Structure-Preserving Compilers from New Notions of Obfuscations

Matteo Campanelli
Protocol Labs

Danilo Francati
Aarhus University

Claudio Orlandi
Aarhus University

Obfuscator

PPT Obfuscator

Circuit. $C : \{0,1\}^n \longrightarrow \{0,1\}^m$

PPT Obfuscator. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C)$

Obfuscator

PPT Obfuscator

Circuit. $C : \{0,1\}^n \longrightarrow \{0,1\}^m$

PPT Obfuscator. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C)$

(1) Correctness. $\forall x \in \{0,1\}^n, C(x) = \widetilde{C}(x)$

(2) Polynomial Slowdown. $\exists p(\cdot), \forall C, |\mathsf{Obf}(1^\lambda, C)| \leq p(|C|)$

Security

Security

(1) Leakage

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

Security

(1) Leakage

(2) Distribution

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

Security

(1) Leakage

(2) Distribution

(3) Functionality

SIM-based

The obfuscation can be simulated.

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

VS.

IND-based

Two obfuscations are indistinguishable.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

(3) Functionality

No restriction

$\forall C$

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

(3) Functionality

No restriction

$\forall C$

VS.

Functionally Eq.

$\forall x, C_0(x) = C_1(x)$

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

(3) Functionality

No restriction

$\forall C$

VS.

Functionally Eq.

$\forall x, C_0(x) = C_1(x)$

VS.

Differing-inputs

Find x such that $C_0(x) \neq C_1(x)$ is hard.

... and more ...

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

(3) Functionality

No restriction

$\forall C$

VS.

Functionally Eq.

$\forall x, C_0(x) = C_1(x)$

VS.

Differing-inputs

Find x such that $C_0(x) \neq C_1(x)$ is hard.

... and more ...

Existing Notions

Virtual Black Box (**VBB**).

Differing-input Obfuscation (**diO**).

Indistinguishability Obfuscation (**iO**).

VBB

VBB-simulation

For every PPT adversary A, there exists a PPT simulator Sim such that for every sampler S, we have:

$$\left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C), \alpha) = 1\right\} \approx_c \left\{\mathsf{Sim}^{C(\cdot)}(1^\lambda, 1^{|C|}, \alpha) = 1\right\}$$

where $(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

VBB

VBB-simulation

For every PPT adversary A, there exists a PPT simulator Sim such that for every sampler S, we have:

$$\left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C), \alpha) = 1\right\} \approx_c \left\{\mathsf{Sim}^{C(\cdot)}(1^\lambda, 1^{|C|}, \alpha) = 1\right\}$$

where $(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Auxiliary Information

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

(3) Functionality

No restriction

$\forall C$

VS.

Functionally Eq.

$\forall x, C_0(x) = C_1(x)$

VS.

Differing-inputs

Find x such that $C_0(x) \neq C_1(x)$ is hard.

... and more ...

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

(3) Functionality

No restriction

$\forall C$

VS.

Functionally Eq.

$\forall x, C_0(x) = C_1(x)$

VS.

Differing-inputs

Find x such that $C_0(x) \neq C_1(x)$ is hard.

... and more ...

Applications of VBB

Applications of VBB

VBB is magic. You can obfuscate any program/circuit and it will work as an ideal oracle.

Applications of VBB

VBB is magic. You can obfuscate any program/circuit and it will work as an ideal oracle.

VBB can be used to implement structure-preserving compilers.

Applications of VBB

VBB is magic. You can obfuscate any program/circuit and it will work as an ideal oracle.

VBB can be used to implement structure-preserving compilers.

Example:

$C_k(m)$:
Return $c \leftarrow \mathsf{Enc}(k, m)$

Applications of VBB

VBB is magic. You can obfuscate any program/circuit and it will work as an ideal oracle.

VBB can be used to implement structure-preserving compilers.

Example:

$C_k(m)$:
Return $c \leftarrow \mathsf{Enc}(k, m)$

$\mathsf{Obf}(1^\lambda, C_k) \rightarrow \widetilde{C}$

Set $pk = \widetilde{C}$
Obtain a PKE

Applications of VBB

VBB is magic. You can obfuscate any program/circuit and it will work as an ideal oracle.

