COA-SECURE OBFUSCATION AND APPLICATIONS

Ran Canetti
Boston University

Suvradip Chakraborty
ETH Zurich

Dakshita Khurana
UIUC

Nishant Kumar*
UIUC

Manoj Prabhakaran
IIT Bombay

Oxana Poburinnaya
Boston University

EUROCRYPT 2022, Trondheim, Norway, May 31, 2022
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:**

![Icon] 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 maul 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 mauld version of a (proprietary) program B as sub-routine and A’s behaviour may depend on B.

- Obfuscation of a (puncturable) PRF $\kappa$ might allow an adversary to create an obfuscation of the same function but with key $\kappa + 1$
Defining Verifiable and non-malleable obfuscation: Challenges
Defining Verifiable and non-malleable obfuscation: Challenges

\[ 1^\kappa, C \rightarrow \text{Obfuscate} \]

\[ \text{rand} \rightarrow \hat{C} \rightarrow \text{Verify} \]

\[ \text{Acc/Rej} \]
Defining Verifiable and non-malleable obfuscation: Challenges

**Attempt 1**: Whenever \( \text{Ver} \) accepts, a specific circuit \( C \) is being obfuscated

![Diagram](image)
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. $\hat{C} \equiv C$. 
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. $\hat{C} \equiv C$.

- Consider a public predicate $\phi$, s.t. if $Ver(\hat{C}, \phi) = \text{Acc} \implies \exists C$ s.t. $\hat{C} \equiv C$ and $\phi(C) = 1$. 

![Diagram](image.png)
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 $\phi$, 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 $\phi$, 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. $\hat{C} \equiv C$.

- Consider a public predicate $\phi$, s.t. if $Ver(\hat{C}, \phi) = Acc \implies \exists C$ s.t. $\hat{C} \equiv C$ 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

- Badrinarayanan, Goyal, Jain, Sahai
  *Verifiable Functional Encryption*, Asiacrypt’16
Prior works

- Badrinarayanan, Goyal, Jain, Sahai
  *Verifiable Functional Encryption*, Asiacrypt’16

  - Does not require any trusted setup assumptions.
Prior works

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

  - Does not require any trusted setup assumptions.
  - Techniques tailor-made for $i\theta$. 
Prior works

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

  - Does not require any trusted setup assumptions.
  - Techniques tailor-made for \( iO \).
  - Limited form of Hiding: Requires a “short proof/witness” of equivalence for the functionally equivalent circuits.
Prior works

- Badrinarayanan, Goyal, Jain, Sahai
  Verifiable Functional Encryption, Asiacrypt’16
  - Does not require any trusted setup assumptions.
  - Techniques tailor-made for iO.
  - Limited form of Hiding: Requires a “short proof/witness” of equivalence for the functionally equivalent circuits.
  - No guarantees against general mauling attacks
Prior works

- Badrinarayanan, Goyal, Jain, Sahai
  Verifiable Functional Encryption, Asiacrypt’16
  - Does not require any trusted setup assumptions.
  - Techniques tailor-made for \(iO\).
  - Limited form of Hiding: Requires a “short proof/witness” of equivalence for the functionally equivalent circuits.
  - No guarantees against general mauling attacks

- Canetti, Varia
  Non-malleable Obfuscation, TCC’09
Prior works

- **Badrinarayanan, Goyal, Jain, Sahai**
  Verifiable Functional Encryption, Asiacrypt’16
  - Does not require any trusted setup assumptions.
  - Techniques tailor-made for \( io \).
  - Limited form of Hiding: Requires a “short proof/witness” of equivalence for the functionally equivalent circuits.
  - No guarantees against general mauling attacks

- **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 $\mathcal{C}$ and let $\phi$ be a predicate. Let $C \in \mathcal{C}$. 

$\sqrt{rand}$

$1^k, C, \phi$ 

Obfuscate $Obf$ 

$\widehat{C}, \phi$ 

Verify $Ver$
Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate. Let $C \in \mathcal{C}$.

Diagram:

- **Obfuscate** ($Obf$) with inputs $1^\kappa, C, \phi$
- **Verify** ($Ver$) with input $\widehat{C}, \phi$
Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate. Let $C \in \mathcal{C}$.

\begin{align*}
1^\kappa, C, \phi &\xrightarrow{\text{Rand}}{{\text{Obfuscate}}} \widetilde{C}, \phi \\
\widetilde{C}, \phi &\xrightarrow{\text{Rand}}{{\text{Verify}}} \widetilde{\mathcal{C}} \cup \{\bot\}
\end{align*}
Our notion: Security against Chosen Obfuscation Attacks (COA Obfuscation): Defn. 1

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

- **Obfuscate** ($\obfuscate{}$)
  
  - Input: $1^\kappa, C, \phi$
  
  - Output: $\widehat{C}, \phi$

- **Verify** ($\verify{}$)
  
