.t $\widetilde{C} \neq \perp$ => there exists an underlying circuit $C \in \mathscr{C}$ such that $\phi(C) = 1$ and $\widetilde{C} = \mathcal{O}(C; r)$.

☑ **COA security**: For "sufficiently similar" circuits C_0 and C_1, and given access to "*de-obfuscation oracle*" $\mathcal{O}^{-1}(\cdot)$:

$$Obf(C_0, \phi) \approx_c Obf(C_1, \phi) \implies \mathcal{O}(C_0) \approx_c \mathcal{O}(C_1)$$

COA Fortification for obfuscation: Defn. 2

COA Fortification for obfuscation: Defn. 2

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $\mathscr{O}$ be an *injective* obfuscation scheme.

COA Fortification for obfuscation: Defn. 2

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $\mathscr{O}$ be an *injective* obfuscation scheme.

Adversary

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$

Challenger

$b \xleftarrow{\$} \{0,1\}$

(C_0, C_1, z) →

← $\widehat{C}$

$\widehat{C} \leftarrow Obf(C_b, \phi)$

$\widehat{C}$ →

Samp must be an admissible ϕ-satisfying sampler

1. If $\widehat{C} = \widehat{C}$, output $\perp$.

2. Run $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$

3. If $\widetilde{C} \neq \perp$, return C such that $\widetilde{C} = \mathscr{O}(C; r)$ for some r

← C

De-obfuscation queries can be made adaptively and in arbitrary order

b' →

The De-obfuscation oracle $\mathbb{O}(\cdot, \phi)$

COA Fortification for obfuscation: Defn. 2

Consider a class of circuits $\mathscr{C}$ and let ϕ be a predicate. Let $\mathcal{O}$ be an *injective* obfuscation scheme.

Adversary

Challenger

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$ $\xrightarrow{(C_0, C_1, z)}$ $b \xleftarrow{\$} \{0,1\}$

$\xleftarrow{\widehat{C}}$ $\widehat{C} \leftarrow Obf(C_b, \phi)$

Samp must be an admissible ϕ-satisfying sampler

$\xrightarrow{\widehat{C}}$

1. If $\widehat{C} = \widehat{C}$, output $\perp$.
2. Run $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$
3. If $\widetilde{C} \neq \perp$, return C such that $\widetilde{C} = \mathcal{O}(C; r)$ for some r

$\xleftarrow{C}$

De-obfuscation queries can be made adaptively and in arbitrary order

$\xrightarrow{b'}$

The De-obfuscation oracle $\mathbb{O}(\cdot, \phi)$

Adversary

Challenger

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$ $\xrightarrow{(C_0, C_1, z)}$ $b \xleftarrow{\$} \{0,1\}$

$\overline{C} \leftarrow \mathcal{O}(1^\kappa, C_b)$

Samp must be an admissible sampler

$\xleftarrow{\overline{C}}$

$\xrightarrow{b'}$

COA Fortification for obfuscation: Defn. 2

Consider a class of circ… bfuscation scheme.

COA Fortification for injective PiO => Def. 1

Adversary

Challenger

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$ $\xrightarrow{(C_0, C_1, z)}$ $b \xleftarrow{\$} \{0,1\}$

$\xleftarrow{\widehat{C}}$ $\widehat{C} \leftarrow Obf(C_b, \phi)$

$\xrightarrow{\widehat{C}}$

$\xleftarrow{C}$

$\xrightarrow{b'}$

Samp must be an admissible ϕ-satisfying sampler

De-obfuscation queries can be made adaptively and in arbitrary order

1. If $\widehat{C} = \widehat{C}$, output $\perp$.
2. Run $\widetilde{C} \leftarrow Ver(\widehat{C}, \phi)$
3. If $\widetilde{C} \neq \perp$, return C such that $\widetilde{C} = \mathcal{O}(C; r)$ for some r

The De-obfuscation oracle $\mathbb{O}(\cdot, \phi)$

Adversary

Challenger

$(C_0, C_1, z) \leftarrow \mathsf{Samp}(1^\kappa)$ $\xrightarrow{(C_0, C_1, z)}$ $b \xleftarrow{\$} \{0,1\}$

$\overline{C} \leftarrow \mathcal{O}(1^\kappa, C_b)$

$\xleftarrow{\overline{C}}$

$\xrightarrow{b'}$

Samp must be an admissible sampler

Applications of COA Obfuscation

Complete CCA (CCCA)-secure Public Key Encryption

Complete CCA (CCCA)-secure Public Key Encryption

- Enhance CCA-secure PKE with the ability of the adversary to submit (pk_i, c_i) to the decryption oracle, where pk_i can be arbitrarily related to the challenge public key pk^*.

Complete CCA (CCCA)-secure Public Key Encryption

- Enhance CCA-secure PKE with the ability of the adversary to submit (pk_i, c_i) to the decryption oracle, where pk_i can be arbitrarily related to the challenge public key pk^*.

- Strengthening of the notion of completely non-malleable encryption [F'05, VV'08] (when defined appropriately)

Complete CCA (CCCA)-secure Public Key Encryption

- Enhance CCA-secure PKE with the ability of the adversary to submit (pk_i, c_i) to the decryption oracle, where pk_i can be arbitrarily related to the challenge public key pk^*.

- Strengthening of the notion of completely non-malleable encryption [F'05, VV'08] (when defined appropriately)

- [F'05] showed that completely non-malleable PKE is impossible to construct in the "plain" model using any black-box reduction to a poly-time falsifiable hardness assumption.

Complete CCA (CCCA)-secure Public Key Encryption

- Enhance CCA-secure PKE with the ability of the adversary to submit (pk_i, c_i) to the decryption oracle, where pk_i can be arbitrarily related to the challenge public key pk^*.

- Strengthening of the notion of completely non-malleable encryption [F'05, VV'08] (when defined appropriately)

- [F'05] showed that completely non-malleable PKE is impossible to construct in the "plain" model using any black-box reduction to a poly-time falsifiable hardness assumption.

- We bypass this impossibility result by constructing CCCA secure PKE using sub-exponential hardness assumptions.

Complete CCA (CCCA)-secure Public Key Encryption

- Enhance CCA-secure PKE with the ability of the adversary to submit (pk_i, c_i) to the decryption oracle, where pk_i can be arbitrarily related to the challenge public key pk^*.

- Strengthening of the notion of completely non-malleable encryption [F'05, VV'08] (when defined appropriately)

- [F'05] showed that completely non-malleable PKE is impossible to construct in the "plain" model using any black-box reduction to a poly-time falsifiable hardness assumption.

- We bypass this impossibility result by constructing CCCA secure PKE using sub-exponential hardness assumptions.

- **Unique Decryptability**:

Complete CCA (CCCA)-secure Public Key Encryption

- Enhance CCA-secure PKE with the ability of the adversary to submit (pk_i, c_i) to the decryption oracle, where pk_i can be arbitrarily related to the challenge public key pk^*.

- Strengthening of the notion of completely non-malleable encryption [F'05, VV'08] (when defined appropriately)

- [F'05] showed that completely non-malleable PKE is impossible to construct in the "plain" model using any black-box reduction to a poly-time falsifiable hardness assumption.

- We bypass this impossibility result by constructing CCCA secure PKE using sub-exponential hardness assumptions.

- **<u>Unique Decryptability</u>**:

    - A string $pk \in \{0,1\}^{poly(\kappa)}$ is "useless" if $\mathsf{Pr}[\mathsf{Enc}(pk, m) \neq \perp] \leq \mu(\kappa)$, for any message $m \in \mathscr{M}$

Complete CCA (CCCA)-secure Public Key Encryption

- Enhance CCA-secure PKE with the ability of the adversary to submit (pk_i, c_i) to the decryption oracle, where pk_i can be arbitrarily related to the challenge public key pk^*.

- Strengthening of the notion of completely non-malleable encryption [F'05, VV'08] (when defined appropriately)

- [F'05] showed that completely non-malleable PKE is impossible to construct in the "plain" model using any black-box reduction to a poly-time falsifiable hardness assumption.

- We bypass this impossibility result by constructing CCCA secure PKE using sub-exponential hardness assumptions.

- **<u>Unique Decryptability</u>**:

    - A string $pk \in \{0,1\}^{poly(\kappa)}$ is "useless" if $\mathsf{Pr}[\mathsf{Enc}(pk, m) \neq \perp] \leq \mu(\kappa)$, for any message $m \in \mathcal{M}$

    - $\forall$ "non-useless" keys pk, $\exists$ "opening" sk s.t. $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m$ w.h.p. Further, $\exists$ pk' s.t. $(pk', sk) \leftarrow \text{KeyGen}(1^\kappa; r)$.