VBB can be used to implement structure-preserving compilers.

Example:

$C_k(m)$:
Return $c \leftarrow \mathsf{Enc}(k, m)$

$\mathsf{Obf}(1^\lambda, C_k) \rightarrow \widetilde{C}$

Set $pk = \widetilde{C}$
Obtain a PKE

Impossibility Results [*]

Some programs/circuits cannot be **VBB**-obfuscated (unconditionally).

Some SKEs cannot be obfuscated into PKEs (unconditionally).

[*] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs.

iO

iO-Sampler

A sampler S is an **iO**-sampler if

$$\mathbb{P}\left[\forall x \in \{0,1\}^n, C_0(x) = C_1(x)\right] \geq 1 - \mathsf{negl}(\lambda)$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

iO

iO-Sampler

A sampler S is an **iO**-sampler if

$$\mathbb{P}\left[\forall x \in \{0,1\}^n, C_0(x) = C_1(x)\right] \geq 1 - \mathsf{negl}(\lambda)$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Indistinguishability

For every **iO**-Sampler S, every PPT adversary A, we have:

$$\left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_0), \alpha) = 1\right\} \approx_c \left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_1), \alpha) = 1\right\}$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

diO

diO-Sampler

A sampler S is an **diO**-sampler if for every PPT adversary A we have:

$$\mathbb{P}\left[C_0(x) \neq C_1(x) \,\middle|\, x \leftarrow \mathsf{A}(1^\lambda, C_0, C_1, \alpha)\right] \leq \mathsf{negl}(\lambda)$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Circuits are known

diO

diO-Sampler

A sampler S is an **diO**-sampler if for every PPT adversary A we have:

$$\mathbb{P}\left[C_0(x) \neq C_1(x) \,\middle|\, x \leftarrow \mathsf{A}(1^\lambda, C_0, C_1, \alpha)\right] \leq \mathsf{negl}(\lambda)$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

diO

diO-Sampler

A sampler S is an **diO**-sampler if for every PPT adversary A we have:

$$\mathbb{P}\left[C_0(x) \neq C_1(x) \,\middle|\, x \leftarrow \mathsf{A}(1^\lambda, C_0, C_1, \alpha)\right] \leq \mathsf{negl}(\lambda)$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Indistinguishability

For every **diO**-sampler S, every PPT adversary A, we have:

$$\left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_0), \alpha) = 1\right\} \approx_c \left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_1), \alpha) = 1\right\}$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

(3) Functionality

No restriction

$\forall C$

VS.

Functionally Eq.

$\forall x, C_0(x) = C_1(x)$

VS.

Differing-inputs

Find x such that $C_0(x) \neq C_1(x)$ is hard.

... and more ...

Security

(1) Leakage

SIM-based

The obfuscation can be simulated.

VS.

IND-based

Two obfuscations are indistinguishable.

(2) Distribution

Every Circuit

$\forall C$ or $\forall C_0, C_1$

VS.

Samplers

$(C, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

or

$(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$

(3) Functionality

No restriction

$\forall C$

VS.

Functionally Eq.

$\forall x, C_0(x) = C_1(x)$

VS.

Differing-inputs

Find x such that $C_0(x) \neq C_1(x)$ is hard.

... and more ...

Limitations of iO and diO

Limitations of iO and diO

They are not powerful enough to implement structure-preserving compilers.

Limitations of iO and diO

They are not powerful enough to implement structure-preserving compilers.

Intuition

We need to include a secret into the circuit.

iO/diO are not enough to deal with secrets.

*Can we obtain structure-preserving transformations from notions of obfuscation weaker than **VBB**?*

New Notions of Obfuscation

Virtual Black Box (**VBB**).

Differing-input Obfuscation (**diO**).

Indistinguishability Obfuscation (**iO**).

New Notions of Obfuscation

Virtual Black Box (**VBB**).

Oracle Differing-input Obfuscation (odiO).

Differing-input Obfuscation (**diO**).

Indistinguishability Obfuscation (**iO**).

New Notions of Obfuscation

Virtual Black Box (**VBB**).