  - Input: $\widehat{C}, \phi$
  
  - Output: $\widehat{C} \cup \{\bot\}$

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

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

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

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

**Soundness:** For any string $\tilde{C}$, if Verify outputs some $\tilde{C}$, i.e., $\tilde{C} \leftarrow \text{Ver}(\tilde{C}, \phi)$ s.t. $\tilde{C} \neq \perp \Rightarrow$ there exists an underlying circuit $C \in \mathcal{C}$ such that $\tilde{C} \equiv C$ and $\phi(C) = 1$. 
Our notion: Security against Chosen Obfuscation Attacks (COA Obfuscation): Defn. 1

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

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

- **Soundness**: For any string $\widehat{C}$, if Verify outputs some $\vec{C}$, i.e., $\vec{C} \leftarrow \text{Ver}(\widehat{C}, \phi)$ s.t. $\vec{C} \neq \bot \Rightarrow$ there exists an underlying circuit $C \in \mathcal{C}$ such that $\vec{C} \equiv C$ and $\phi(C) = 1$.

- **COA security**: For “sufficiently similar” circuits $C_0$ and $C_1$, $\text{Obf}(C_0, \phi) \approx_c \text{Obf}(C_1, \phi)$, even given access to a “de-obfuscation oracle” $\mathcal{O}(\cdot, \phi)$.
Defining COA Security: Defn. 1
Defining COA Security: Defn. 1

Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate.
Defining COA Security: Defn. 1

Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate.
Defining COA Security: Defn. 1

Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate.

$\text{Adversary}$

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

$\text{Challenger}$

$(C_0, C_1, z)$
Defining COA Security: Defn. 1

Consider a class of circuits \( \mathcal{C} \) and let \( \phi \) be a predicate.

\[
(C_0, C_1, z) \leftarrow \text{Samp}(1^\kappa)
\]

\[
(C_0, C_1, z)
\]

Samp must be “admissible \( \phi \)-satisfying “sampler:

1. \( C_0, C_1 \in \mathcal{C} \) 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 \( \mathcal{C} \) and let \( \phi \) be a predicate.

\[
(C_0, C_1, z) \leftarrow \text{Samp}(1^\kappa)
\]

Samp must be “admissible \( \phi \)-satisfying “sampler:

1. \( C_0, C_1 \in \mathcal{C} \) 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 \( \mathcal{C} \) and let \( \phi \) be a predicate.

1. \( (C_0, C_1, z) \leftarrow \text{Samp}(1^\kappa) \)

**Adversary**

2. For any black-box PPT distinguisher \( \mathcal{L} \)

\[
\begin{align*}
(C_0, C_1, z) &\leftarrow \mathcal{L}(C_b, \phi) \\
\text{Challenger} &\leftarrow \{0,1\} \\
\widehat{C} &\leftarrow \text{Obf}(C_b, \phi)
\end{align*}
\]

Samp must be "admissible \( \phi \)-satisfying "sampler:

- \( C_0, C_1 \in \mathcal{C} \) 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 $\mathcal{C}$ and let $\phi$ be a predicate.

### Adversary

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

Samp must be "admissible $\phi$-satisfying "sampler:

1. $C_0, C_1 \in \mathcal{C}$ and $\phi(C_0) = \phi(C_1) = 1$
2. $C_0 \approx_c C_1$ for any black-box PPT distinguisher

### Challenger

$$b \overset{\$}{\leftarrow} \{0,1\}$$

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

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

The De-obfuscation oracle $\emptyset(\cdot, \phi)$
Defining COA Security: Defn. 1

Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate.

**Adversary**

\[ (C_0, C_1, z) \leftarrow \text{Samp}(1^\kappa) \]

Samp must be "admissible $\phi$-satisfying "sampler:

1. $C_0, C_1 \in \mathcal{C}$ and $\phi(C_0) = \phi(C_1) = 1$
2. $C_0 \approx_c C_1$ for any black-box PPT distinguisher

**Challenger**

\[ b \overset{\$}{\leftarrow} \{0,1\} \]

\[ \widehat{C} \overset{\$}{\leftarrow} \text{Obf}(C_b, \phi) \]

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

The De-obfuscation oracle $\mathcal{O}(\cdot, \phi)$
Defining COA Security: Defn. 1

Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate.

**Adversary**

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

**Challenger**

$b \overset{$\$}{\leftarrow} \{0, 1\}$

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

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

Samp must be “admissible $\phi$-satisfying "sampler:

$C_0, C_1 \in \mathcal{C}$ 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
Defining COA Security: Defn. 1

Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate.

Adversary

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

Samp must be "admissible $\phi$-satisfying "sampler:

$C_0, C_1 \in \mathcal{C}$ 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

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

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

The De-obfuscation oracle $\mathcal{O}(\cdot, \phi)$
COA Fortification for Obfuscation: Defn. 2
Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate. Let $\mathcal{O}$ be an injective obfuscation scheme.
Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate. Let $\mathcal{O}$ be an injective obfuscation scheme.

\[\begin{align*}
1^\kappa, C, \phi &\quad \xrightarrow{\text{rand}} \quad \mathcal{O}b\text{fuscate} \quad \mathcal{O}bf \\
\tilde{C}, \phi &\quad \xrightarrow{\text{rand}} \quad \mathcal{V}er\text{ify} \quad \mathcal{V}er \quad \tilde{C} \cup \{\bot\}
\end{align*}\]
Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate. Let $\varnothing$ be an injective obfuscation scheme.