Complete CCA-secure Public Key Encryption

Complete CCA-secure Public Key Encryption

Adversary

Challenger

Complete CCA-secure Public Key Encryption

Adversary

Challenger

$$\xleftarrow{\quad pk^* \quad} \qquad (pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^\kappa)$$

Complete CCA-secure Public Key Encryption

Adversary

Challenger

pk^*

$(pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^\kappa)$

(pk_i, c_i)

1. Check if pk_i is "useless"
2. If "not useless" find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

Complete CCA-secure Public Key Encryption

Adversary

Challenger

$(pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^\kappa)$

$\xleftarrow{pk^*}$

$\xrightarrow{(pk_i, c_i)}$

1. Check if pk_i is "useless"
2. If "not useless" find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

$\xleftarrow{m_i}$

Complete CCA-secure Public Key Encryption

Adversary

Challenger

$(pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^\kappa)$

$\xleftarrow{\quad pk^* \quad}$

$\xrightarrow{\quad (pk_i, c_i) \quad}$

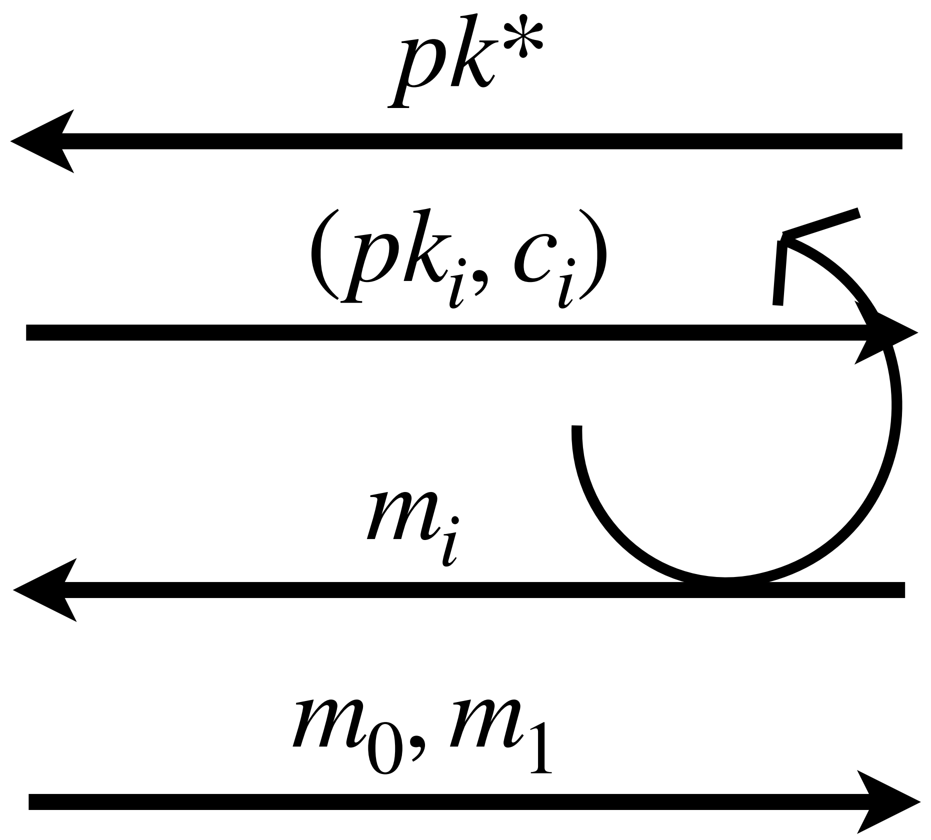
$\xleftarrow{\quad m_i \quad}$

1. Check if pk_i is "useless"
2. If "not useless" find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

Complete CCA-secure Public Key Encryption

Adversary

Challenger



$(pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^\kappa)$

1. Check if pk_i is "useless"
2. If "not useless" find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

Complete CCA-secure Public Key Encryption

Adversary

Challenger

pk^* ← $(pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^\kappa)$

(pk_i, c_i) →

1. Check if pk_i is "useless"
2. If "not useless" find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

m_i ←

m_0, m_1 →

c^* ← $c^* \leftarrow \mathsf{Enc}(pk^*, m_b)$

Complete CCA-secure Public Key Encryption

Adversary

Challenger

pk^* ← | $(pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^\kappa)$

(pk_i, c_i) →

1. Check if pk_i is "useless"
2. If "not useless" find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

m_i ←

m_0, m_1 →

c^* ← | $c^* \leftarrow \mathsf{Enc}(pk^*, m_b)$

(pk_j, c_j) →

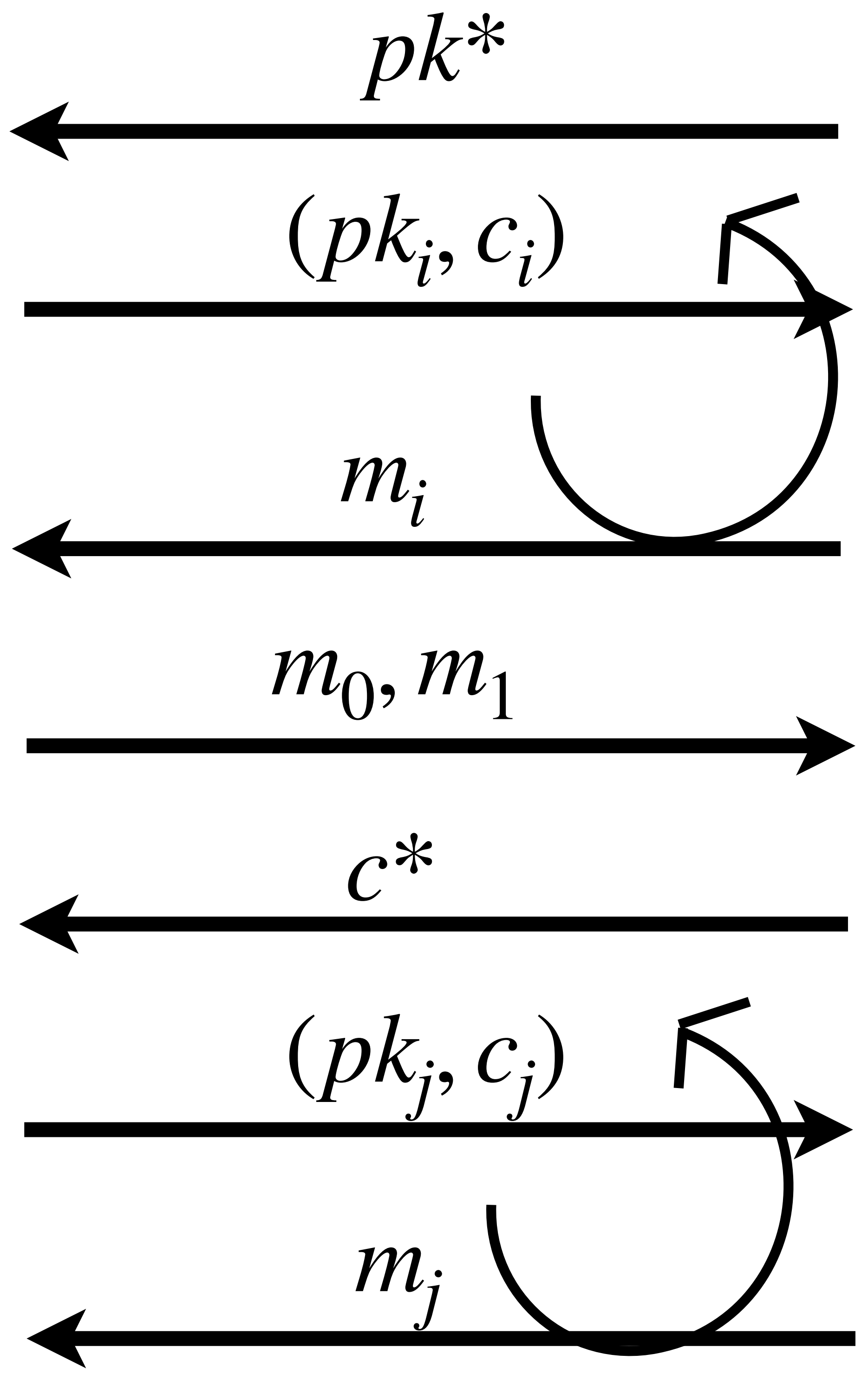
1. Check if pk_j is "useless"
2. If "not useless" and $(pk_j, c_j) \neq (pk^*, c^*)$, find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