 ***Oracle Indistinguishability Obfuscation (oiO).*** 

 ***Oracle Differing-input Obfuscation (odiO).*** 

Differing-input Obfuscation (**diO**).

Indistinguishability Obfuscation (**iO**).

odiO

odiO-Sampler

A sampler S is an **odiO**-sampler if for every PPT adversary A we have:

$$\mathbb{P}\left[C_0(x) \neq C_1(x) \,\middle|\, x \leftarrow \mathsf{A}^{C_0(\cdot),C_1(\cdot)}(1^\lambda, 1^{|C|}, \alpha)\right] \leq \mathsf{negl}(\lambda)$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Indistinguishability

For every **odiO**-sampler S, every PPT adversary A, we have:

$$\left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_0), \alpha) = 1\right\} \approx_c \left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_1), \alpha) = 1\right\}$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Circuits are secret

odiO

odiO-Sampler

A sampler S is an **odiO**-sampler if for every PPT adversary A we have:

$$\mathbb{P}\left[C_0(x) \neq C_1(x) \,\middle|\, x \leftarrow \mathsf{A}^{C_0(\cdot),C_1(\cdot)}(1^\lambda, 1^{|C|}, \alpha)\right] \leq \mathsf{negl}(\lambda)$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Indistinguishability

For every **odiO**-sampler S, every PPT adversary A, we have:

$$\left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_0), \alpha) = 1\right\} \approx_c \left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_1), \alpha) = 1\right\}$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

oiO

oiO-Sampler

A sampler S is an **oiO**-sampler if for every PPT adversary A we have:

$$\left\{\mathsf{A}^{C_0(\cdot)}(1^\lambda, 1^{|C_0|}, \alpha) = 1\right\} \approx_c \left\{\mathsf{A}^{C_1(\cdot)}(1^\lambda, 1^{|C_1|}\alpha) = 1\right\}$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Indistinguishability

For every **odiO**-sampler S, every PPT adversary A, we have:

$$\left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_0), \alpha) = 1\right\} \approx_c \left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_1), \alpha) = 1\right\}$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Circuits are secret

oiO

oiO-Sampler

A sampler S is an **oiO**-sampler if for every PPT adversary A we have:

$$\left\{\mathsf{A}^{C_0(\cdot)}(1^\lambda, 1^{|C_0|}, \alpha) = 1\right\} \approx_c \left\{\mathsf{A}^{C_1(\cdot)}(1^\lambda, 1^{|C_1|}\alpha) = 1\right\}$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

Indistinguishability

For every **odiO**-sampler S, every PPT adversary A, we have:

$$\left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_0), \alpha) = 1\right\} \approx_c \left\{\mathsf{A}(1^\lambda, \mathsf{Obf}(1^\lambda, C_1), \alpha) = 1\right\}$$

where $(C_0, C_1, \alpha) \leftarrow \mathsf{S}(1^\lambda)$.

iO vs. **diO** vs. **odiO** vs. **oiO**

iO vs. **diO** vs. **odiO** vs. **oiO**

IND-based def.

They have the same
IND-based definition.

iO vs. **diO** vs. **odiO** vs. **oiO**

IND-based def.

They have the same
IND-based definition.

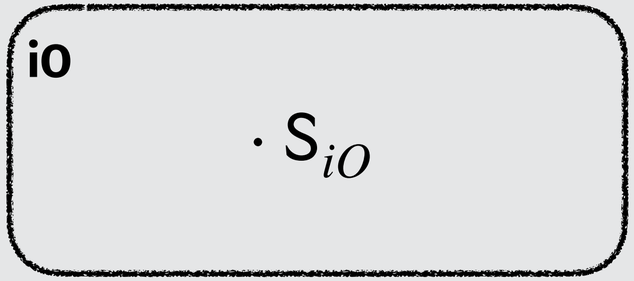
Samplers

iO vs. **diO** vs. **odiO** vs. **oiO**

IND-based def.

They have the same
IND-based definition.

