Indistinguishability Obfuscation via Mathematical Proofs of Equivalence

Abhishek Jain  
Johns Hopkins University

Zhengzhong Jin  
MIT
Indistinguishability Obfuscation (iO)

Circuit/Turing Machine

1 function main() {
2     console.log('hello, world');
3 }
4 main();

C

(Circuit/Turing Machine)

iO

C’
Indistinguishability Obfuscation (iO)

**Preserve Functionality:** \( \forall x, C'(x) = C(x) \)

```
function main() {
    console.log('hello, world');
    main();
}
```
Indistinguishability Security
Indistinguishability Security

For any $C_0, C_1$ if $\forall x \ C_0(x) = C_1(x)$

$$iO(1^\lambda, C_0) \approx_c iO(1^\lambda, C_1) \quad (\lambda : \text{Security Parameter})$$
Indistinguishability Security

For any $C_0, C_1$ if $\forall x \ C_0(x) = C_1(x)$

$$iO(1^\lambda, C_0) \approx_c iO(1^\lambda, C_1) \quad (\lambda : \text{Security Parameter})$$

Non-falsifiability

$C_0, C_1$ $\iff$ $iO(C_b)$ $\iff$ $b'$
Indistinguishability Security

For any $C_0, C_1$ if $\forall x \ C_0(x) = C_1(x)$

\[iO(1^\lambda, C_0) \approx_c iO(1^\lambda, C_1)\]  

($\lambda$ : Security Parameter)

Non-falsifiability

\[C_0, C_1\]
\[\downarrow\]
\[iO(C_b)\]
\[\leftarrow\]
\[b'\]
\[\downarrow\]
\[b \leftarrow \{0,1\}\]
Indistinguishability Security

For any $C_0, C_1$ if $\forall x \ C_0(x) = C_1(x)$

$$iO(1^\lambda, C_0) \approx_c iO(1^\lambda, C_1)$$  ($\lambda$ : Security Parameter)

Non-falsifiability

$C_0, C_1$  

$iO(C_b)$  

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

Check $\forall x \ C_0(x) = C_1(x) \land b = b'$ inefficiently
Can we build iO?
Can we build iO?

• A long line of works:
  [Garg-Gentry-Halevi-Raykova-Sahai-Waters’13][Pass-Seth-Telang’14]
  [Gentry-Lewko-Sahai-Waters’15][Ananth-Jain’15][Bitansky-Vaikuntanathan’15]
  [Lin’16][Lin-Vaikuntanathan’16][Lin-Pass-Karn Seth-Telang’16]
  [Garg-Miles-Mukherjee-Sahai-Srinivasan-Zhandry’16][Ananth-Sahai’17][Lin’17]
  [Lin-Tessaro’17][Agrawal’19][Jain-Lin-Matt-Sahai’19][Brakerski-Dottling-Malavolta’20]...
Can we build iO?

• A long line of works:
  [Garg-Gentry-Halevi-Raykova-Sahai-Waters’13][Pass-Seth-Telang’14]
  [Gentry-Lewko-Sahai-Waters’15][Ananth-Jain’15][Bitansky-Vaikuntanathan’15]
  [Lin’16][Lin-Vaikuntanathan’16][Lin-Pass-Karn Seth-Telang’16]
  [Garg-Miles-Mukherjee-Sahai-Srinivasan-Zhandry’16][Ananth-Sahai’17][Lin’17]
  [Lin-Tessaro’17][Agrawal’19][Jain-Lin-Matt-Sahai’19][Brakerski-Dottling-Malavolta’20]...

• **iO from Well-Founded Assumptions** [Jain-Lin-Sahai’20]
  Based on **Sub-exponential Security** of Learning with Errors, and Learning Parity with Noise and more...
$2| input |$ - Loss in Reduction

Adv. for iO
$2^{|\text{input}|}$-Loss in Reduction

Adv. for iO

2$^{|\text{input}|}$-time Reduction

Break

Assumption $P$
$2^{|\text{input}|}$-Loss in Reduction

Assume $2^{\lambda_c}$ -Security of $P$ & set $2^{\lambda_c} > 2^{|\text{input}|}$
$2^{|\text{input}|}$-Loss in Reduction

Adv. for $iO$

2$^{|\text{input}|}$-time Reduction

Break

Assumption $P$

Assume $2^{\lambda^c}$-Security of $P$ & set $2^{\lambda^c} > 2^{|\text{input}|}$

$|\text{input}| < \lambda^c$
2|input|-Security Loss is Bad
$2|\text{input}|$-Security Loss is Bad

$M$ (Turing Machine) \hspace{2cm} \text{iO for Turing Machines} \hspace{2cm} M'$ (Turing Machine)
2$|\text{input}|$-Security Loss is Bad

Ideally: $M'$ works for unbounded input-length
$2^{|\text{input}|}$-Security Loss is Bad

Ideally: $M'$ works for unbounded input-length

Prior work:
Input length of $M'$ is bounded (since $|\text{input}| < \lambda^c$)

iO: the “Central Hub” [Sahai-Waters’13]

Nash Equilibrium [Bitansky-Paneth-Rosen’14]

NIZKs/SNARGs

Deniable Encryption

Witness Encryption

Software watermarking

...
$2^{|input|}$-Security Loss “Spreads”

- Nash Equilibrium
- Witness Encryption
- NIZKs/SNARGs
- Deniable Encryption
- Software watermarking

...
2|input| - Security Loss “Spreads”

- Nash Equilibrium
- Witness Encryption
- NIZKs/SNARGs
- Large CRS
- Deniable Encryption
- Software watermarking

...
2^{\text{input}}$-Security Loss “Spreads”

- Nash Equilibrium
- Witness Encryption
- Large CRS
- NIZKs/SNARGs
- Deniable Encryption
- Software watermarking
- Large ciphertext
Question: Can we build iO with a security loss independent of the input length?
Is $2^{\text{input}}$-Loss Inherent? (folklore)
Is $2^{|\text{input}|}$-Loss Inherent? (folklore)

Adv. for iO

Assumption $P$

$(C_0 \equiv C_1)$
Is $2^{|\text{input}|}$-Loss Inherent? (folklore)

\[ (C_0 \equiv C_1) \]

Adv. for iO

Break

Assumption $P$
Is $2^{|\text{input}|}$-Loss Inherent? (folklore)
Is $2^{|\text{input}|}$-Loss Inherent? (folklore)

Assumption $P$

$C_0^*(x^*) \neq C_1(x^*)$

"inadmissible"

Adv. for iO
Is $2^{|\text{input}|}$-Loss Inherent? (folklore)

C^*_0, C^*_1

"inadmissible"
Adv. for iO

$C^*_0(x^*) \neq C^*_1(x^*)$

Not Break

Assumption $P$
Is $2^{|\text{input}|}$-Loss Inherent? (folklore)

"inadmissible"  
Adv. for iO

$C_0^*(x^*) \neq C_1(x^*)$

Reduction needs to check $C_0^* \equiv C_1^*$

Assumption $P$
Is $2^{|\text{input}|}$-Loss Inherent? (folklore)

```
C_0^*(x^*) \neq C_1(x^*)
```