m_j ←

Complete CCA-secure Public Key Encryption

Adversary

Challenger



$(pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^\kappa)$

1. Check if pk_i is "useless"
2. If "not useless" find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

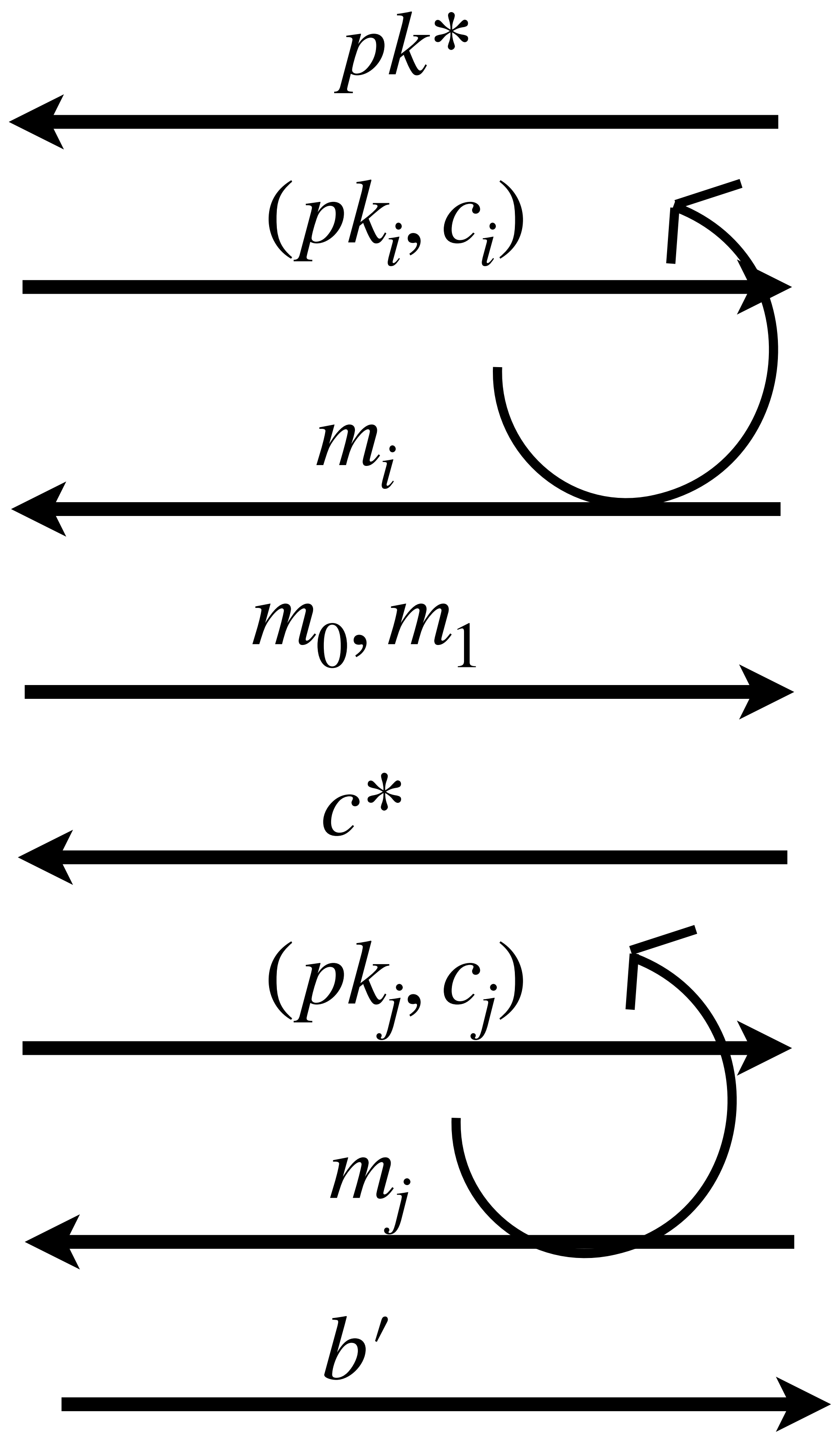
$c^* \leftarrow \mathsf{Enc}(pk^*, m_b)$

1. Check if pk_j is "useless"
2. If "not useless" and $(pk_j, c_j) \neq (pk^*, c^*)$, find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

Complete CCA-secure Public Key Encryption

Adversary

Challenger



$(pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^\kappa)$

1. Check if pk_i is "useless"
2. If "not useless" find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

$c^* \leftarrow \mathsf{Enc}(pk^*, m_b)$

1. Check if pk_j is "useless"
2. If "not useless" and $(pk_j, c_j) \neq (pk^*, c^*)$, find opening sk and return $m_i := \mathsf{Dec}(sk, c_i)$

Constructing CCCA-secure PKE

Constructing CCCA-secure PKE

1. Let $c\mathcal{O} = (Obf, Ver)$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate ϕ: $\phi(C)$ attest that C is of the form of P.

Constructing CCCA-secure PKE

1. Let $c\mathcal{O} = (Obf, Ver)$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate ϕ: $\phi(C)$ attest that C is of the form of P.

2. Let $G : \{0,1\}^{\kappa} \rightarrow \{0,1\}^{2\kappa}$ be a PRG, and $F_1 : \{0,1\}^{2\kappa} \rightarrow \{0,1\}$ and $F_2 : \{0,1\}^{2\kappa+1} \rightarrow \{0,1\}^{\kappa}$ be two puncturable PRFs.

Constructing CCCA-secure PKE

1. Let $c\mathcal{O} = (Obf, Ver)$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate ϕ: $\phi(C)$ attest that C is of the form of P.

2. Let $G : \{0,1\}^{\kappa} \to \{0,1\}^{2\kappa}$ be a PRG, and $F_1 : \{0,1\}^{2\kappa} \to \{0,1\}$ and $F_2 : \{0,1\}^{2\kappa+1} \to \{0,1\}^{\kappa}$ be two puncturable PRFs.

- KeyGen(1^{κ}): Sample keys K_1 and K_2 for F_1 and F_2 resp., output $pk = \widehat{P} \leftarrow Obf(P, \phi)$, and $sk = (K_1, K_2)$.

Constructing CCCA-secure PKE

1. Let $c\mathcal{O} = (Obf, Ver)$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate ϕ: $\phi(C)$ attest that C is of the form of P.

2. Let $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ be a PRG, and $F_1 : \{0,1\}^{2\kappa} \to \{0,1\}$ and $F_2 : \{0,1\}^{2\kappa+1} \to \{0,1\}^\kappa$ be two puncturable PRFs.

- KeyGen(1^κ): Sample keys K_1 and K_2 for F_1 and F_2 resp., output $pk = \widehat{P} \leftarrow Obf(P, \phi)$, and $sk = (K_1, K_2)$.

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Constructing CCCA-secure PKE

1. Let $c\mathcal{O} = (Obf, Ver)$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate ϕ: $\phi(C)$ attest that C is of the form of P.

2. Let $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ be a PRG, and $F_1 : \{0,1\}^{2\kappa} \to \{0,1\}$ and $F_2 : \{0,1\}^{2\kappa+1} \to \{0,1\}^\kappa$ be two puncturable PRFs.

- KeyGen(1^κ): Sample keys K_1 and K_2 for F_1 and F_2 resp., output $pk = \widehat{P} \leftarrow Obf(P, \phi)$, and $sk = (K_1, K_2)$.

- Enc(pk, m): Run $\widetilde{P} \leftarrow Ver(\widehat{P}, \phi)$; sample random r and output $\widetilde{P}(m, r)$.

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Constructing CCCA-secure PKE

1. Let $c\mathcal{O} = (Obf, Ver)$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate ϕ: $\phi(C)$ attest that C is of the form of P.

2. Let $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ be a PRG, and $F_1 : \{0,1\}^{2\kappa} \to \{0,1\}$ and $F_2 : \{0,1\}^{2\kappa+1} \to \{0,1\}^\kappa$ be two puncturable PRFs.

- KeyGen(1^κ): Sample keys K_1 and K_2 for F_1 and F_2 resp., output $pk = \widehat{P} \leftarrow Obf(P, \phi)$, and $sk = (K_1, K_2)$.
- Enc(pk, m): Run $\widetilde{P} \leftarrow Ver(\widehat{P}, \phi)$; sample random r and output $\widetilde{P}(m, r)$.
- Dec$(sk = (K_1, K_2), c = (c_1, c_2, c_3))$: Verify authenticity of c_3 and recover m.

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Constructing CCCA-secure PKE

1. Let $c\mathcal{O} = (Obf, Ver)$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate ϕ: $\phi(C)$ attest that C is of the form of P.

2. Let $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ be a PRG, and $F_1 : \{0,1\}^{2\kappa} \to \{0,1\}$ and $F_2 : \{0,1\}^{2\kappa+1} \to \{0,1\}^\kappa$ be two puncturable PRFs.

- KeyGen(1^κ): Sample keys K_1 and K_2 for F_1 and F_2 resp., output $pk = \widehat{P} \leftarrow Obf(P, \phi)$, and $sk = (K_1, K_2)$.

- Enc(pk, m): Run $\widetilde{P} \leftarrow Ver(\widehat{P}, \phi)$; sample random r and output $\widetilde{P}(m, r)$.

- Dec$(sk = (K_1, K_2), c = (c_1, c_2, c_3))$: Verify authenticity of c_3 and recover m.

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1, K_2}

- **Unique Decryptability**: Follows from soundness of $c\mathcal{O}$ + (perfect)injectivity of $i\mathcal{O}$

Constructing CCCA-secure PKE

1. Let $c\mathcal{O} = (Obf, Ver)$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate ϕ: $\phi(C)$ attest that C is of the form of P.

2. Let $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ be a PRG, and $F_1 : \{0,1\}^{2\kappa} \to \{0,1\}$ and $F_2 : \{0,1\}^{2\kappa+1} \to \{0,1\}^\kappa$ be two puncturable PRFs.

- KeyGen(1^κ): Sample keys K_1 and K_2 for F_1 and F_2 resp., output $pk = \widehat{P} \leftarrow Obf(P, \phi)$, and $sk = (K_1, K_2)$.

- Enc(pk, m): Run $\widetilde{P} \leftarrow Ver(\widehat{P}, \phi)$; sample random r and output $\widetilde{P}(m, r)$.

- Dec($sk = (K_1, K_2), c = (c_1, c_2, c_3)$): Verify authenticity of c_3 and recover m.

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

- **Unique Decryptability**: Follows from soundness of $c\mathcal{O}$ + (perfect)injectivity of $i\mathcal{O}$
  - If $\widetilde{P} \neq \perp \implies \exists P' = P'_{K'_1,K'_2}$ s.t. $\widetilde{P} = i\mathcal{O}(P'; r)$ and hence can recover P' using $i\mathcal{O}^{-1}(\widetilde{P})$. Use (K'_1, K'_2) to decrypt.