Samplers

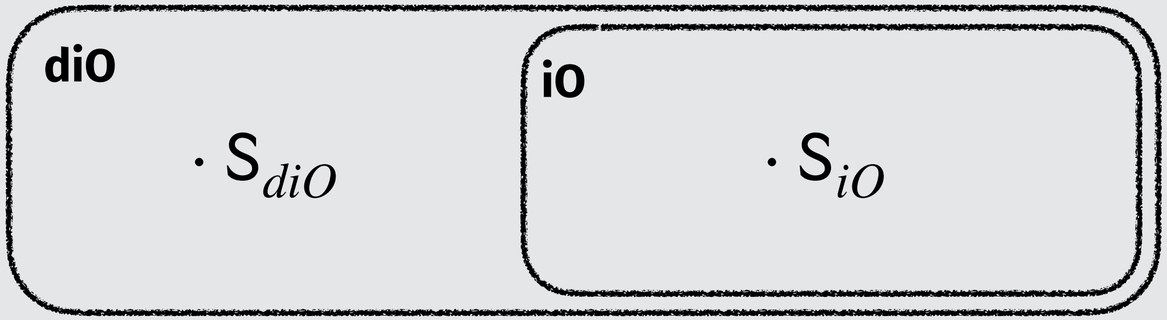


iO vs. diO vs. odiO vs. oiO

IND-based def.

They have the same
IND-based definition.

Samplers

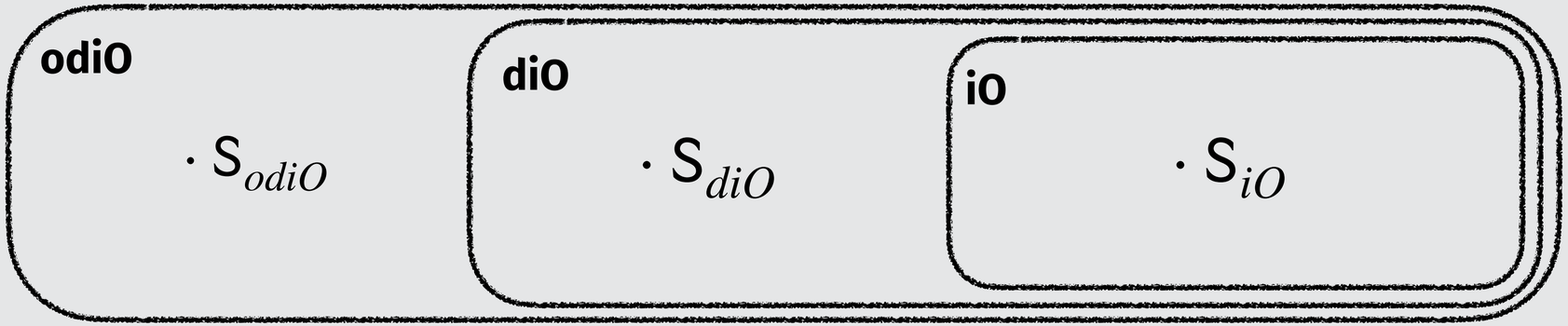


iO vs. **diO** vs. **odiO** vs. **oiO**

IND-based def.

They have the same
IND-based definition.

Samplers

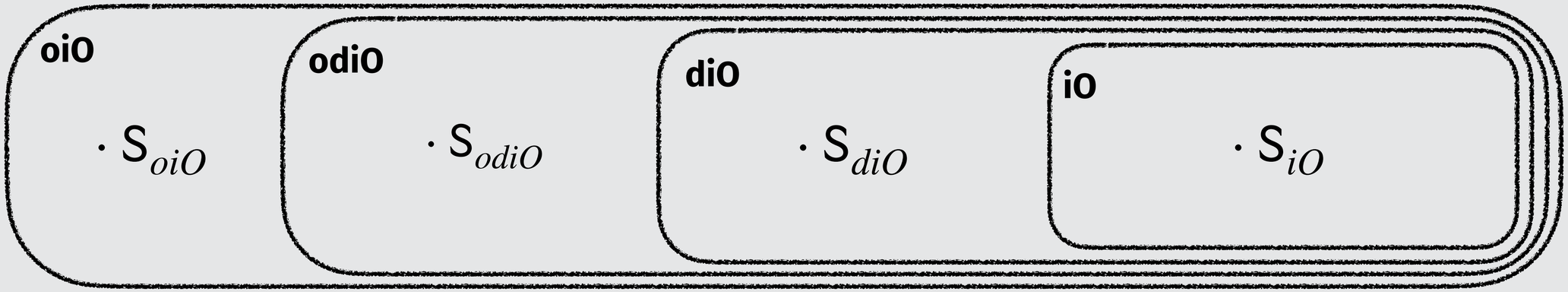


iO vs. **diO** vs. **odiO** vs. **oiO**

IND-based def.