Not Break

```
C_0^* \equiv C_1^*
```

Reduction needs to check $C_0^* \equiv C_1^*$

"inadmissible"
Adv. for iO

Assumption $P$
Broader Perspective
Broader Perspective

Non-Falsifiable definition appears in many other places in crypto
Broader Perspective

**Non-Falsifiable** definition appears in many other places in crypto

**Example:** Soundness of Proof Systems

(for $L \in NP$)
Broader Perspective

**Non-Falsifiable** definition appears in many other places in crypto

**Example:** Soundness of Proof Systems

(for $L \in NP$)

If $x \notin L$, any cheating proof should be rejected
Broader Perspective

Non-Falsifiable definition appears in many other places in crypto

**Example**: Soundness of Proof Systems  
(for $L \in NP$)  
If $x \notin L$, any cheating proof should be rejected  
Non-Falsifiable
Broader Perspective

Non-Falsifiable definition appears in many other places in crypto

**Example:** Soundness of Proof Systems

(for $L \in NP$)

If $x \notin L$, any cheating proof should be rejected

Non-Falsifiable

[Gentry-Wichs’10] impossibility for SNARGs
Reduction checks $C_0(x^*) = C_1(x^*)$ for every $x^*$ with $2|x^*|\text{-loss}$
Reduction checks $C_0(x^*) = C_1(x^*)$ for every $x^*$ with $2|x^*|$-loss

Previous Works:

“$\forall x \ C_0(x) = C_1(x)$” can be decided in P

[Garg-Pandey-Srinivasan’16, Garg-Srinivasan’16, Garg-Pandey-Srinivasan-Zhandry’17]

[Liu-Zhandry’17]
Reduction checks $C_0(x^*) = C_1(x^*)$ for every $x^*$ with $2|x^*|$-loss

Previous Works:

“$\forall x \ C_0(x) = C_1(x)$” can be decided in $\mathbf{P}$

[Garg-Pandey-Srinivasan’16, Garg-Srinivasan’16, Garg-Pandey-Srinivasan-Zhandry’17]

[Liu-Zhandry’17]

This Work:

Leverage math. proofs of “$\forall x \ C_0(x) = C_1(x)$” to avoid the $2|x|$-loss
Why Math. Proofs Exist?

Recall: when iO is used in the security proof

... 

• Construct $C_0$, $C_1$
• **Write a math. proof** for $\forall x \ C_0(x) = C_1(x)$
• Apply iO security to derive $iO(C_0) \approx_c iO(C_1)$

... 

The proof must be “short” (length $\ll 2^{|x|}$) 
Otherwise, we (human brain) can’t understand it.
Our Results I (for Propositional Logic)

$iO$ with security loss independent of $|input|$ for any ckts $\{C_n^1\}_n, \{C_n^2\}_n$ where $C_n^1(x) \leftrightarrow C_n^2(x)$ have poly-size proofs in Extended Frege systems.
Our Results I (for Propositional Logic)

\(iO\) with security loss independent of \(|input|\) for any ckts \(\{C^1_n\}_n, \{C^2_n\}_n\) where \(C^1_n(x) \leftrightarrow C^2_n(x)\) have poly-size proofs in Extended Frege systems.

(Assumptions: \(2^{p(\lambda)}\)-secure LWE, OWF, iO for circuits of size independent of \(|input|\).)
Extended Frege System ($\mathcal{EF}$)

Variables represent True/False

• Axioms:
  
  \[
  (p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r)) \\
  p \rightarrow (q \rightarrow p) \\
  p \rightarrow \neg\neg p
  \]

• Inference Rule:
  
  \[
  p, p \rightarrow q \vdash q
  \]

• Extension Rule:
  
  \[e \leftrightarrow \phi\]
  
  (assign a new variable $e$ to a formula $\phi$)

What theorems have poly-size $\mathcal{EF}$-proofs?
Cook’s Theory $PV$ [Cook’75]
Cook’s Theory $PV$ [Cook’75]

Poly-size Proofs in $\mathcal{EF}$

Propositional Translation

Theory $PV$
Cook’s Theory $PV$ [Cook’75]

Poly-size Proofs in $\mathcal{EF}$

Propositional Translation

Theory $PV$

Variables represent *natural numbers*
Cook’s Theory $PV$ [Cook’75]

Poly-size Proofs in $\mathcal{EF}$

Propositional Translation

Theory $PV$

Variables represent *natural numbers*

Allow definition of *any* polynomial-time functions, e.g.
- Arithmetic: $+, -, \times, \div, \leq, <, \lfloor \cdot \rfloor, \text{mod}, ...$
- Logic Symbols: $\rightarrow, \neg, \land, ...$
Our Results II (for Cook’s Theory PV)

$iO$ for any unbounded-input Turing machines $M_1, M_2$, with $\vdash_{PV} M_1(x) = M_2(x)$.

Assumptions: sub-exponential security of LWE & iO for circuits.
What Theorems Can $PV$ Prove?
What Theorems Can $PV$ Prove?

Prior work:
What Theorems Can $PV$ Prove?

Prior work:
• Correctness of “natural” algorithms in P
What Theorems Can $PV$ Prove?

Prior work:
- Correctness of “natural” algorithms in P
- Basic Linear Algebra
What Theorems Can $PV$ Prove?

Prior work:
• Correctness of “natural” algorithms in $P$
• Basic Linear Algebra
• Combinatorial Theorems
What Theorems Can $PV$ Prove?

Prior work:
• Correctness of “natural” algorithms in $P$
• Basic Linear Algebra
• Combinatorial Theorems
  …
What Theorems Can $PV$ Prove?

Prior work:
• Correctness of “natural” algorithms in P
• Basic Linear Algebra
• Combinatorial Theorems
...

This work:
Many crypto algorithms are “natural”:
ElGamal Encryption
Regev’s Encryption
Puncturable PRFs
...

What Theorems Can $PV$ Prove?

**Prior work:**
- Correctness of “natural” algorithms in P
- Basic Linear Algebra
- Combinatorial Theorems
  ...

**This work:**
- Many crypto algorithms are “natural”:
  - ElGamal Encryption
  - Regev’s Encryption
  - Puncturable PRFs
  ...

**Unprovable Theorems (assuming Factoring):**
- Fermat’s Little Theorem
- Correctness for “Primes is in P”
Our Results III: Applications

SNARGs with CRS size $\text{poly}(\lambda, T_R)$ for $L \in NP \cap \text{coNP}$, if
Our Results III: Applications

SNARGs with CRS size $\text{poly}(\lambda, T_{\bar{R}})$ for $L \in NP \cap \text{coNP}$, if

“$L \cap \bar{L} = \phi$” is provable in $PV$,

$(\vdash_{PV} \bar{R}(x, \bar{w}) = 1 \rightarrow R(x, w) = 0)$

$R$ (resp. $\bar{R}$) is NP-relation machine of $L$ (resp. $\bar{L}$)
Our Results III: Applications

SNARGs with CRS size $\text{poly}(\lambda, T_{\overline{R}})$ for $L \in NP \cap \text{coNP}$, if

"$L \cap \overline{L} = \phi$" is provable in $PV$,

$(\vdash_{PV} \overline{R}(x, \overline{w}) = 1 \rightarrow R(x, w) = 0)$

$R$ (resp. $\overline{R}$) is NP-relation machine of $L$ (resp. $\overline{L}$)

(Also apply to witness encryptions with ciphertext size $\text{poly}(\lambda, T_{\overline{R}})$)
How do we leverage math. proofs?
How do we leverage math. proofs?