Zooming into the proof of CCCA-secure PKE

Zooming into the proof of CCCA-secure PKE

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Zooming into the proof of CCCA-secure PKE

Either of the following two cases may arise (for each decryption query):

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Zooming into the proof of CCCA-secure PKE

Either of the following two cases may arise (for each decryption query):

1. $(pk, c) \neq (pk^*, c^*)$ and $pk_i = pk^*$: In this case this can be reduced to the CCA-security of the [SW'14] construction.

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Zooming into the proof of CCCA-secure PKE

Either of the following two cases may arise (for each decryption query):

1. $(pk, c) \neq (pk^*, c^*)$ and $pk_i = pk^*$: In this case this can be reduced to the CCA-security of the [SW'14] construction.

2. $(pk, c) \neq (pk^*, c^*)$ and $pk \neq pk^*$: In this case, letting $pk = \widehat{P}$:

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Zooming into the proof of CCCA-secure PKE

Either of the following two cases may arise (for each decryption query):

1. $(pk, c) \neq (pk^*, c^*)$ and $pk_i = pk^*$: In this case this can be reduced to the CCA-security of the [SW'14] construction.

2. $(pk, c) \neq (pk^*, c^*)$ and $pk \neq pk^*$: In this case, letting $pk = \widehat{P}$:

- Let $\widetilde{P} \leftarrow Ver(\widehat{P}, \phi)$,

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Zooming into the proof of CCCA-secure PKE

Either of the following two cases may arise (for each decryption query):

1. $(pk, c) \neq (pk^*, c^*)$ and $pk_i = pk^*$: In this case this can be reduced to the CCA-security of the [SW'14] construction.

2. $(pk, c) \neq (pk^*, c^*)$ and $pk \neq pk^*$: In this case, letting $pk = \widehat{P}$:

- Let $\widetilde{P} \leftarrow Ver(\widehat{P}, \phi)$,

- If $\widetilde{P} \neq \perp \implies \exists P' = P'_{K'_1,K'_2}$ such that $\widetilde{P} \leftarrow i\mathcal{O}(1^\kappa, P')$.

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Zooming into the proof of CCCA-secure PKE

Either of the following two cases may arise (for each decryption query):

1. $(pk, c) \neq (pk^*, c^*)$ and $pk_i = pk^*$: In this case this can be reduced to the CCA-security of the [SW'14] construction.

2. $(pk, c) \neq (pk^*, c^*)$ and $pk \neq pk^*$: In this case, letting $pk = \widehat{P}$:

- Let $\widetilde{P} \leftarrow Ver(\widehat{P}, \phi)$,
- If $\widetilde{P} \neq \perp \implies \exists P' = P'_{K'_1,K'_2}$ such that $\widetilde{P} \leftarrow i\mathcal{O}(1^\kappa, P')$.
- Recover P' using $i\mathcal{O}^{-1}(\widetilde{P})$, use $sk' = (K'_1, K'_2)$ to decrypt c. (follows from the COA fortification of injective $i\mathcal{O}$ w.r.t. predicate ϕ).

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1,K_2}

Zooming into the proof of CCCA-secure PKE

Either of the following two cases may arise (for each decryption query):

1. $(pk, c) \neq (pk^*, c^*)$ and $pk_i = pk^*$: In this case this can be reduced to the CCA-security of the [SW'14] construction.

2. $(pk, c) \neq (pk^*, c^*)$ and $pk \neq pk^*$: In this case, letting $pk = \widehat{P}$:

- Let $\widetilde{P} \leftarrow Ver(\widehat{P}, \phi)$,
- If $\widetilde{P} \neq \perp \implies \exists P' = P'_{K'_1, K'_2}$ such that $\widetilde{P} \leftarrow i\mathcal{O}(1^\kappa, P')$.
- Recover P' using $i\mathcal{O}^{-1}(\widetilde{P})$, use $sk' = (K'_1, K'_2)$ to decrypt c. (follows from the COA fortification of injective $i\mathcal{O}$ w.r.t. predicate ϕ).

1. Constants: K_1, K_2
2. Input: m, r

a. $c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m$
b. $c_3 = F_2(K_2, c_1 || c_2)$
c. Output $c = (c_1, c_2, c_3)$

Program P_{K_1, K_2}

Construction of COA Obfuscation

Non-Interactive Distributional Indistinguishable (NIDI) argument [K'21]

Non-Interactive Distributional Indistinguishable (NIDI) argument [K'21]

- A NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ consists of two algorithms (P, V) such that:

Non-Interactive Distributional Indistinguishable (NIDI) argument [K'21]

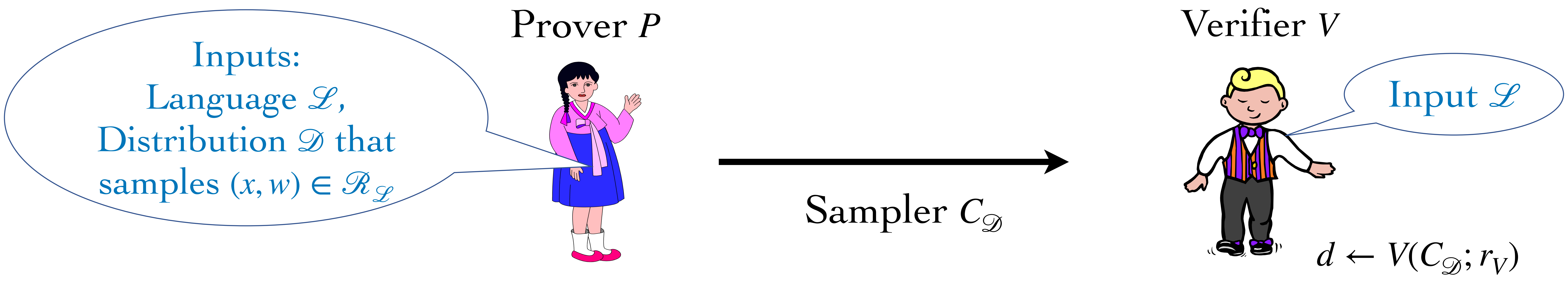
- A NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ consists of two algorithms (P, V) such that:

Prover P

Verifier V

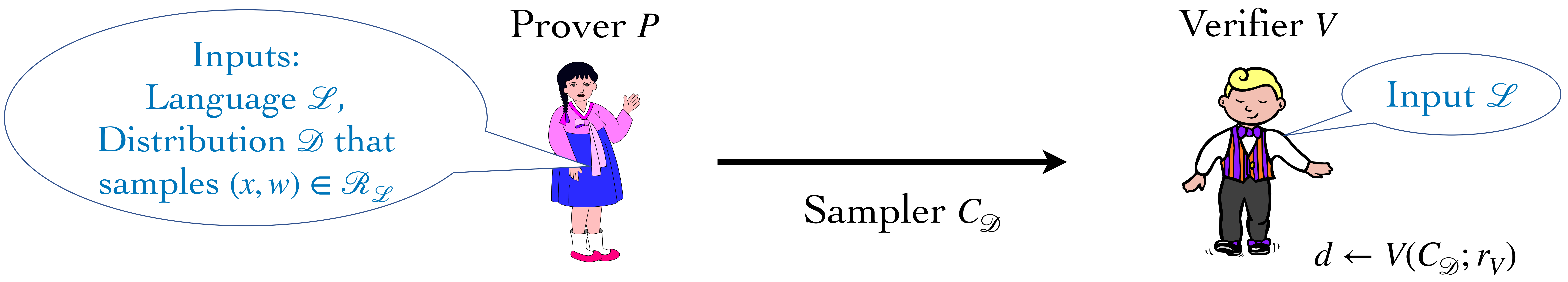
Non-Interactive Distributional Indistinguishable (NIDI) argument [K'21]

- A NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ consists of two algorithms (P, V) such that:



Non-Interactive Distributional Indistinguishable (NIDI) argument [K'21]

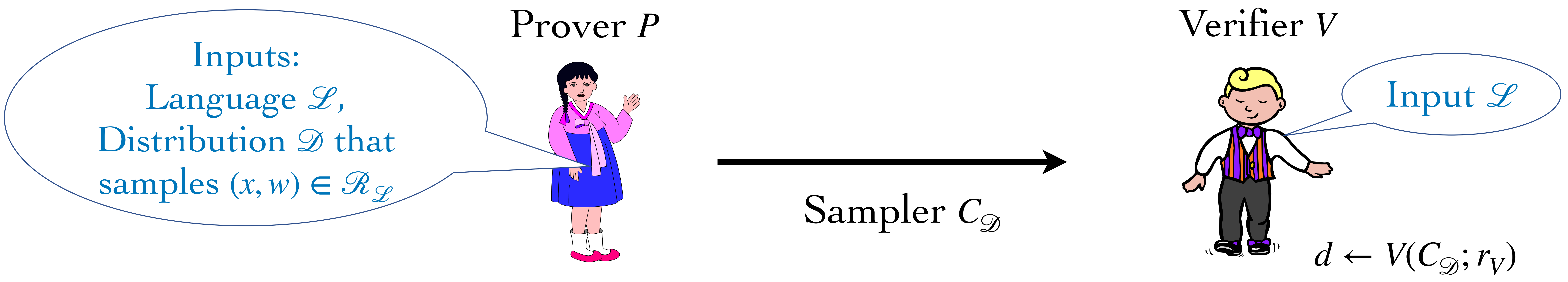
- A NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ consists of two algorithms (P, V) such that:



- **Completeness**: d is in $Supp(\mathscr{D})|_x$

Non-Interactive Distributional Indistinguishable (NIDI) argument [K'21]

- A NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ consists of two algorithms (P, V) such that:

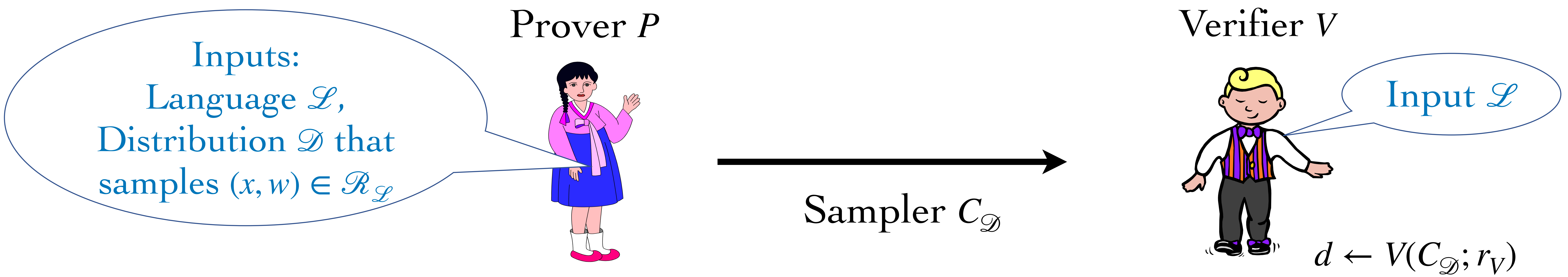


- **Completeness**: d is in $Supp(\mathscr{D})|_x$

- **Soundness**: If $d \neq \perp$, then $d \in \mathscr{L}$

Non-Interactive Distributional Indistinguishable (NIDI) argument [K'21]

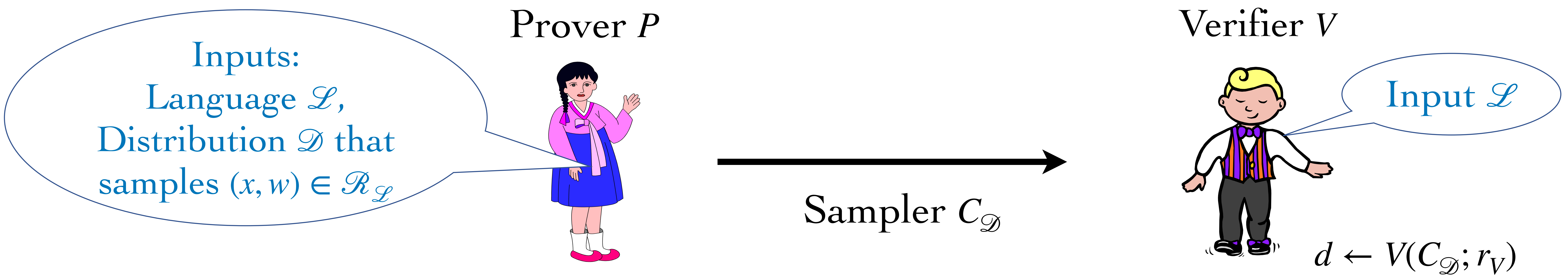
- A NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ consists of two algorithms (P, V) such that:



- **Completeness**: d is in $Supp(\mathscr{D})|_x$
- **Soundness**: If $d \neq \perp$, then $d \in \mathscr{L}$
- **Privacy**: For all D_1, D_2 s.t. $D_1|_x \approx_c D_2|_x$, we have $C_{\mathscr{D}_1} \approx_c C_{\mathscr{D}_2}$

Non-Interactive Distributional Indistinguishable (NIDI) argument [K'21]

- A NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ consists of two algorithms (P, V) such that:



- **Completeness**: d is in $Supp(\mathscr{D})|_x$
- **Soundness**: If $d \neq \perp$, then $d \in \mathscr{L}$
- **Privacy**: For all D_1, D_2 s.t. $D_1|_x \approx_c D_2|_x$, we have $C_{\mathscr{D}_1} \approx_c C_{\mathscr{D}_2}$

Theorem: Assuming sub-exp $i\mathscr{O}$ and sub-exp secure OWF, there exists NIDI arguments for NP.

Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

- A r-NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ w.r.t. a “finite" oracle $\mathbb{O}$ consists of two algorithms (P, V) such that:

Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

- A r-NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ w.r.t. a "finite" oracle $\mathbb{O}$ consists of two algorithms (P, V) such that:

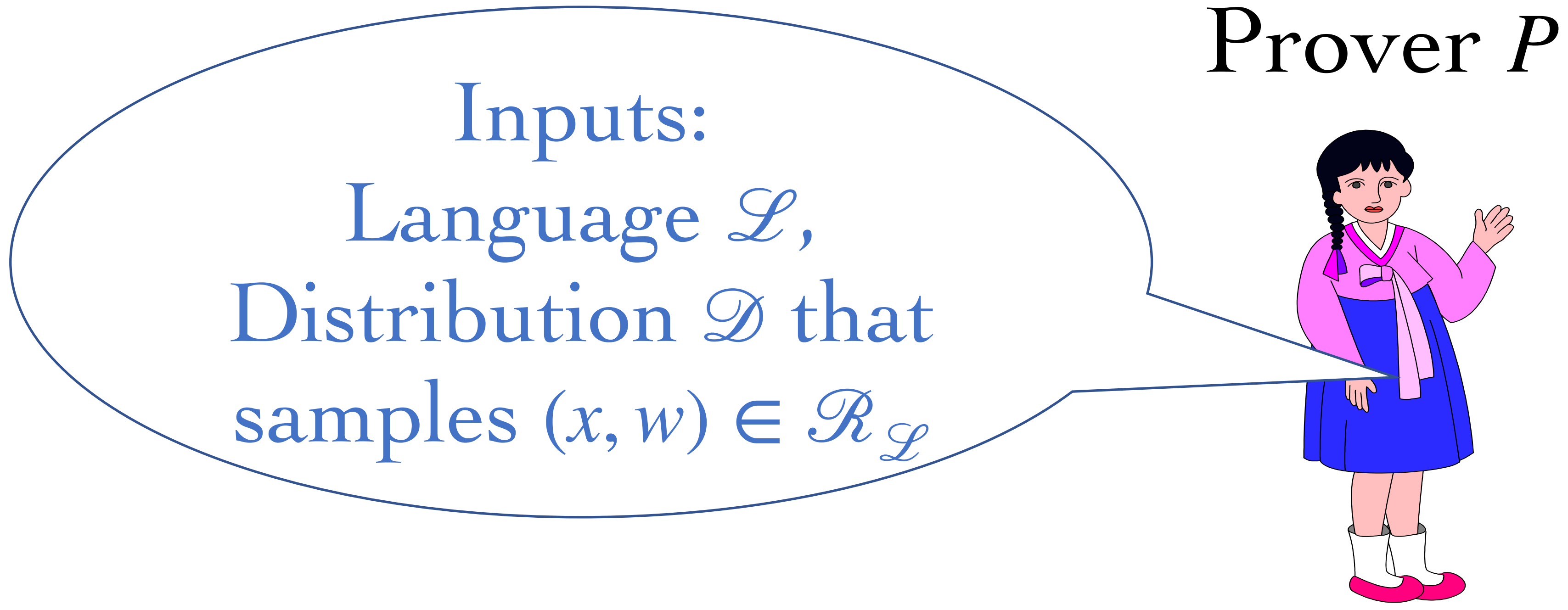
Prover P

Verifier V

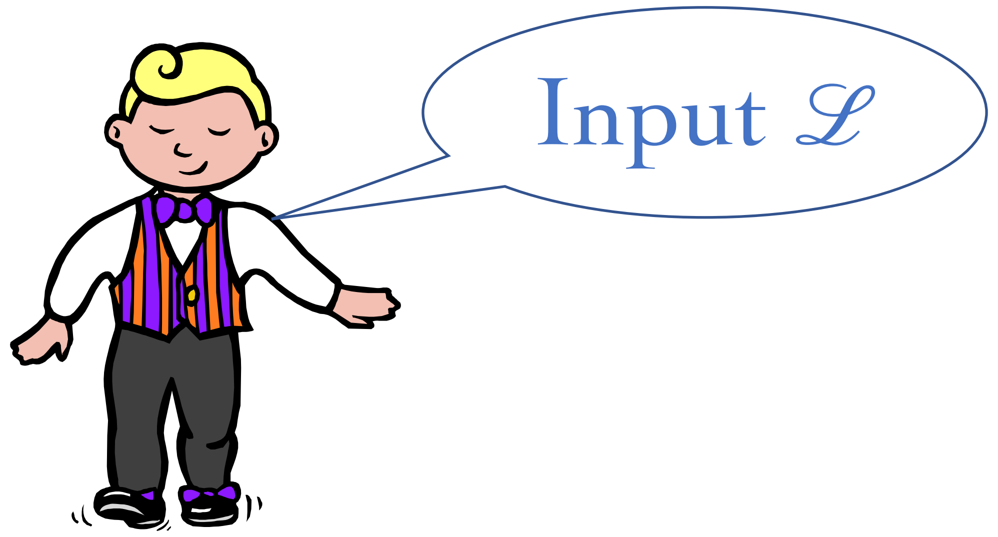
Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

- A r-NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ w.r.t. a “finite" oracle $\mathbb{O}$ consists of two algorithms (P, V) such that:

Prover P

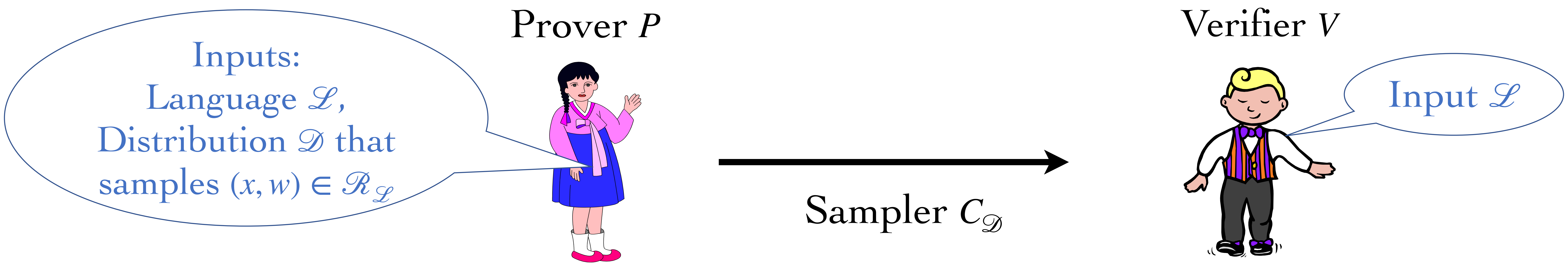


Verifier V



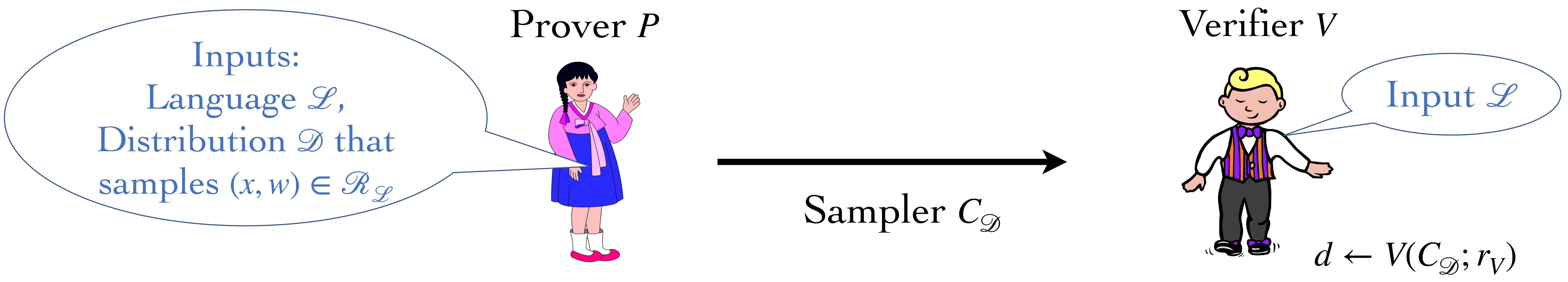
Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

- A r-NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ w.r.t. a "finite" oracle $\mathbb{O}$ consists of two algorithms (P, V) such that:



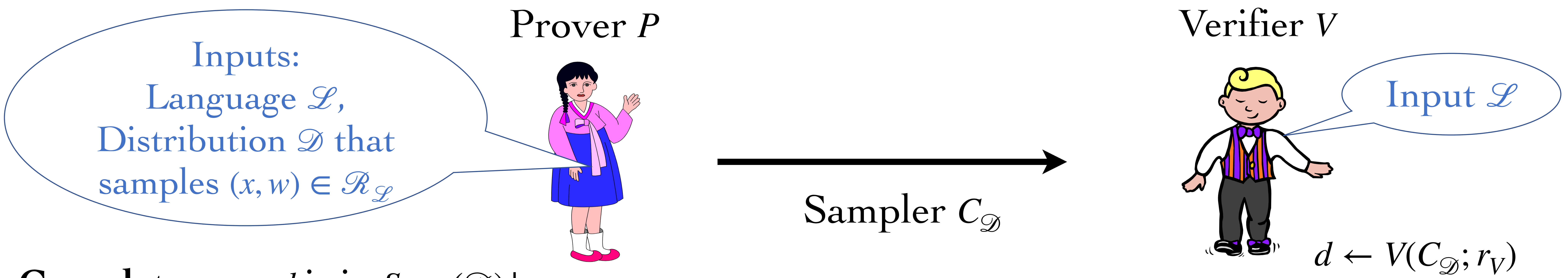
Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

- A r-NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ w.r.t. a "finite" oracle $\mathbb{O}$ consists of two algorithms (P, V) such that:



Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

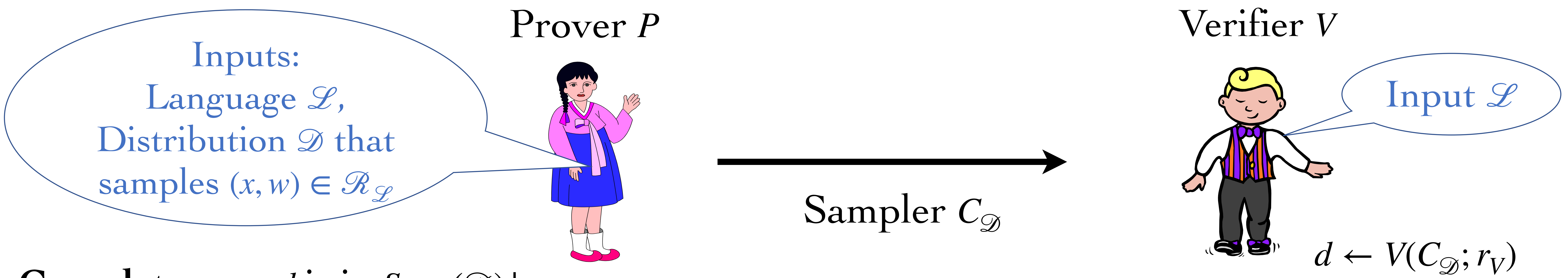
- A r-NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ w.r.t. a "finite" oracle $\mathbb{O}$ consists of two algorithms (P, V) such that:



- **Completeness**: d is in $Supp(\mathscr{D})|_x$

Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

- A r-NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ w.r.t. a “finite" oracle $\mathbb{O}$ consists of two algorithms (P, V) such that:



- **Completeness**: d is in $Supp(\mathscr{D})|_x$
- **Soundness**: If $d \neq \perp$, then $d \in \mathscr{L}$

Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

- A r-NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ w.r.t. a "finite" oracle $\mathbb{O}$ consists of two algorithms (P, V) such that:

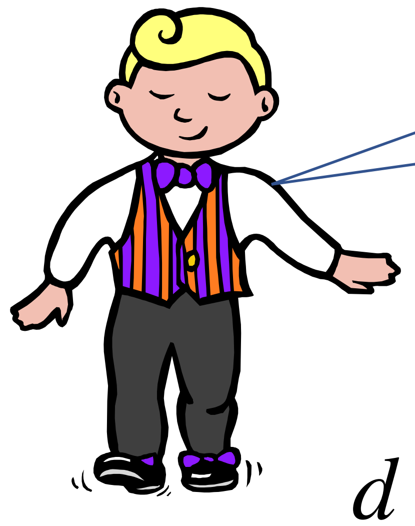
Prover P

Inputs:
Language $\mathscr{L}$,
Distribution $\mathscr{D}$ that
samples $(x, w) \in \mathscr{R}_{\mathscr{L}}$

Sampler $C_{\mathscr{D}}$

Verifier V

Input $\mathscr{L}$



$d \leftarrow V(C_{\mathscr{D}}; r_V)$

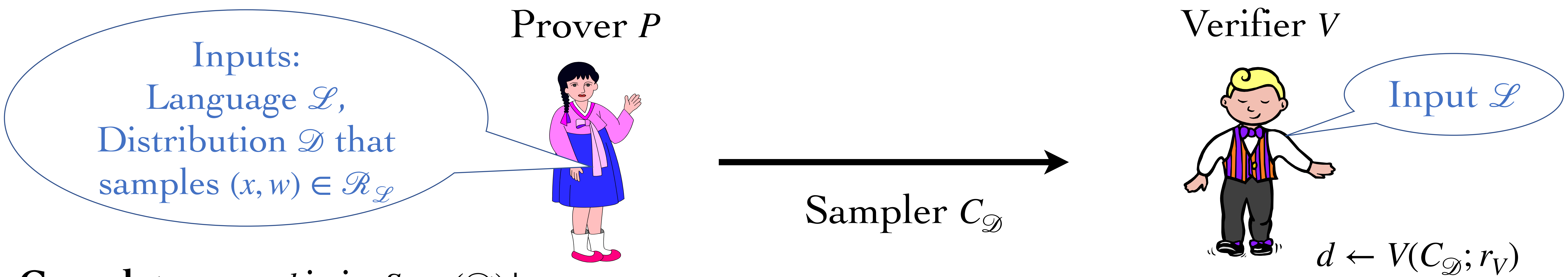
- **Completeness**: d is in $Supp(\mathscr{D})|_x$

- **Soundness**: If $d \neq \perp$, then $d \in \mathscr{L}$

- **Robustness**: For all D_1, D_2 s.t. $D_1|_x \approx_c D_2|_x$, we have $C_{\mathscr{D}_1} \approx_c C_{\mathscr{D}_2}$, even if the distinguishers get access to the oracle $\mathbb{O}$.

Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

- A r-NIDI argument system for an NP language $\mathscr{L}$ with relation $\mathscr{R}_{\mathscr{L}}$ w.r.t. a “finite" oracle $\mathbb{O}$ consists of two algorithms (P, V) such that:



- **Completeness**: d is in $Supp(\mathscr{D})|_x$
- **Soundness**: If $d \neq \bot$, then $d \in \mathscr{L}$
- **Robustness**: For all D_1, D_2 s.t. $D_1|_x \approx_c D_2|_x$, we have $C_{\mathscr{D}_1} \approx_c C_{\mathscr{D}_2}$, even if the distinguishers get access to the oracle $\mathbb{O}$.

We construct r-NIDI arguments by modifying the [K’21] construction by making the underlying primitives to be secure in the presence of $\mathbb{O}$ (using complexity leveraging).

Construction of COA secure Obfuscation

Construction of COA secure Obfuscation

- Let CCACom be a (non-interactive) CCA-secure commitment scheme $(Com, Decomm)$; let $\mathbb{O}$ be an (inefficient) oracle that implements the decommitment oracle $Decomm$ for CCACom in time T.

Construction of COA secure Obfuscation

- Let CCACom be a (non-interactive) CCA-secure commitment scheme $(Com, Decomm)$; let $\mathbb{O}$ be an (inefficient) oracle that implements the decommitment oracle $Decomm$ for CCACom in time T.

- Let $\mathcal{O}$ be an (injective) obfuscation scheme for our COA fortification (secure against T-sized adversaries).

Construction of COA secure Obfuscation

- Let CCACom be a (non-interactive) CCA-secure commitment scheme $(Com, Decomm)$; let $\mathbb{O}$ be an (inefficient) oracle that implements the decommitment oracle *Decomm* for CCACom in time T.

- Let $\mathcal{O}$ be an (injective) obfuscation scheme for our COA fortification (secure against T-sized adversaries).

- Let (P,V) be an r-NIDI argument system w.r.t $\mathbb{O}$ for the following language:
  $\mathscr{L}_\phi := \{\{(O, c)\} : \exists (C, r_1, r_2) : O = \mathcal{O}(C; r_1) \wedge c = Comm(C; r_2) \wedge \phi(C) = 1\}$

Construction of COA secure Obfuscation

- Let CCACom be a (non-interactive) CCA-secure commitment scheme $(Com, Decomm)$; let $\mathbb{O}$ be an (inefficient) oracle that implements the decommitment oracle $Decomm$ for CCACom in time T.

- Let $\mathcal{O}$ be an (injective) obfuscation scheme for our COA fortification (secure against T-sized adversaries).

- Let (P,V) be an r-NIDI argument system w.r.t $\mathbb{O}$ for the following language:
$\mathcal{L}_\phi := \{\{(O, c)\} : \exists (C, r_1, r_2) : O = \mathcal{O}(C; r_1) \wedge c = Comm(C; r_2) \wedge \phi(C) = 1\}$

- Sample randomness r_1, r_2 and define the distribution $\mathcal{D}_C(\cdot)$ as:
$\mathcal{D}_C(r_1 || r_2) = \{O = \mathcal{O}(C; r_1), c = Comm(C; r_2)\}$

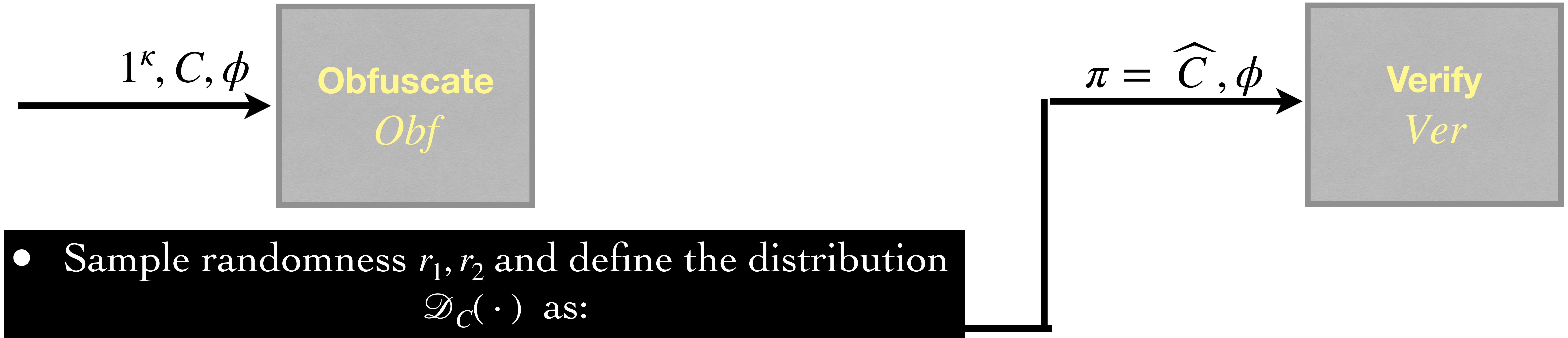
- Compute $\pi \leftarrow$ r-NIDI$.P(\mathcal{D}_C, \mathcal{L}_\phi)$ and set $\widehat{C} = \pi$

Construction of COA secure Obfuscation

- Let CCACom be a (non-interactive) CCA-secure commitment scheme $(Com, Decomm)$; let $\mathbb{O}$ be an (inefficient) oracle that implements the decommitment oracle *Decomm* for CCACom in time T.

- Let $\mathcal{O}$ be an (injective) obfuscation scheme for our COA fortification (secure against T-sized adversaries).

- Let (P,V) be an r-NIDI argument system w.r.t $\mathbb{O}$ for the following language:
$\mathscr{L}_\phi := \{\{(O, c)\} : \exists(C, r_1, r_2) : O = \mathcal{O}(C; r_1) \wedge c = Comm(C; r_2) \wedge \phi(C) = 1\}$



- Sample randomness r_1, r_2 and define the distribution $\mathscr{D}_C(\cdot)$ as:
$\mathscr{D}_C(r_1 || r_2) = \{O = \mathcal{O}(C; r_1), c = Comm(C; r_2)\}$

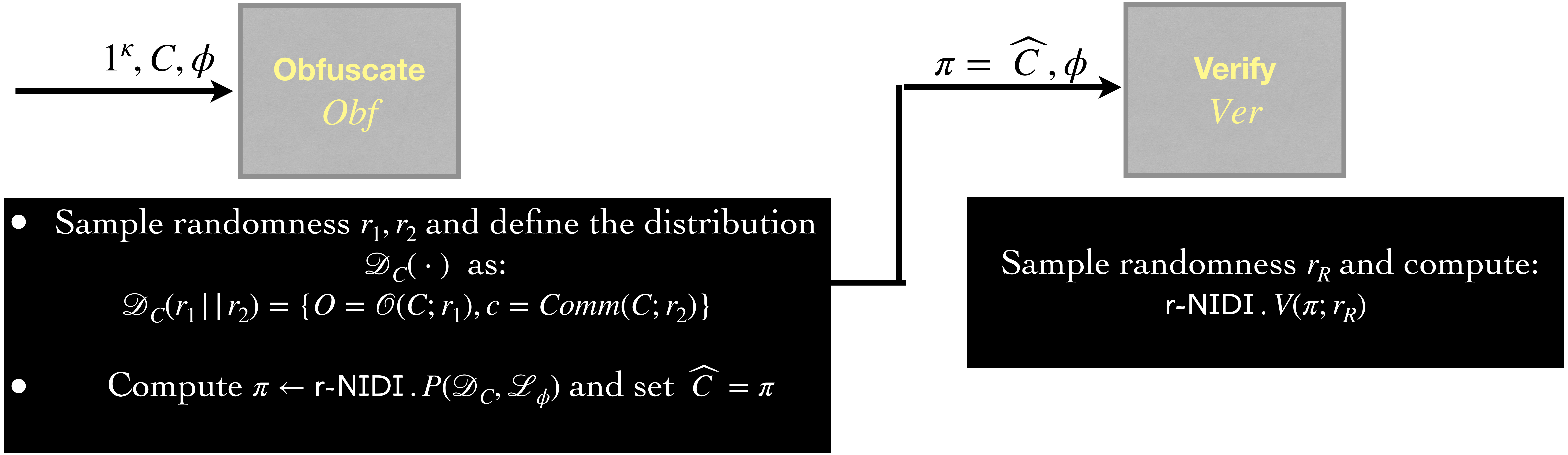
- Compute $\pi \leftarrow \text{r-NIDI}.P(\mathscr{D}_C, \mathscr{L}_\phi)$ and set $\widehat{C} = \pi$

Construction of COA secure Obfuscation

- Let CCACom be a (non-interactive) CCA-secure commitment scheme $(Com, Decomm)$; let $\mathbb{O}$ be an (inefficient) oracle that implements the decommitment oracle *Decomm* for CCACom in time T.

- Let $\mathcal{O}$ be an (injective) obfuscation scheme for our COA fortification (secure against T-sized adversaries).

- Let (P,V) be an r-NIDI argument system w.r.t $\mathbb{O}$ for the following language:
$\mathcal{L}_\phi := \{\{(O, c)\} : \exists(C, r_1, r_2) : O = \mathcal{O}(C; r_1) \wedge c = Comm(C; r_2) \wedge \phi(C) = 1\}$



Zooming into the proof of our COA construction

Zooming into the proof of our COA construction

- **Hybrid 0**: "Real" game: Given ckts $\mathbf{C_0}$ and $\mathbf{C_1}$, obfuscate $\mathbf{C_0}$ and commit to $\mathbf{C_0}$. De-obfuscation oracle is implemented using $\mathcal{O}^{-1}(\cdot)$.