**Correctness:** For a legitimate $C$, i.e., $\phi(C) = 1$, $\widetilde{C} = \varnothing(C; r)$. 
COA Fortification for Obfuscation: Defn. 2

Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate. Let $\varnothing$ be an injective obfuscation scheme.

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

Soundness: For any string $\widehat{C}$, if Verify outputs some $\widetilde{C}$, i.e., $\widetilde{C} \leftarrow \text{Ver}(\widehat{C}, \phi)$ s.t. $\widetilde{C} \neq \bot$ $\Rightarrow$ there exists an underlying circuit $C \in \mathcal{C}$ such that $\phi(C) = 1$ and $\widetilde{C} = \varnothing(C; r)$. 
COA Fortification for Obfuscation: Defn. 2

Consider a class of circuits $\mathcal{C}$ and let $\phi$ be a predicate. Let $\varnothing$ be an injective obfuscation scheme.

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

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

$\square$ **COA security:** For “sufficiently similar” circuits $C_0$ and $C_1$, and given access to “de-obfuscation oracle” $\varnothing^{-1}(\cdot)$:

$$\text{Obf}(C_0, \phi) \approx_C \text{Obf}(C_1, \phi) \Rightarrow \varnothing(C_0) \approx_C \varnothing(C_1)$$
COA Fortification for obfuscation: Defn. 2
COA Fortification for obfuscation: Defn. 2

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

**Adversary**

$$(C_0, C_1, z) \leftarrow \text{Samp}(1^k)$$

**Challenger**

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

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

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

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

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

Samp must be an admissible $\phi$-satisfying sampler.
COA Fortification for obfuscation: Defn. 2

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

Adversary

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

Challenger

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

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

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

Samp must be an admissible $\phi$-satisfying sampler

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

1. If $\widehat{C} = 0$, output $\bot$.
2. Run $\widehat{C} \leftarrow \text{Ver}(\widehat{C}, \phi)$.
3. If $\widehat{C} \neq \bot$, return $C$ such that $\widehat{C} = \mathcal{O}(C; r)$ for some $r$.

The De-obfuscation oracle $\mathcal{O}(\cdot, \phi)$
COA Fortification for obfuscation: Defn. 2

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

**Adversary**

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

**Challenger**

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

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

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

**COA Fortification for injective PiO => Def. 1**

**Adversary**

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

**Challenger**

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

$$\mathcal{O}(C; r)$$

Samp must be an admissible $\phi$-satisfying sampler

Samp must be an admissible sampler

The De-obfuscation oracle $\mathcal{O}(\cdot, \phi)$
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 \((p_{ki}, c_i)\) to the decryption oracle, where \(p_{ki}\) can be arbitrarily related to the challenge public key \(p_{k*}\).
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.

- **Unique Decryptability:**

  - A string $pk \in \{0,1\}^{poly(k)}$ is “useless” if $\Pr[\text{Enc}(pk, m) \neq \bot] \leq \mu(k)$, for any message $m \in \mathcal{M}$
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.

- **Unique Decryptability:**
  
  - A string $pk \in \{0,1\}^{poly(\kappa)}$ is “useless” if $\Pr[\text{Enc}(pk, m) \neq \bot] \leq \mu(\kappa)$, for any message $m \in \mathcal{M}$
  
  - $\forall$ “non-useless” keys $pk$, $\exists$ “opening” $sk$ s.t. $\text{Dec}(sk, \text{Enc}(pk, m)) = m$ 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
Complete CCA-secure Public Key Encryption

Adversary

Challenger

$(pk^*, sk^*) \leftarrow \text{KeyGen}(1^n)$
Complete CCA-secure Public Key Encryption

**Adversary**

\[ \text{Challenge} \]

\[ (p_k^*, s_k^*) \leftarrow \text{KeyGen}(1^\kappa) \]

1. Check if \( p_k \) is “useless”
2. If “not useless” find opening \( s_k \) and return \( m_i := \text{Dec}(s_k, c_i) \)
Complete CCA-secure Public Key Encryption

**Adversary**

\[ (p_k^*, s_k^*) \leftarrow \text{KeyGen}(1^\kappa) \]

1. Check if \( p_k_i \) is “useless”
2. If “not useless” find opening \( s_k \) and return \( m_i := \text{Dec}(s_k, c_i) \)

**Challenger**
Complete CCA-secure Public Key Encryption

**Adversary**

\[(pk^*, sk^*) \leftarrow \text{KeyGen}(1^\kappa)\]