(An Overview)
δ-Equivalence for Circuits
$\delta$-Equivalence for Circuits

$C$: 

$C'$
$\delta$-Equivalence for Circuits

$C$ and $C'$ are almost the same, except for a functionality equivalent sub-circuit of size $O(\log n)$
**EF-Proofs imply δ-Equivalence**

Poly. EF proof for $C_0(x) \iff C_1(x)$

\[
\begin{align*}
C_0 &\equiv C^{(1)} \\
C^{(2)} &\quad C^{(3)} \\
&\quad \vdots
\end{align*}
\]

$C^{(i)}$ and $C^{(i+1)}$ are δ-equivalent

$C^{(poly)} \equiv C_1$
Focus: iO for $\delta$-Equivalent Ckts
Focus: iO for \(\delta\)-Equivalent Ckts

Assume iO for \(\delta\)-Equivalent Ckts: \(\delta iO\)
Focus: iO for $\delta$-Equivalent Ckts

Assume iO for $\delta$-Equivalent Ckts: $\delta iO$

$$\delta iO(C^{(1)}) \quad \delta iO(C^{(2)}) \quad \delta iO(C^{(3)}) \quad \ldots \quad \delta iO(C^{(\ell)})$$
Focus: iO for $\delta$-Equivalent Ckts

Assume iO for $\delta$-Equivalent Ckts: $\delta iO$

$$\delta iO(C^{(1)}) \approx \delta iO(C^{(2)}) \approx \delta iO(C^{(3)}) \ldots \delta iO(C^{(\ell_1)})$$
Focus: iO for $\delta$-Equivalent Ckts

Assume iO for $\delta$-Equivalent Ckts: $\delta iO$

\[
\delta iO(C^{(1)}) \approx \delta iO(C^{(2)}) \approx \delta iO(C^{(3)}) \ldots \delta iO(C^{(\ell)})
\]

\[
\Rightarrow \delta iO(C_0) \approx_c \delta iO(C_1)
\]
Focus: iO for $\delta$-Equivalent Ckts

Assume iO for $\delta$-Equivalent Ckts: $\delta iO$

$$\delta iO(C^{(1)}) \approx \delta iO(C^{(2)}) \approx \delta iO(C^{(3)}) \ldots \delta iO(C^{(\ell)})$$

$$\Rightarrow \delta iO(C_0) \approx_c \delta iO(C_1)$$

**Total Security Loss** = $\ell' \cdot$Loss of $\delta iO \quad (\ell' = poly)$
Focus: iO for $\delta$-Equivalent Ckts

Assume iO for $\delta$-Equivalent Ckts: $\delta iO$

$$
\delta iO(C^{(1)}) \approx \delta iO(C^{(2)}) \approx \delta iO(C^{(3)}) \ldots \delta iO(C^{(\ell)})
$$

$$
\Rightarrow \delta iO(C_0) \approx_c \delta iO(C_1)
$$

**Total Security Loss** = $\ell' \cdot \text{Loss of } \delta iO \quad (\ell' = \text{poly})$

If loss of $\delta iO$ is independent of $|input|$, so is the total loss.
Constructing $\delta iO$
Constructing $\delta iO$
Constructing $\delta iO$

“Gate-by-Gate” Obfuscation:
Obfuscate each gate separately
Constructing $\delta iO$

“Gate-by-Gate” Obfuscation:
Obfuscate each gate separately
Constructing $\delta iO$

"Gate-by-Gate" Obfuscation: Obfuscate each gate separately

Topology is preserved
Constructing $\delta_{iO}$

"Gate-by-Gate" Obfuscation:
Obfuscate each gate separately

Topology is preserved
Constructing $\delta iO$

“Gate-by-Gate” Obfuscation:
Obfuscate each gate separately

Topology is preserved
Security Proof w/o $2^{\|input\|}$-Loss
Security Proof w/o $2^{|input|}$-Loss

$C$: 

$C'$: 

$C, C': \delta$-Equivalent
Security Proof w/o $2^{|input|}$-Loss

$C$, $C': \delta$-Equivalent
Security Proof w/o $2^{|input|}$-Loss

$\mathcal{C}$, $\mathcal{C}'$: $\delta$-Equivalent
Security Proof w/o $2^{\text{input}}$-Loss

$C$, $C'$: $\delta$-Equivalent
Security Proof w/o $2^{|\text{input}|}$-Loss

$C, C': \delta$-Equivalent
Security Proof w/o $2^{|input|}$-Loss

Check all inputs to Sub-ckt

Security Loss: $2^{|\text{subckt input}|} = 2^{O(\log n)} = \text{poly}$!
Security Proof w/o $2^{\left|\text{input}\right|}$-Loss

$C$: $C'$: $\delta$-Equivalent

Check all inputs to Sub-ckt

Security Loss: $2^{\left|\text{subckt input}\right|} = 2^{O(\log n)} = \text{poly}$!
Security Proof w/o $2^{|input|}$-Loss

$C$: 

$C'$: 

$C, C'$: $\delta$-Equivalent

Check all inputs to $\textbf{Sub-ckt}$

Security Loss: $2^{|\text{subckt input}|} = 2^{O(\log n)} = \text{poly}$!
Technical Details

• $\mathcal{EF}$-Proofs $\Rightarrow$ $\delta$-Equivalence
• Construct $\delta iO$
• iO for Turing machines
Technical Details

• $\mathcal{EF}$-Proofs $\Rightarrow$ $\delta$-Equivalence
• Construct $\delta iO$
• iO for Turing machines
Goal: $\mathcal{EF}$-Proofs $\Rightarrow$ $\delta$-Equivalence

Poly. $\mathcal{EF}$ proof for $C_0(x) \leftrightarrow C_1(x)$

$C_0 \equiv C^{(1)}$  

$C^{(2)}$  

$C^{(3)}$  

...  

$C^{(poly)} \equiv C_1$

$\delta$-equivalent
Goal: $\mathcal{EF}$-Proofs $\Rightarrow$ $\delta$-Equivalence

Poly. $\mathcal{EF}$ proof for $C_0(x) \leftrightarrow C_1(x)$

Alternative View: A sequence of local changes
Key Observation

Proofs in logic systems are “local”
Key Observation

Proofs in logic systems are “local”
(Similar to $\delta$-equivalence)
Key Observation

Proofs in logic systems are “local”
(Similar to $\delta$-equivalence)

Each line in $\mathcal{E}\mathcal{F}$-proofs is also a circuit
(Can be used to modify circuits)
Stage I: Grow $C_1$

Add Gates in $C_1$ one-by-one
Stage I: Grow $C_1$

Add Gates in $C_1$ *one-by-one*

Set output as $C_0(x)$
Stage I: Grow $C_1$

Add Gates in $C_1$ one-by-one

Set output as $C_0(x)$

$\delta$-Equivalence
When a gate is added, its output is not used anywhere
Stage I: Grow $C_1$

Add Gates in $C_1$ one-by-one

Set output as $C_0(x)$

$\delta$-Equivalence
When a gate is added, its output is not used anywhere
Stage II: Grow the Proof