Zooming into the proof of our COA construction

- **Hybrid 0**: "Real" game: Given ckts $\mathbf{C_0}$ and $\mathbf{C_1}$, obfuscate $\mathbf{C_0}$ and commit to $\mathbf{C_0}$. De-obfuscation oracle is implemented using $\mathcal{O}^{-1}(\cdot)$.

$$\mathcal{D}_{C_0}(r_1 || r_2) = \{O = \mathcal{O}(C_0; r_1), c = Comm(C_0; r_2)\}$$

$O^* = \mathcal{O}(\mathbf{C_0})$ $\quad c^* = Comm(\mathbf{C_0})$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t. } \quad C_i = \mathcal{O}^{-1}(O_i) \quad \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

Zooming into the proof of our COA construction

- **Hybrid 0**: "Real" game: Given ckts $\mathbf{C_0}$ and $\mathbf{C_1}$, obfuscate $\mathbf{C_0}$ and commit to $\mathbf{C_0}$. De-obfuscation oracle is implemented using $\mathcal{O}^{-1}(\cdot)$.

$$\mathcal{D}_{C_0}(r_1 || r_2) = \{O = \mathcal{O}(C_0; r_1), c = Comm(C_0; r_2)\}$$

$O^* = \mathcal{O}(\mathbf{C_0})$ $\quad c^* = Comm(\mathbf{C_0})$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t. } C_i = \mathcal{O}^{-1}(O_i) \quad \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

- **Hybrid 1**: De-obfuscation oracle implemented using decommitment oracle *Decomm*.

Zooming into the proof of our COA construction

- **Hybrid 0**: "Real" game: Given ckts $\mathbf{C_0}$ and $\mathbf{C_1}$, obfuscate $\mathbf{C_0}$ and commit to $\mathbf{C_0}$. De-obfuscation oracle is implemented using $\mathcal{O}^{-1}(\cdot)$.

$O^* = \mathcal{O}(\mathbf{C_0})$ $\quad$ $c^* = Comm(\mathbf{C_0})$

$$\mathcal{D}_{C_0}(r_1 || r_2) = \{O = \mathcal{O}(C_0; r_1), c = Comm(C_0; r_2)\}$$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t. } \; C_i = \mathcal{O}^{-1}(O_i) \quad \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

- **Hybrid 1**: De-obfuscation oracle implemented using decommitment oracle *Decomm*.

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t. } \; C_i = Decomm(c_i) \quad \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

Zooming into the proof of our COA construction

- **Hybrid 0**: "Real" game: Given ckts $\mathbf{C_0}$ and $\mathbf{C_1}$, obfuscate $\mathbf{C_0}$ and commit to $\mathbf{C_0}$. De-obfuscation oracle is implemented using $\mathcal{O}^{-1}(\cdot)$.

$O^* = \mathcal{O}(\mathbf{C_0})$ $\quad c^* = Comm(\mathbf{C_0})$

$$\mathcal{D}_{C_0}(r_1 || r_2) = \{O = \mathcal{O}(C_0; r_1), c = Comm(C_0; r_2)\}$$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i \quad \text{s.t.} \quad C_i = \mathcal{O}^{-1}(O_i) & \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

- **Hybrid 1**: De-obfuscation oracle implemented using decommitment oracle *Decomm*.

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i \quad \text{s.t.} \quad C_i = Decomm(c_i) & \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

Claim: Hybrid 0 $\approx_c$ Hybrid 1 => Follows from the "soundness" of r-NIDI + "perfect infectivity" of $\mathcal{O}$ + "non-malleability" of CCACom

Zooming into the proof of our COA construction

Zooming into the proof of our COA construction

- **Hybrid 2**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_1}$. De-obfuscation oracle is implemented using *Decomm*.

Zooming into the proof of our COA construction

- **Hybrid 2**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_1}$. De-obfuscation oracle is implemented using *Decomm*.

$$\mathscr{D}_{C_1}(r_1 || r_2) = \{O = \mathcal{O}(C_1; r_1), c = Comm(C_1; r_2)\}$$

$O^* = \mathcal{O}(\mathbf{C_1})$ $\quad c^* = Comm(\mathbf{C_1})$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t.} \quad C_i = Decomm(c_i) \quad \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

Zooming into the proof of our COA construction

- **Hybrid 2**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_1}$. De-obfuscation oracle is implemented using *Decomm*.

$$\mathscr{D}_{C_1}(r_1 || r_2) = \{O = \mathscr{O}(C_1; r_1), c = Comm(C_1; r_2)\}$$

$O^* = \mathscr{O}(\mathbf{C_1})$ $\quad c^* = Comm(\mathbf{C_1})$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t. } C_i = Decomm(c_i) \quad \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

Claim: Hybrid 1 $\approx_c$ Hybrid 2

Zooming into the proof of our COA construction

- **Hybrid 2**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_1}$. De-obfuscation oracle is implemented using *Decomm*.

$$\mathscr{D}_{C_1}(r_1 || r_2) = \{O = \mathcal{O}(C_1; r_1), c = Comm(C_1; r_2)\}$$

$O^* = \mathcal{O}(\mathbf{C_1})$ $\quad$ $c^* = Comm(\mathbf{C_1})$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t.} \quad C_i = Decomm(c_i) \quad \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

Claim: Hybrid 1 $\approx_c$ Hybrid 2

- **Hybrid 12**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_0}$. De-obfuscation oracle is implemented using *Decomm*.

Zooming into the proof of our COA construction

- **Hybrid 2**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_1}$. De-obfuscation oracle is implemented using *Decomm*.

$O^* = \mathcal{O}(\mathbf{C_1})$ $\quad c^* = Comm(\mathbf{C_1})$

$$\mathcal{D}_{C_1}(r_1 || r_2) = \{O = \mathcal{O}(C_1; r_1), c = Comm(C_1; r_2)\}$$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t.} \quad C_i = Decomm(c_i) \quad \text{where} \quad \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

Claim: Hybrid 1 $\approx_c$ Hybrid 2

- **Hybrid 12**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_0}$. De-obfuscation oracle is implemented using *Decomm*.

$O^* = \mathcal{O}(\mathbf{C_1})$ $\quad c^* = Comm(\mathbf{C_0})$

$$\mathcal{D}_{C_0, C_1}(r_1 || r_2) = \{O = \mathcal{O}(\mathbf{C_1}; r_1), c = Comm(\mathbf{C_0}; r_2)\}$$

Zooming into the proof of our COA construction

- **Hybrid 2**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_1}$. De-obfuscation oracle is implemented using *Decomm*.

$$\mathscr{D}_{C_1}(r_1 || r_2) = \{O = \mathcal{O}(C_1; r_1), c = Comm(C_1; r_2)\}$$

$O^* = \mathcal{O}(\mathbf{C_1})$ $\quad c^* = Comm(\mathbf{C_1})$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t. } \; C_i = Decomm(c_i) \quad \text{where } \; \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

Claim: Hybrid 1 $\approx_c$ Hybrid 2

- **Hybrid 12**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_0}$. De-obfuscation oracle is implemented using *Decomm*.

$$\mathscr{D}_{C_0, C_1}(r_1 || r_2) = \{O = \mathcal{O}(\mathbf{C_1}; r_1), c = Comm(\mathbf{C_0}; r_2)\}$$

$O^* = \mathcal{O}(\mathbf{C_1})$ $\quad c^* = Comm(\mathbf{C_0})$

Claim: Hybrid 1 $\approx_c$ Hybrid 12 => CCA-security of CCACom.

Zooming into the proof of our COA construction

- **Hybrid 2**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_1}$. De-obfuscation oracle is implemented using *Decomm*.

$$\mathscr{D}_{C_1}(r_1 || r_2) = \{O = \mathscr{O}(C_1; r_1), c = Comm(C_1; r_2)\}$$

$O^* = \mathscr{O}(\mathbf{C_1})$ $\quad$ $c^* = Comm(\mathbf{C_1})$

$$\text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{\mathbf{C}}^* = \pi^* \\ C_i & \text{s.t. } C_i = Decomm(c_i) \quad \text{where } \widetilde{C}_i = (O_i, c_i) \leftarrow Ver(\widehat{C}_i, \phi) \end{cases}$$

Claim: Hybrid 1 $\approx_c$ Hybrid 2

- **Hybrid 12**: Obfuscate $\mathbf{C_1}$ and commit to $\mathbf{C_0}$. De-obfuscation oracle is implemented using *Decomm*.

$$\mathscr{D}_{C_0, C_1}(r_1 || r_2) = \{O = \mathscr{O}(\mathbf{C_1}; r_1), c = Comm(\mathbf{C_0}; r_2)\}$$

$O^* = \mathscr{O}(\mathbf{C_1})$ $\quad$ $c^* = Comm(\mathbf{C_0})$

Claim: Hybrid 1 $\approx_c$ Hybrid 12 => CCA-security of CCACom.

Claim: Hybrid 12 $\approx_c$ Hybrid 2 => Follows from the (T, ϵ)-security of $\mathscr{O}$.

Open Problems

Open Problems

Open Problems

- Construct COA-secure obfuscation for the more traditional definition (where the verifier is deterministic)?

Open Problems

- Construct COA-secure obfuscation for the more traditional definition (where the verifier is deterministic)?
- More applications of COA-secure Obfuscation?


THANK
YOU!


?