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

**Challenger**
Complete CCA-secure Public Key Encryption

Challenger

\[(pk^*, sk^*) \leftarrow \text{KeyGen}(1^\kappa)\]

1. Check if \(pk_i\) is “useless”
2. If “not useless” find opening \(sk\) and return
   \[m_i := \text{Dec}(sk, c_i)\]
Complete CCA-secure Public Key Encryption

\[
(pk^*, sk^*) \leftarrow \text{KeyGen}(1^\kappa)
\]

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

\[c^* \leftarrow \text{Enc}(pk^*, m_b)\]
Complete CCA-secure Public Key Encryption

Adversary

\( pk^* \)

\((pk_i, c_i)\)

\( m_i \)

\( m_0, m_1 \)

\( c^* \)

\((pk_j, c_j)\)

\( m_j \)

Challenger

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

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

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

1. Check if \( pk_j \) is “useless”
2. If “not useless” and \((pk_j, c_j) \neq (pk^*, c^*)\), find opening \( sk \) and return \( m_i := \text{Dec}(sk, c_i)\)
Complete CCA-secure Public Key Encryption

**Adversary**

1. Check if $p_k$ is “useless”
2. If “not useless” find opening $sk$ and return $m_i := \text{Dec}(sk, c_i)$

**Challenger**

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

1. $c^* \leftarrow \text{Enc}(pk^*, m_b)$

2. $c^* \leftarrow \text{Enc}(pk^*, m_b)$

1. Check if $p_k$ is “useless”
2. If “not useless” and $(pk_j, c_j) \neq (pk^*, c^*)$, find opening $sk$ and return $m_i := \text{Dec}(sk, c_i)$
Complete CCA-secure Public Key Encryption

**Adversary**

1. Check if $p_k$ is “useless”
2. If “not useless” find opening $sk$ and return $m_i := \text{Dec}(sk, c_i)$

**Challenger**

1. (pk*, sk*) ← KeyGen(1^σ)
2. $c* ← \text{Enc}(pk*, m_b)$
3. $c* = \text{c*}
4. Check if $p_k$ is “useless”
5. If “not useless” and $(pk, c) \neq (pk*, c*)$, find opening $sk$ and return $m_i := \text{Dec}(sk, c_i)$
Constructing CCCA-secure PKE
Constructing CCCA-secure PKE

1. Let $c\theta = (Obf, Ver)$ be COA-fortification of injective $i\theta$ w.r.t predicate $\phi$: $\phi(C)$ attest that $C$ is of the form of $P$. 
Constructing CCCA-secure PKE

1. Let $cO = (Obf, Ver)$ be COA-fortification of injective $iO$ w.r.t predicate $\phi$: $\phi(C)$ 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 $\phi$: $\phi(C)$ 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$^\kappa$)**: 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 $c\mathcal{O} = (\text{Obf}, \text{Ver})$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate $\phi$: $\phi(C)$ 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.

KeyGen$(1^\kappa)$: Sample keys $K_1$ and $K_2$ for $F_1$ and $F_2$ resp., output $pk = \hat{P} \leftarrow \text{Obf}(P, \phi)$, 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 $\phi$: $\phi(C)$ attest that $C$ is of the form of $P$.

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

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

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

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 $c\mathcal{O} = (\text{Obf}, \text{Ver})$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate $\phi$: $\phi(C)$ 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($1^\kappa$)**: Sample keys $K_1$ and $K_2$ for $F_1$ and $F_2$ resp., output $pk = \hat{P} \leftarrow \text{Obf}(P, \phi)$, and $sk = (K_1, K_2)$.

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

- **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 $\phi$: $\phi(C)$ attest that $C$ is of the form of $P$.

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

   - **KeyGen($1^k$):** Sample keys $K_1$ and $K_2$ for $F_1$ and $F_2$ resp., output $pk = \hat{P} \leftarrow Obf(P, \phi)$, and $sk = (K_1, K_2)$.
   
   - **Enc($pk, m$):** Run $\hat{P} \leftarrow Ver(\hat{P}, \phi)$; sample random $r$ and output $\tilde{P}(m, r)$.

   - **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. $c_3 = F_2(K_2, c_1 \mid c_2)$
   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 $c\mathcal{O} = (\text{Obf}, \text{Ver})$ be COA-fortification of injective $i\mathcal{O}$ w.r.t predicate $\phi$: $\phi(C)$ attest that $C$ is of the form of $P$.

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

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

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

- **Dec**(sk = (K1, K2), c = (c1, c2, c3)): Verify authenticity of $c_3$ and recover $m$.

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

- If $\widehat{P} \neq \bot \implies \exists P' = P'_{K_1,K_2}$ s.t. $\widehat{P} = i\mathcal{O}(P'; r)$ and hence can recover $P'$ using $i\mathcal{O}^{-1}(\widehat{P})$. Use $(K_1', K_2')$ to decrypt.
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. $c_3 = F_2(K_2, c_1 \mid \mid c_2)$
   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.