$\mathcal{EF}$-Proof of $C_0(x) \leftrightarrow C_1(x): \theta_1, \theta_2, \ldots, \theta_\ell$
Stage II: Grow the Proof

$\mathcal{EF}$-Proof of $C_0(x) \leftrightarrow C_1(x)$: $\theta_1, \theta_2, \ldots, \theta_\ell$
Stage II: Grow the Proof

$\mathcal{EF}$-Proof of $C_0(x) \leftrightarrow C_1(x)$: $\theta_1, \theta_2, \ldots, \theta_\ell$

Add $\theta_i$ one-by-one
Stage II: Grow the Proof

$\mathcal{EF}$-Proof of $C_0(x) \leftrightarrow C_1(x)$: $\theta_1, \theta_2, \ldots, \theta_\ell$

Add $\theta_i$ one-by-one
Stage II: Grow the Proof

$\mathcal{EF}$-Proof of $C_0(x) \leftrightarrow C_1(x)$: $\theta_1, \theta_2, \ldots, \theta_\ell$

**Intuition:** $\theta_i$’s (i.e. lines of the proof) are “true”, so the functionality is preserved.
Stage II: \( \delta \)-Equivalence

\[ i \text{-th Step: Add } \theta_i \]

**Before:** \( C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \)

**After:** \( C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \land \theta_i \)
Stage II: $\delta$-Equivalence

$i$-th Step: Add $\theta_i$

**Before:** $C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1}$

**After:** $C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \land \theta_i$

How $\theta_i$ is derived:
Stage II: $\delta$-Equivalence

$i$-th Step: Add $\theta_i$

**Before:** $C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1}$

**After:** $C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \land \theta_i$

How $\theta_i$ is derived:

- Axiom
Stage II: $\delta$-Equivalence

**i-th Step: Add $\theta_i$**

**Before:** $C_0(x) \land \theta_1 \land \ldots \land \theta_{i-1} \land 1$

**After:** $C_0(x) \land \theta_1 \land \ldots \land \theta_{i-1} \land \theta_i$

How $\theta_i$ is derived:

- Axiom
Stage II: $\delta$-Equivalence

$i$-th Step: Add $\theta_i$

**Before:** \[ C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \land 1 \]

**After:** \[ C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \land \theta_i \]

How $\theta_i$ is derived:

- Axiom \[ 1 \equiv \theta_i \text{ (Axioms are tautologies)} \]
Stage II: $\delta$-Equivalence

**i-th Step:** Add $\theta_i$

**Before:**
$$C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \land 1$$

**After:**
$$C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \land \theta_i$$

**How $\theta_i$ is derived:**

- Axiom
Stage II: $\delta$-Equivalence

*i*-th Step: Add $\theta_i$

**Before:**

\[
C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \land 1
\]

**After:**

\[
C_0(x) \land \theta_1 \land \cdots \land \theta_{i-1} \land \theta_i
\]

How $\theta_i$ is derived:

- Axiom
- Inference Rule: Modus Ponens ($p, p \rightarrow q \vdash q$)
Stage II: $\delta$-Equivalence

$i$-th Step: Add $\theta_i$.

**Before:**

**After:**

How $\theta_i$ is derived:

- Axiom
- Inference Rule: Modus Ponens ($p, p \to q \vdash q$)
Stage II: $\delta$-Equivalence

**i-th Step:** Add $\theta_i$.

**Before:** $C_0(x) \land p \land \cdots \land (p \rightarrow q) \land \cdots$

**After:** $C_0(x) \land p \land \cdots \land (p \rightarrow q) \land \cdots \land q$

How $\theta_i$ is derived:

- Axiom
- Inference Rule: Modus Ponens $(p, p \rightarrow q \vdash q)$
Stage II: $\delta$-Equivalence

$i$-th Step: Add $\theta_i$

**Before:** $C_0(x) \land p \land \cdots \land (p \rightarrow q) \land \cdots$

**After:** $C_0(x) \land p \land \cdots \land (p \rightarrow q) \land \cdots \land q$

How $\theta_i$ is derived:

- Axiom
- Inference Rule: Modus Ponens $(p, p \rightarrow q \vdash q)$

\[ p \land (p \rightarrow q) \equiv p \land (p \rightarrow q) \land q \]
Stage III: **Switch** $o_0$ to $o_1$

$$o_0 \land \theta_1 \land \cdots \land \theta_\ell$$
Stage III: Switch $o_0$ to $o_1$

\[ o_0 \land \theta_1 \land \cdots \land \theta_\ell \]
\[ o_1 \land \theta_1 \land \cdots \land \theta_\ell \]
Stage III: Switch $o_0$ to $o_1$

$\delta$-Equivalence

$\theta_\ell$ is “$o_0 \leftrightarrow o_1$” (A proof of $C_0(x) \leftrightarrow C_1(x)$ must end with $o_0 \leftrightarrow o_1$)
Stage III: Switch $o_0$ to $o_1$

\[ o_0 \land \theta_1 \land \cdots \land \theta_\ell \]

$\delta$-Equivalence

$\theta_\ell$ is "$o_0 \leftrightarrow o_1$" (A proof of $C_0(x) \leftrightarrow C_1(x)$ must end with $o_0 \leftrightarrow o_1$)

\[ o_0 \land (o_0 \leftrightarrow o_1) \equiv o_1 \land (o_0 \leftrightarrow o_1) \]
Stage IV: **Shrink** the Proof

\[ o_1 \land \theta_1 \land \cdots \land \theta_\ell \]
Stage IV: Shrink the Proof

$C_0$

$C_1$

$o_0 \land \theta_1 \land \cdots \land \theta_\ell$

Delete $\theta_i$

one-by-one
Stage IV: **Shrink the Proof**

\[ o_0 \wedge \theta_1 \wedge \cdots \wedge \theta_\ell \]

Delete \( \theta_i \) one-by-one
Stage IV: **Shrink the Proof**

\[ C!_0 \mid C!_1 \]

\[ o_0 \mid o_1 \wedge \theta_1 \wedge \ldots \wedge \theta_\ell \]

Delete \( \theta_i \) \( \text{one-by-one} \)

\[ C_0 \mid C_1 \]

**\( \delta\)-Equivalence:** Similar to “Growing the proof” Stage
Stage V: Shrink $C_0$
Stage V: Shrink $C_0$
Stage V: **Shrink** $C_0$

Delete $C_0$ gate-by-gate
Stage V: *Shrink* $C_0$

Delete $C_0$ *gate-by-gate*
Stage V: \textbf{Shrink} $\mathcal{C}_0$

\textcolor{blue}{$\delta$-Equivalence:}
Before we delete a gate, the output of that gate is never used.
More Details: **Multi-Arity Gates?**

**We Use:** Multi-arity $\land$-Gate

$$C_0(x) \land \theta_1 \land \theta_2 \ldots \land \theta_\ell$$
More Details: **Multi-Arity Gates?**

**We Use:** Multi-arity $\land$-Gate

$$C_0(x) \land \theta_1 \land \theta_2 ... \land \theta_\ell$$

**δiO:** Only Support $O(1)$-arity Gates

iO for Ckts
More Details: **Multi-Arity Gates?**

**We Use:** Multi-arity \( \land \)-Gate

\[
C_0(x) \land \theta_1 \land \theta_2 \ldots \land \theta_\ell
\]

**Arity-2 \( \land \)-Tree**

\[
\begin{array}{c}
\land \\
C_0(x) \quad \theta_1 \quad \ldots \quad \theta_\ell
\end{array}
\]

**\( \delta\text{i}O\): Only Support \( O(1) \)-arity Gates**

\[
\begin{array}{c}
\land \\
iO \text{ for Ckts}
\end{array}
\]
Technical Details

- $\mathcal{EF}$-Proofs $\Rightarrow$ $\delta$-Equivalence
- Construct $\delta iO$
- iO for Turing machines
Gate-by-Gate Obfuscation
Gate-by-Gate Obfuscation

\[ g_i O (G_{a t e_g}) \]

\[ \delta i O \]
Gate-by-Gate Obfuscation

\[
\delta iO : \text{Secret key encryption under key } K
\]

\[
iO(Gate_g)
\]
Gate-by-Gate Obfuscation

\[ \text{Gate}_g(\begin{array}{c} m_l \\ K_l \end{array}, \begin{array}{c} m_r \\ K_r \end{array}) \]

Decrypt \( m_l, m_r \)
\[ m_o = g(w_l, w_r) \]

Output: \( m_o^{K_o} \)

\( \Box \): Secret key encryption under key \( K \)

Input/output: 
\( \delta_iO \)

\( iO(\text{Gate}_g) \)
Mix-and-Match Attack
Mix-and-Match Attack

Input: $x$
Mix-and-Match Attack

Input: $x$

Input: $x'$

...
Mix-and-Match Attack

Input: $x$

Mix-n-Match

Input: $x'$
Mix-and-Match Attack

Input: $x$

Input: $x'$

Mix-n-Match
Mix-and-Match Attack

Input: $x$

Mix-n-Match

Input: $x'$

The obfuscated gate reveals more info than it should do.
Add Authentication
Add Authentication

∀ wire $w$, sign $ct_w$ with $x$:

$$\sigma_w := MAC_{K_w}(ct_w || x)$$
Add Authentication

∀ wire w, sign $ct_w$ with $x$:

$\sigma_w := MAC_{K_w}(ct_w || x)$

\[ Gate_g(ct_l, ct_r, \sigma_l, \sigma_r, x) \]

Verify MAC $\sigma_l, \sigma_r$ w.r.t. $l, r$

...(Decrypt, compute $g$, and re-encrypt)...

Also sign and output $\sigma_o$ w.r.t. $o$
Add Authentication

∀ wire $w$, sign $ct_w$ with $x$:

$$\sigma_w := MAC_{K_w}(ct_w || x)$$

\[
\text{Gate}_g\left(ct_l, ct_r, \sigma_l, \sigma_r, x\right)
\]

Verify MAC $\sigma_l, \sigma_r$ w.r.t. $l, r$

...(Decrypt, compute $g$, and re-encrypt)...

Also sign and output $\sigma_o$ w.r.t. $o$

$x$ is too long!
Add Authentication

∀ wire $w$, sign $ct_w$ with $x$:
\[
\sigma_w := MAC_{K_w}(ct_w || x)
\]

Verify MAC $\sigma_l, \sigma_r$ w.r.t. $l, r$

...(Decrypt, compute $g$, and re-encrypt)...

Also sign and output $\sigma_o$ w.r.t. $o$

$\text{Gate}_g (ct_l, ct_r, \sigma_l, \sigma_r, x)$

$x$ is too long!

Gate $g$ may not depend on the entire $x$
(e.g. $NC^0$ circuits)
Define Dependence

\[ \text{Dep}(l) \]

\[ \text{Dep}(r) \]
Define Dependence

\[ \text{Dep}(l) := \{w \mid l \text{ depends on wire } w\} \]
Define Dependence

\[
\text{Dep}(l) : = \{ w \mid l \text{ depends on wire } w \}
\]

\[
CT_l : = \{ \text{ciphertext of } w \}_{w \in \text{Dep}(l)} \text{ (An Index Set)}
\]
Define Dependence

\[ \text{Dep}(l) := \{ w | l \text{ depends on wire } w \} \]

\[ CT_l := \{ \text{ciphertext of } w \}_w \in \text{Dep}(l) \text{ (An Index Set)} \]

\[ (\text{Dep}(r), CT_r: \text{Similar}) \]
Use $CT_l, CT_r$ in $Gate_g$

$$Gate_g(ct_l, ct_r, \sigma_l, \sigma_r, CT_l, CT_r)$$

Check $\sigma_l =? MAC_{k_l}(ct_l || CT_l)$
Check $\sigma_r =? MAC_{k_r}(ct_r || CT_r)$

...(Decrypt, compute $g$, and re-encrypt)...
Use $CT_l, CT_r$ in $Gate_g$

$Gate_g(ct_l, ct_r, \sigma_l, \sigma_r, CT_l, CT_r)$

Check $\sigma_l =? MAC_{k_l}(ct_l||CT_l)$
Check $\sigma_r =? MAC_{k_r}(ct_r||CT_r)$

Check **consistency** of $CT_l$ and $CT_r$

...(Decrypt, compute $g$, and re-encrypt)...

Dep($l$)   Dep($r$)

$g$
Use $CT_l, CT_r$ in $Gate_g$

$Gate_g(\text{ct}_l, \text{ct}_r, \sigma_l, \sigma_r, CT_l, CT_r)$

- Check $\sigma_l = \text{MAC}_{k_l}(\text{ct}_l || CT_l)$
- Check $\sigma_r = \text{MAC}_{k_r}(\text{ct}_r || CT_r)$
- Check consistency of $CT_l$ and $CT_r$

...(Decrypt, compute $g$, and re-encrypt)...

$CT_l$ and $CT_r$ are Consistent:

$CT_l, CT_r$ contains same ciphertexts in $\text{Dep}(l) \cap \text{Dep}(r)$
Proof of Security (High Level)

For any \( \delta \)-Equivalent Ckts:

\[ C_0 \quad C_1 \]
Proof of Security (High Level)

For any $\delta$-Equivalent Ckts:

$\delta iO(C_0)$
Proof of Security (High Level)

For any \( \delta \)-Equivalent Ckts:

\[ \delta iO(C_0) \]

\[ C_0 \]

\[ C_1 \]
Proof of Security (High Level)

For any $\delta$-Equivalent Ckts:

$\delta iO(C_0)$

Direct-Gate$_g(\text{ct}_l, \text{ct}_r, \sigma_l, \sigma_r, \text{CT}_l, \text{CT}_r)$

...(check MACs & consistency)...

...(encrypt output wire)...

$C_0$

$C_1$
Proof of Security (High Level)

For any $\delta$-Equivalent Ckts:

$\delta iO(C_0)$

$\text{Direct-Gate}_g(\text{ct}_l, \text{ct}_r, \sigma_l, \sigma_r, CT_l, CT_r)$

...(check MACs & consistency)...

Sub-ckt.input $\leftarrow$ Decryt $(CT_l, CT_r)$

...(encrypt output wire)...