Program \(P_{K_1, K_2}\)

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 \(c = (c_1, c_2, c_3)\)
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 \(pk = \hat{P}\):

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 \(pk = \hat{P}\):

   Let \(\hat{P} \leftarrow \text{Ver}(\hat{P}, \phi)\),

\[
\begin{align*}
1. \text{Constants: } K_1, K_2 \\
2. \text{Input: } m, r \\
a. c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m \\
b. c_3 = F_2(K_2, c_1 || c_2) \\
c. \text{Output } c = (c_1, c_2, c_3)
\end{align*}
\]

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 \(pk_1 = pk^*\): 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 \(pk = \hat{P}\):

   - Let \(\tilde{P} \leftarrow \text{Ver}(\hat{P}, \phi)\),
   - If \(\tilde{P} \neq \bot \implies \exists P' = P'_{K_1, K_2}\) such that \(\tilde{P} \leftarrow i\delta(1^\kappa, P')\).
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 \(p_{k_i} = pk^*\): 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 \(\widehat{P} \leftarrow Ver(\widehat{P}, \phi)\),
- If \(\widehat{P} \neq \perp \implies \exists P^\prime = P_{K_1', K_2'}\) such that \(\widehat{P} \leftarrow iO(1^k, P^\prime)\).
- Recover \(P^\prime\) using \(iO^{-1}(\widehat{P})\), use \(sk' = (K_1', K_2')\) to decrypt \(c\). (follows from the COA fortification of injective \(iO\) w.r.t. predicate \(\phi\)).

Program \(P_{K_1, K_2}\)

\[1. \text{Constants: } K_1, K_2\]
\[2. \text{Input: } m, r\]
\[a. \ c_1 = G(r); c_2 = F_1(K_1, c_1) \oplus m\]
\[b. \ c_3 = F_2(K_2, c_1 || c_2)\]
\[c. \text{Output } c = (c_1, c_2, c_3)\]
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 \(pk = \widetilde{P}\):

\[\begin{align*}
&\text{Let } \widetilde{P} \leftarrow \text{Ver}(\widetilde{P}, \phi), \\
&\text{If } \widetilde{P} \neq \bot \implies \exists P' = P'_{K_1, K_2} \text{ such that } \widetilde{P} \leftarrow i\theta(1^k, P').
\end{align*}\]

\(\star\) Recover \(P'\) using \(i\theta^{-1}(\widetilde{P})\), use \(sk' = (K_1', K_2')\) to decrypt \(c\). (follows from the COA fortification of injective \(i\theta\) w.r.t. predicate \(\phi\)).
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 $\mathcal{L}$ with relation $R_{\mathcal{L}}$ consists of two algorithms $(P, V)$ such that:
Non-Interactive Distributional Indistinguishable (NIDI) argument [K’21]

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

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

Prover $P$  

Verifier $V$  

Sampler $\mathcal{C}_\mathcal{D}$  

Input $\mathcal{L}$

d $\leftarrow V(\mathcal{C}_\mathcal{D}; r_V)$
Non-Interactive Distributional Indistinguishable (NIDI) argument [K'21]

- A NIDI argument system for an NP language $\mathcal{L}$ with relation $\mathcal{R}_{\mathcal{L}}$ consists of two algorithms $(P, V)$ such that:
  
  Completeness: $d$ is in $\text{Supp}(\mathcal{D})|_x$

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

- Prover $P$

- Verifier $V$

- Sampler $C_{\mathcal{D}}$

  \[ d \leftarrow V(C_{\mathcal{D}}; r_V) \]
Non-Interactive Distributional Indistinguishable (NIDI) argument [K’21]

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

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

![Diagram of NIDI argument system]

Inputs: Language $\mathcal{L}$, Distribution $\mathcal{D}$ that samples $(x, w) \in \mathcal{R}_\mathbf{x}$
Non-Interactive Distributional Indistinguishable (NIDI) argument [K’21]

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

  - **Completeness:** $d$ is in $\text{Supp}(\mathcal{D})|_x$
  
  - **Soundness:** If $d \neq \bot$, then $d \in \mathcal{L}$
  
  - **Privacy:** For all $D_1, D_2$ s.t. $D_1|_x \approx_c D_2|_x$, we have $C_{\mathcal{D}_1} \approx_c C_{\mathcal{D}_2}$
Non-Interactive Distributional Indistinguishable (NIDI) argument [K’21]

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

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

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

  - **Privacy**: For all $D_1, D_2$ s.t. $D_1|_x \approx_c D_2|_x$, we have $C_{\mathcal{D}_1} \approx_c C_{\mathcal{D}_2}$

**Theorem**: Assuming sub-exp $i\mathcal{O}$ 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 $\mathcal{L}$ with relation $\mathcal{R}_{\mathcal{L}}$ w.r.t. a "finite" oracle $\mathcal{O}$ 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 $\mathcal{L}$ with relation $R_{\mathcal{L}}$ w.r.t. a "finite" oracle $\mathcal{O}$ 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 $\mathcal{L}$ with relation $\mathcal{R}_\mathcal{D}$ w.r.t. a "finite" oracle $\mathcal{O}$ consists of two algorithms $(P, V)$ such that:

**Inputs:** Language $\mathcal{L}$, Distribution $\mathcal{D}$ that samples $(x, w) \in \mathcal{R}_\mathcal{D}$
Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

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

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

Prover \( P \)  
Verifier \( V \)  
Sampler \( C_\mathcal{D} \)
**Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument**

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

  - **Prover $P$**
    - Inputs: Language $\mathcal{L}$, Distribution $\mathcal{D}$ that samples $(x, w) \in \mathcal{R}_\mathcal{L}$

  - **Verifier $V$**
    - Input $\mathcal{L}$
    - $d \leftarrow V(C_\mathcal{D}; r_V)$
A r-NIDI argument system for an NP language \( \mathcal{L} \) with relation \( \mathcal{R}_\mathcal{L} \) w.r.t. a "finite" oracle \( \mathcal{O} \) consists of two algorithms \((P, V)\) such that:

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

Diagram:

- **Prover** \( P \):
  - Inputs: Language \( \mathcal{L} \), Distribution \( \mathcal{D} \) that samples \((x, w)\) \( \in \mathcal{R}_\mathcal{L} \)

- **Verifier** \( V \):
  - Input \( \mathcal{L} \)
  - \( d \leftarrow V(C_\mathcal{D}; r_V) \)
Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

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

- **Completeness**: if \( d \in \text{Supp}(\mathcal{D}) \), then \( d \in \mathcal{L} \)
- **Soundness**: if \( d \neq \perp \), then \( d \in \mathcal{L} \)
Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

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

  - **Completeness:** \( d \) is in \( \text{Supp}(\mathcal{D})|_x \)
  - **Soundness:** If \( d \neq \bot \), then \( d \in \mathcal{L} \)
  - **Robustness:** For all \( D_1, D_2 \) s.t. \( D_1|_x \approx_c D_2|_x \), we have \( C_{\mathcal{D}_1} \approx_c C_{\mathcal{D}_2} \), even if the distinguishers get access to the oracle \( \mathcal{O} \).
Robust Non-Interactive Distributional Indistinguishable (r-NIDI) argument

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

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

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

- Let CCACom be a (non-interactive) CCA-secure commitment scheme \((\text{Com}, \text{Decom})\); let \(O\) be an (inefficient) oracle that implements the decommitment oracle \(\text{Decom}\) for CCACom in time \(T\).
Construction of COA secure Obfuscation

- Let CCACom be a (non-interactive) CCA-secure commitment scheme \((Com, Decomm)\); let \(\mathcal{O}\) 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 \((Com, Decom)\); let \(\mathcal{O}\) be an (inefficient) oracle that implements the decommitment oracle \(Decom\) 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 \(\mathcal{O}\) for the following language:

\[
\mathcal{L}_\phi := \{(O,c) : \exists (C,r_1,r_2) : O = \mathcal{O}(C;r_1) \land c = \text{Comm}(C;r_2) \land \phi(C) = 1\}
\]
Construction of COA secure Obfuscation

- Let CCACom be a (non-interactive) CCA-secure commitment scheme \((\text{Com}, \text{Decomm})\); let \(\mathcal{O}\) be an (inefficient) oracle that implements the decommitment oracle \(\text{Decomm}\) 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 \(\mathcal{O}\) for the following language:
  \[
  \mathcal{L}_\phi := \{(O, c) : \exists (C, r_1, r_2) : O = \mathcal{O}(C; r_1) \land c = \text{Comm}(C; r_2) \land \phi(C) = 1\}
  \]

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

- Compute \(\pi \leftarrow \text{r-NIDI} \cdot P(\mathcal{D}_C, \mathcal{L}_\phi)\) and set \(\hat{C} = \pi\)
Construction of COA secure Obfuscation

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

- Let \(\varnothing\) 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 \(\mathcal{O}\) for the following language:

\[
\mathcal{L}_\phi := \{(O, c) : \exists (C, r_1, r_2) : O = \mathcal{O}(C; r_1) \land c = \text{Comm}(C; r_2) \land \phi(C) = 1\}
\]

- Sample randomness \(r_1, r_2\) and define the distribution \(\mathcal{D}_C(\cdot)\) as:

\[
\mathcal{D}_C(r_1 || r_2) = \{O = \mathcal{O}(C; r_1), c = \text{Comm}(C; r_2)\}
\]

- Compute \(\pi \leftarrow \text{r-NIDI}.P(\mathcal{D}_C, \mathcal{L}_\phi)\) 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 \(\mathbb{O}\) 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:

\[
\mathcal{L}_\phi := \{((O, c)) : \exists (C, r_1, r_2) : O = \mathbb{O}(C; r_1) \land c = Comm(C; r_2) \land \phi(C) = 1\}
\]