$C_0$

$C_1$
Proof of Security (High Level)

For any \( \delta \text{-Equivalent Ckts:} \)

\[
\delta iO(C_0)
\]

\[
\text{Direct-Gate}_g(ct_l, ct_r, \sigma_l, \sigma_r, CT_l, CT_r)
\]

...(check MACs & consistency)...

\[
\text{Sub-ckt.input} \leftarrow \text{Decryt} (CT_l, CT_r)
\]

\boxed{\text{Directly Compute Sub-ckt(sub-ckt.input)}}

...(encrypt output wire)...

\[C_0\]

\[C_1\]
Proof of Security (High Level)

For any \(\delta\)-Equivalent Ckts:

\[
\delta iO(C_0) \approx MAC\ security
\]

Direct Gate \(g(ct_l, ct_r, \sigma_l, \sigma_r, CT_l, CT_r)\)

...(check MACs & consistency)...

Sub-ckt.input \(\leftarrow\) Decryt \((CT_l, CT_r)\)

Directly Compute Sub-ckt(sub-ckt.input)

...(encrypt output wire)...

For any \(\delta\)-Equivalent Ckts:
Proof of Security (High Level)

\[ \delta iO(C_0) \approx \text{MAC security} \]

Sub-ckt: Direct-Gates
Proof of Security (High Level)

\[ \delta iO(C_0) \approx \text{MAC security} \approx \text{Sub-ckt: Direct-Gates} \approx \text{iO Security} \]

( □ ≡ □ )
Proof of Security (High Level)

\[ \delta iO(C_0) \approx MAC \text{ security} \]

Sub-ckt: Direct-Gates

\[ \approx \]

Sub-ckt: Direct-Gates but use \( C_1 \)

iO Security

\( \equiv \)
Proof of Security (High Level)

\[ \delta_{iO}(C_0) \approx \text{MAC security} \]
\[ \approx \text{Sub-ckt: Direct-Gates} \]
\[ \approx \text{iO Security} \]
\[ (\equiv) \]
\[ \approx \text{Sub-ckt: Direct-Gates but use } C_1 \]
\[ \approx \delta_{iO}(C_1) \]
Proof of Security (High Level)

Extend this idea to general circuits?

**Challenge**: |CT₁| is too large.

\[ \delta_{iO}(C₀) \approx \text{MAC security} \]

\[ \approx \quad \text{Sub-ckt: Direct-Gates} \]

\[ \approx \quad \text{iO Security} \]

\[ ( \quad \equiv \quad ) \]

\[ \delta_{iO}(C₁) \approx \quad \text{Sub-ckt: Direct-Gates but use } C₁ \]
Proof of Security (High Level)

Extend this idea to general circuits?

Challenge: $|CT_l|$ is too large.

$\delta iO(C_0) \approx \text{MAC security}
\approx \text{Sub-ckt: Direct-Gates}
\approx \text{iO Security}
(\equiv)
\approx \delta iO(C_1)$

Observation: $g$ only depends on sub-ckt input

Sub-ckt: Direct-Gates but use $C_1$
Somewhere **Statistical** Binding (SSB) Hash

[Hubacek-Wichs’15, Okamoto-Pietrzak-Waters-Wichs’15]
Somewhere **Statistical** Binding (SSB) Hash

[Hubacek-Wichs’15, Okamoto-Pietrzak-Waters-Wichs’15]

**Normal Mode**

\[ K \]
**Somewhere Statistical Binding (SSB) Hash**

[Hubacek-Wichs’15, Okamoto-Pietrzak-Waters-Wichs’15]

<table>
<thead>
<tr>
<th>Normal Mode</th>
<th>≈ c</th>
<th>Trapdoor Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td></td>
<td>$K^*(S \subseteq [n])$</td>
</tr>
</tbody>
</table>
Somewhere **Statistical Binding (SSB) Hash**

[Hubacek-Wichs’15, Okamoto-Pietrzak-Waters-Wichs’15]

\[
\begin{array}{c|c}
\text{Normal Mode} & \text{Trapdoor Mode} \\
K & K^*(S \subseteq [n]) \\
\end{array}
\]

\[ h \leftarrow \text{SSB}(K, m_1, m_2, \ldots, m_n) \]
Somewhere **Statistical Binding** (SSB) Hash
[Hubacek-Wichs’15, Okamoto-Pietrzak-Waters-Wichs’15]

\[
\begin{align*}
\text{Normal Mode} & \quad K \quad \approx_c \quad \text{Trapdoor Mode} \\
 & \quad K^\ast(S \subseteq [n]) \\
\end{align*}
\]

\[
\begin{align*}
\text{Normal Mode} & \quad h \leftarrow \text{SSB}(K,m_1,m_2,\ldots,m_n) \\
\text{Trapdoor Mode} & \quad h \leftarrow \text{SSB}(K^\ast, m_1, m|_S, \ldots, m_n)
\end{align*}
\]
Somewhere **Statistical Binding (SSB) Hash**

[Hubacek-Wichs’15, Okamoto-Pietrzak-Waters-Wichs’15]

\[ h \leftarrow \text{SSB}(K, m_1, m_2, \ldots, m_n) \]

\[ h \leftarrow \text{SSB}(K^*, m_1, m|_S, \ldots, m_n) \]

In Our Setting: \((S := \{\text{input wires to sub-ckt}\})\)
SSB Hash $CT_l, CT_r$

Outside $Gate_g$:

$$h_l = SSB(CT_l)$$
$$h_r = SSB(CT_r)$$

$Gate_g(ct_l, ct_r, \sigma_l, \sigma_r, CT_l, CT_r)$

Check $\sigma_l =? MAC_{k_l}(ct_l || CT_l)$
Check $\sigma_r =? MAC_{k_r}(ct_r || CT_r)$

...(Decrypt, compute $g$, and re-encrypt)...)
SSB Hash $CT_l$, $CT_r$

Outside $Gate_g$:

$h_l = SSB(CT_l)$
$h_r = SSB(CT_r)$

$Gate_g(ct_l, ct_r, \sigma_l, \sigma_r, \ldots)$

Check $\sigma_l = ? MAC_{k_l}(ct_l || CT_l)$
Check $\sigma_r = ? MAC_{k_r}(ct_r || CT_r)$

...(Decrypt, compute $g$, and re-encrypt)
SSB Hash $CT_l$, $CT_r$

Outside $Gate_g$:

$$h_l = SSB(CT_l)$$
$$h_r = SSB(CT_r)$$

$Gate_g(\text{ct}_l, \text{ct}_r, \sigma_l, \sigma_r, h_l, h_r)$

Check $\sigma_l = ? MAC_{k_l}(\text{ct}_l || CT_l)$

Check $\sigma_r = ? MAC_{k_r}(\text{ct}_r || CT_r)$

...(Decrypt, compute $g$, and re-encrypt)...
SSB Hash $CT_l, CT_r$

**Outside*** $Gate_g$:  

- $h_l = SSB(CT_l)$
- $h_r = SSB(CT_r)$

**$Gate_g(\text{ct}_l, \text{ct}_r, \sigma_l, \sigma_r, h_l, h_r)$**

- Check $\sigma_l = ? MAC_{k_l}(\text{ct}_l || )$
- Check $\sigma_r = ? MAC_{k_r}(\text{ct}_r || )$

...(Decrypt, compute $g$, and re-encrypt)
SSB Hash $CT_l, CT_r$

Outside $Gate_g$:

\[
\begin{align*}
  h_l &= SSB(CT_l) \\
  h_r &= SSB(CT_r)
\end{align*}
\]

\[
Gate_g( ct_l, ct_r, \sigma_l, \sigma_r, h_l, h_r )
\]

Check $\sigma_l =? MAC_{k_l}(ct_l || h_l)$

Check $\sigma_r =? MAC_{k_r}(ct_r || h_r)$

...(Decrypt, compute $g$, and re-encrypt)
SSB Hash $CT_l, CT_r$

Outside $Gate_g$:

\[
\begin{aligned}
    h_l &= SSB(CT_l) \\
    h_r &= SSB(CT_r)
\end{aligned}
\]

\[
\begin{aligned}
    Gate_g(\text{ct}_l, \text{ct}_r, \sigma_l, \sigma_r, \ h_l, h_r )
\end{aligned}
\]

- Check $\sigma_l = ? MAC_{k_l}(\text{ct}_l|| h_l )$
- Check $\sigma_r = ? MAC_{k_r}(\text{ct}_r|| h_r )$

Check consistency of $CT_l$ and $CT_r$

...(Decrypt, compute $g$, and re-encrypt)
SSB Hash $CT_l, CT_r$

**Outside Gate**: $G_{\text{outside}}$

$$h_l = SSB(CT_l)$$
$$h_r = SSB(CT_r)$$

**Gate** $G_{\text{in}}(ct_l, ct_r, \sigma_l, \sigma_r, h_l, h_r)$

- Check $\sigma_l = \overset?= MAC_{k_l}(ct_l|| h_l)$
- Check $\sigma_r = \overset?= MAC_{k_r}(ct_r|| h_r)$

Check consistency of $CT_l$ and $CT_r$???

...(Decrypt, compute $g$, and re-encrypt)
SSB Hash $CT_l, CT_r$

**Outside Gate** $g$:

$h_l = SSB(CT_l)$

$h_r = SSB(CT_r)$

$Gate_g(ctl, ctr, \sigma_l, \sigma_r, h_l, h_r)$

Check $\sigma_l = ? MAC_{k_l}(ct_l || h_l)$

Check $\sigma_r = ? MAC_{k_r}(ct_r || h_r)$

Check consistency of $CT_l$ and $CT_r$???

...(Decrypt, compute $g$, and re-encrypt)...

SNARGs?
No Statistical Soundness
SSB Hash $CT_l, CT_r$

**Outside $Gate_g$:**

$$h_l = SSB(CT_l)$$

$$h_r = SSB(CT_r)$$

$Gate_g(ct_l, ct_r, \sigma_l, \sigma_r, h_l, h_r)$

Check $\sigma_l = ? MAC_{k_l}(ct_l || h_l)$

Check $\sigma_r = ? MAC_{k_r}(ct_r || h_r)$

Check consistency of $CT_l$ and $CT_r$???

...(Decrypt, compute $g$, and re-encrypt)...

---

SNARGs?

No Statistical Soundness

Consistency for sub-ckt input (binding positions) is enough
Recall: SNARGs for Batch-Index
[Choudhuri-Jain-Jin’21]

Index Language: $L = \{i | \exists w: C(i, w) = 1\}$
Recall: SNARGs for Batch-Index
[Choudhuri-Jain-Jin’21]

Index Language: \( L = \{i|\exists w: C(i, w) = 1\} \)
Recall: SNARGs for Batch-Index
[Choudhuri-Jain-Jin’21]

Index Language: $L = \{i | \exists w: C(i, w) = 1\}$

CRS → “1, 2, ... $k \in L$” → CRS
Recall: SNARGs for Batch-Index
[Choudhuri-Jain-Jin’21]

Index Language: $L = \{i | \exists w: C(i, w) = 1\}$

Verify in time $\text{poly}(\lambda, |C|, \log k)$
Recall: SNARGs for Batch-Index
[Choudhuri-Jain-Jin’21]

Index Language: $L = \{i | \exists w: C(i, w) = 1\}$

CRS

“1, 2, ... $k \in L$”

Verify in time $\text{poly}(\lambda, |C|, \log k)$

Accept/Reject
Recall: SNARGs for Batch-Index
[Choudhuri-Jain-Jin’21]

Index Language: $L = \{i | \exists w: C(i, w) = 1\}$

Completeness:
If $[k] \subseteq L$, honestly generated proof will be accepted.
Somewhere Statistical Soundness

<table>
<thead>
<tr>
<th>Normal Mode</th>
<th>$\approx_c$</th>
<th>Trapdoor Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CRS$</td>
<td>$\approx_c$</td>
<td>$CRS^*(S)$</td>
</tr>
</tbody>
</table>
Somewhere Statistical Soundness

Normal Mode

\[ CRS \approx_c CRS^*(S) \]

Trapdoor Mode
Somewhere Statistical Soundness

\[ CRS \approx_c CRS^*(S) \]
Somewhere Statistical Soundness

Normal Mode

\[ CRS \]

\[ \approx_c \]

Trapdoor Mode

\[ CRS^* (S) \]
Somewhere Statistical Soundness

\[
\begin{array}{ccc}
\text{Normal Mode} & \approx_c & \text{Trapdoor Mode} \\
CRS & & CRS^*(S)
\end{array}
\]

Statistical Sound for $S$:
If $S \subseteq L$ does not hold, then unbounded adv. can’t find cheating proof.
Construct Succinct Proof of Consistency
Construct Succinct Proof of Consistency

SSB by Merkle Tree

SSB($CT_l$)

$c_{t1}$ $c_{t2}$ $c_{tN}$

SSB($CT_r$)

$c_{t1}'$ $c_{tN}'$
Construct Succinct Proof of Consistency

SSB by Merkle Tree

\[ \text{Prove: } \forall w \in [N], \exists \text{ local openings } \& \ ct_w, ct'_w \text{ s.t.} \]
\[ \text{if } ct_w \neq \bot \land ct'_w \neq \bot, \text{ then } ct_w = ct'_w \text{ (consistent)} \]
Construct Succinct Proof of Consistency

SSB by Merkle Tree

Prove: \( \forall w \in [N], \exists \) local openings \& \( ct_w, ct_w' \) s.t. if \( ct_w \neq \bot \land ct_w' \neq \bot \), then \( ct_w = ct_w' \) (consistent)

Proof via SNARGs for Batch-Index
Add **Proof** of Consistency

**Outside** $Gate_g$:

\[
\begin{align*}
  h_l &= SSB(CT_l) \\
  h_r &= SSB(CT_r)
\end{align*}
\]

\[Gate_g(\text{ct}_l, \text{ct}_r, \pi_l, \pi_r, h_l, h_r, \pi)\]

...(Verify the MACs)...

...(Decrypt, compute $g$, and re-encrypt)....
Add **Proof** of Consistency

**Outside** $Gate_g$:

- $h_l = SSB(CT_l)$
- $h_r = SSB(CT_r)$

$\pi$ : consistency proof for $h_l, h_r$

---

$Gate_g(ct_l, ct_r, \sigma_l, \sigma_r, h_l, h_r, \pi)$

...(Verify the MACs)...