1\(^x\), \(C\), \(\phi\)  \[\text{Obfuscate}\]  \(\text{Obf}\)

\[\pi = \widehat{C}, \phi\]  \[\text{Verify}\]  \(\text{Ver}\)

- Sample randomness \(r_1, r_2\) and define the distribution \(\mathcal{D}_C(\cdot)\) as:

\[
\mathcal{D}_C(r_1 || r_2) = \{O = \mathbb{O}(C; r_1), c = Comm(C; r_2)\}
\]

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

Sample randomness \(r_R\) and compute:
\(r\)-NIDI. \(V(\pi; r_R)\)
Zooming into the proof of our COA construction
Zooming into the proof of our COA construction

- **Hybrid 0:** “Real” game: Given ckts $C_0$ and $C_1$, obfuscate $C_0$ and commit to $C_0$. De-obfuscation oracle is implemented using $\sigma^{-1}(\cdot)$. 
Zooming into the proof of our COA construction

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

\[ O^* = \mathcal{O}(C_0) \quad c^* = \text{Comm}(C_0) \]

\[
\mathcal{D}_{C_0}(r_1 \mid r_2) = \{ O = \mathcal{O}(C_0; r_1), c = \text{Comm}(C_0; r_2) \}
\]

\[
\text{DeObf}(\widehat{C}_i) = \begin{cases} 
\bot & \text{if } \widehat{C}_i = \widehat{C}^* = \pi^* \\
C_i & \text{s.t. } C_i = \mathcal{O}^{-1}(O_i) \quad \text{where } \overline{C}_i = (O_i, c_i) \leftarrow \text{Ver}(\widehat{C}_i, \phi)
\end{cases}
\]
Zooming into the proof of our COA construction

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

\[
O^* = \delta(C_0) \quad c^* = \text{Comm}(C_0)
\]

\[
\mathcal{D}_{C_0}(r_1 \| r_2) = \{ O = \delta(C_0; r_1), c = \text{Comm}(C_0; r_2) \}
\]

\[
\text{DeObf}(\widehat{C}_i) = \begin{cases} 
\bot & \text{if } \widehat{C}_i = \widehat{C}^* = \pi^* \\
C_i & \text{s.t. } C_i = \delta^{-1}(O_i) \quad \text{where } \overline{C}_i = (O_i, c_i) \leftarrow \text{Ver}(\widehat{C}_i, \phi)
\end{cases}
\]

- **Hybrid 1**: De-obfuscation oracle implemented using decommitment oracle $\text{Decomm}$. 


Zooming into the proof of our COA construction

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

  $$O^* = \phi(C_0) \quad c^* = \text{Comm}(C_0)$$

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

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

- **Hybrid 1**: De-obfuscation oracle implemented using decommitment oracle $\text{Decomm}$.

  $$\text{DeObf}(\widehat{C}^*_i) = \begin{cases} 
  \perp & \text{if } \widehat{C}_i = \widehat{C}^* = \pi^* \\
  C_i & \text{s.t. } C_i = \text{Decomm}(c_i) \text{ where } \overline{C}_i = (O_i, c_i) \leftarrow \text{Ver}(\widehat{C}_i, \phi)
  \end{cases}$$
Zooming into the proof of our COA construction

- **Hybrid 0**: “Real” game: Given ckts $C_O$ and $C_I$, obfuscate $C_O$ and commit to $C_O$. De-obfuscation oracle is implemented using $\sigma^{-1}(\cdot)$.

$$O^* = \sigma(C_O) \quad c^* = \text{Comm}(C_O)$$

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

- **Hybrid 1**: De-obfuscation oracle implemented using decommitment oracle $\text{Decomm}$.

$$\text{Deobf}(\widehat{C}_i) = \begin{cases} \bot & \text{if } \widehat{C}_i = \widehat{C}^* = \pi^* \\ C_i & \text{s.t. } C_i = \sigma^{-1}(O_i) \quad \text{where } \overline{C}_i = (O_i, c_i) \leftarrow \text{Ver}(\widehat{C}_i, \phi) \end{cases}$$

Claim: Hybrid $0 \approx_c$ Hybrid $1$ => Follows from the “soundness” of r-NIDI + “perfect infectivity” of $\sigma$ + “non-malleability” of CCACOM
Zooming into the proof of our COA construction
Zooming into the proof of our COA construction

• **Hybrid 2**: Obfuscate $C_1$ and commit to $C_1$. De-obfuscation oracle is implemented using $Dcomm$. 
Zooming into the proof of our COA construction

- Hybrid 2: Obfuscate $C_1$ and commit to $C_1$. De-obfuscation oracle is implemented using $Decomm$. 