...(Decrypt, compute $g$, and re-encrypt)......
Add Proof of Consistency

**Outside** $Gate_g$:

- $h_l = SSB(CT_l)$
- $h_r = SSB(CT_r)$

$\pi$: consistency proof for $h_l, h_r$

$\text{Gate}_g(\text{ct}_l, \text{ct}_r, \sigma_l, \sigma_r, h_l, h_r, \pi)$

...(Verify the MACs)...

Verify the proof $\pi$ w.r.t. $h_l, h_r$

...(Decrypt, compute $g$, and re-encrypt)....
Technical Details

- $\mathcal{EF}$-Proofs $\Rightarrow$ $\delta$-Equivalence
- Construct $\delta iO$
- $iO$ for Turing machines
Recall: Cook’s Translation
Recall: Cook’s Translation

Poly-size $\mathcal{EF}$-Proofs $\xrightarrow{\text{Cook’s Translation}}$ Proofs in $PV$
Recall: Cook’s Translation

Poly-size $\mathcal{EF}$-Proofs $\vdash 01 \ M \ "x = M/ (x)"

Cook’s Translation

Proofs in $PV$

$\vdash_{PV} M_1(x) = M_2(x)$
Recall: Cook’s Translation

Poly-size $\mathcal{EF}$-Proofs $\vdash_{\text{PV}} M_1(x) = M_2(x)$

Cook’s Translation

Proofs in PV

Input length $n$
Recall: Cook’s Translation

Poly-size $\mathcal{EF}$-Proofs $\vdash_{\mathcal{EF}} C_{1,n}(x) \leftrightarrow C_{2,n}(x)$

Cook’s Translation

Input length $n$

Proofs in $PV$ $\vdash_{PV} M_1(x) = M_2(x)$

$C_{b,n}(x)$: Circuit that computes $M_b$ for input $|x| = n$. 
Recall: Cook’s Translation

Poly-size $\mathcal{EF}$-Proofs $\vdash_{\mathcal{EF}} C_{1,n}(x) \leftrightarrow C_{2,n}(x)$

Cook’s Translation

Proofs in $PV$ $\vdash_{PV} M_1(x) = M_2(x)$

Input length $n$

$C_{b,n}(x)$: Circuit that computes $M_b$ for input $|x| = n$. 
Recall: Cook’s Translation

Poly-size $\mathcal{EF}$-Proofs $\vdash 01 M \to x = M/(x)$ $\vdash \mathcal{F} C \to C/(x)$

Proofs in $PV$

Input length $n$

$\vdash_{\mathcal{F}} C_{1,n}(x) \leftrightarrow C_{2,n}(x)$

$\vdash_{PV} M_1(x) = M_2(x)$

$C_{b,n}(x)$: Circuit that computes $M_b$ for input $|x| = n$.

Use $\delta iO$?
iO for TMs \textit{from} \textit{\$iO\$}
iO for TMs \textbf{from} \( \delta iO \)

\( M \)

Turing Machine
iO for TMs from $\delta_{iO}$

$N_0 = \lambda^{\log \lambda}$

All input length $n \leq N_0$
iO for TMs from $\delta iO$

$M$
Turing Machine

\begin{align*}
(N_0 &= \lambda^{\log \lambda}) \\
\text{All Input length} &\Rightarrow n \leq N_0
\end{align*}

$C_1$
$C_2$
\ldots
$C_{N_0}$
iO for TMs from $\delta iO$

$M$

Turing Machine

$(N_0 = \lambda^{\log \lambda})$

All Input length $n \leq N_0$

$C_1$

$C_2$

$\ldots$

$C_{N_0}$

$\delta iO$
iO for TMs from $\delta iO$

$M$ Turing Machine

$\forall$ Input length $n \leq N_0$

$(N_0 = \lambda^{\log \lambda})$

$C_1$

$C_2$

$\ldots$

$C_{N_0}$

$\delta iO(C_1)$

$\delta iO(C_2)$

$\ldots$

$\delta iO(C_{N_0})$

Obfuscated Program
iO for TMs from $\delta iO$

$M$ Turing Machine

$N_0 = \lambda^{\log \lambda}$

All Input length $n \leq N_0$

$C_1$, $C_2$, ..., $C_{N_0}$

$\delta iO(C_1)$, $\delta iO(C_2)$, ..., $\delta iO(C_{N_0})$

Obfuscation time is super-poly!

Obfuscated Program
Efficient Construction
Efficient Construction

$C_1, C_2, \ldots, C_{N_0}$ have a succinct description
Efficient Construction

$C_1, C_2, \ldots, C_{N_0}$ have a succinct description

i.e. $\exists$ circuit $[M](\cdot, \cdot)$, s.t. $[M](n, i)$ outputs the description of $i$-th gate in $C_n$
Efficient Construction

$C_1, C_2, \ldots, C_{N_0}$ have a succinct description

i.e. $\exists$ circuit $[M](\cdot, \cdot)$, s.t. $[M](n, i)$ outputs the description of $i$-th gate in $C_n$

Recall: $\delta iO$ Construction

$g_1, g_2, \ldots, g_{|C|}$  $\xrightarrow{\delta iO}$  $iO(Gate_{g_1})$  $iO(Gate_{g_2})$  $\ldots$  $iO(Gate_{g_{|C|}})$

Generate $Gate_g$ from $[M](\cdot, \cdot)$
Efficient Construction
Efficient Construction

$M$

Turing Machine
Efficient Construction

$M$

Turing Machine
Efficient Construction

\[ M \quad \text{Turing Machine} \quad \rightarrow \quad UGate(\cdot;\cdot;\cdot) \]
Efficient Construction

$M$ Turing Machine $\rightarrow UGate(\cdot,\cdot,\cdot) \rightarrow iO(UGate)$ (iO for small circuit)
Efficient Construction

\( M \)

Turing Machine \( \rightarrow \)

\( UGate(\cdot,\cdot,\cdot) \)

\( \rightarrow \)

\( iO(UGate) \)

(iO for small circuit)

**"Uniform" Gate** \( UGate(n, i, input') \)

Get description of \( i \)-th gate:

\[ g \leftarrow [M](n, i) \]

Emulate

\[ Gate_g(input') \]
Summary & Future Directions
Summary & Future Directions

Inference Rules in Logic systems for Proving Equivalence
Summary & Future Directions

Inference Rules in Logic systems for Proving Equivalence
Summary & Future Directions

Inference Rules in **Logic systems** for Proving Equivalence

Techniques to argue Indistinguishability for iO
Summary & Future Directions

Inference Rules in Logic systems for Proving Equivalence

Techniques to argue Indistinguishability for iO

$\mathcal{EF} / PV$
Summary & Future Directions

Inference Rules in Logic systems for Proving Equivalence

Techniques to argue Indistinguishability for iO

ℰℱ / PV
Summary & Future Directions

Inference Rules in Logic systems for Proving Equivalence

Techniques to argue Indistinguishability for iO

\( \mathcal{EF} / PV \)  \hspace{2cm}  \( \delta \)-equivalence & \( \delta iO \)
Summary & Future Directions

Inference Rules in Logic systems for Proving Equivalence

Techniques to argue Indistinguishability for iO

\( \mathcal{EF} / PV \)

\( \delta \)-equivalence & \( \delta iO \)

**ZFC**
(Zermelo-Fraenkel set theory with axiom of Choice)