\[
O^* = \mathcal{O}(C_1) \quad c^* = \text{Comm}(C_1)
\]

\[
\mathcal{D}_{C_1}(r_1 \mid r_2) = \{ O = \mathcal{O}(C_1; r_1), c = \text{Comm}(C_1; r_2) \}
\]

\[
\text{DeObf}(\widehat{C}_i) = \begin{cases} 
\bot & \text{if } \widehat{C}_i = \widehat{C}^* = \pi^* \\
C_i & \text{s.t. } C_i = \text{Decomm}(c_i) \quad \text{where} \quad \overline{C}_i = (O_i, c_i) \leftarrow \text{Ver}(\widehat{C}_i, \phi)
\end{cases}
\]
Zooming into the proof of our COA construction

- **Hybrid 2**: Obfuscate $C_1$ and commit to $C_1$. De-obfuscation oracle is implemented using $Deomm$.

\[
\mathcal{D}_{C_1}(r_1 \mid r_2) = \{ O = \mathcal{O}(C_1; r_1), c = Comm(C_1; r_2) \}
\]

\[
O^* = \mathcal{O}(C_1) \quad c^* = Comm(C_1)
\]

\[
\text{DeObf}(\widehat{C}_i) = \begin{cases} 
\bot & \text{if } \widehat{C}_i = \widehat{C}^* = \pi^* \\
C_i & \text{s.t. } C_i = Decomm(c_i) \text{ where } \widehat{C}_i = (O_i, c_i) \leftarrow \text{Ver}(\widehat{C}_i, \phi) 
\end{cases}
\]

**Claim**: Hybrid 1 $\approx_c$ Hybrid 2
Zooming into the proof of our COA construction

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

\[
D_{C_1}(r_1 | r_2) = \{ O = \mathcal{O}(C_1; r_1), c = \text{Comm}(C_1; r_2) \}
\]

\[
O^* = \mathcal{O}(C_1) \quad c^* = \text{Comm}(C_1)
\]

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

**Claim**: Hybrid 1 $\approx_c$ Hybrid 2

- **Hybrid 12**: Obfuscate $C_1$ and commit to $C_0$. De-obfuscation oracle is implemented using $Decomm$. 
Zooming into the proof of our COA construction

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

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

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

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

  **Claim**: Hybrid 1 $\approx_c$ Hybrid 2

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

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

  $$\mathcal{D}_{C_0, C_1}(r_1 | r_2) = \{ O = \mathcal{O}(C_1; r_1), c = \text{Comm}(C_0; r_2) \}$$
Zooming into the proof of our COA construction

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

\[
\begin{align*}
D_{C_1}(r_1 || r_2) &= \{ O = \mathcal{O}(C_1; r_1), c = \text{Comm}(C_1; r_2) \}
\end{align*}
\]

\[
\begin{align*}
O^* &= \mathcal{O}(C_1) \quad c^* = \text{Comm}(C_1)
\end{align*}
\]

\[
\text{DeObf}(\widehat{C}_i) = \begin{cases} 
\bot & \text{if } \widehat{C}_i = \widehat{C}^* = \pi^* \\
C_i & \text{s.t. } C_i = \text{Decomm}(c_i) \text{ where } \widehat{C}_i = (O_i, c_i) \leftarrow \text{Ver}(\widehat{C}_i, \phi)
\end{cases}
\]

**Claim**: Hybrid 1 $\approx_c$ Hybrid 2

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

\[
\begin{align*}
D_{C_0, C_1}(r_1 || r_2) &= \{ O = \mathcal{O}(C_1; r_1), c = \text{Comm}(C_0; r_2) \}
\end{align*}
\]

\[
\begin{align*}
O^* &= \mathcal{O}(C_1) \quad c^* = \text{Comm}(C_0)
\end{align*}
\]

**Claim**: Hybrid 1 $\approx_c$ Hybrid 12 $\Rightarrow$ CCA-security of CCACom.
Zooming into the proof of our COA construction

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

  \[ O^* = \mathcal{O}(C_1) \quad c^* = Comm(C_1) \]

  \[ \mathcal{D}_{C_1}(r_1 \parallel r_2) = \{ O = \mathcal{O}(C_1; r_1), c = Comm(C_1; r_2) \} \]

  \[ \text{DeObf}(\widehat{C}_i) = \begin{cases} \perp & \text{if } \widehat{C}_i = \widehat{C}^* = \pi^* \\ C_i & \text{s.t. } C_i = Decom(C_i) \text{ where } \widehat{C}_i = (O, c_i) \leftarrow \text{Ver}(\widehat{C}_i, \phi) \end{cases} \]

  **Claim**: Hybrid 1 $\approx_c$ Hybrid 2

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

  \[ O^* = \mathcal{O}(C_1) \quad c^* = Comm(C_0) \]

  \[ \mathcal{D}_{C_0, C_1}(r_1 \parallel r_2) = \{ O = \mathcal{O}(C_1; r_1), c = Comm(C_0; r_2) \} \]

  **Claim**: Hybrid 1 $\approx_c$ Hybrid 12 $\Rightarrow$ CCA-security of CCACom.

  **Claim**: Hybrid 12 $\approx_c$ Hybrid 2 $\Rightarrow$ Follows from the $(T, \epsilon)$-security of $\mathcal{O}$. 
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!

❓