They have the same
IND-based definition.

Samplers



iO vs. **diO** vs. **odiO** vs. **oiO**

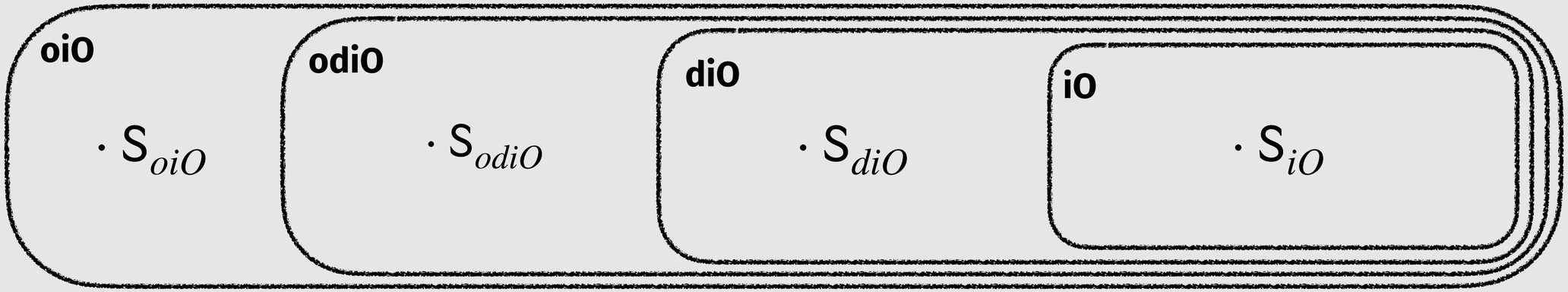
IND-based def.

They have the same
IND-based definition.

Implications

$$\text{iO} \Leftarrow \text{diO} \Leftarrow \text{odiO} \Leftarrow \text{oiO}$$

Samplers



oiO (odiO) vs. **VBB**

oiO (odiO) vs. **VBB**

SIM-based def. vs. IND-based def.

(**VBB**) SIM-based def. $\Rightarrow$ (**oiO**) IND-based def.

oiO (odiO) vs. **VBB**

SIM-based def. vs. IND-based def.

(**VBB**) SIM-based def. $\Rightarrow$ (**oiO**) IND-based def.

Result

$$\mathsf{odiO} \Leftarrow \mathsf{oiO} \Leftarrow \mathsf{VBB}$$

Applications of odiO

***odiO**-based structure-preserving compilers*

Applications of odiO

***odiO**-based structure-preserving compilers*

Designated verifier (DV) non-interactive argument systems

↓

Publicly verifiable (PV) non-interactive argument systems

Applications of odiO

***odiO**-based structure-preserving compilers*

Designated verifier (DV) non-interactive argument systems

↓

Publicly verifiable (PV) non-interactive argument systems

Message authentication codes

↓

Signatures

Applications of odiO

***odiO**-based structure-preserving compilers*

Designated verifier (DV) non-interactive argument systems

↓

Publicly verifiable (PV) non-interactive argument systems

Message authentication codes

↓

Signatures

IV-based symmetric key encryption

↓

Public key encryption

Applications of odiO

***odiO**-based structure-preserving compilers*

Designated verifier (DV) non-interactive argument systems

↓

Publicly verifiable (PV) non-interactive argument systems

Message authentication codes

↓

Signatures

IV-based symmetric key encryption

↓

Public key encryption

IV-based SKE

$\mathsf{Enc}(k, m; iv) = (c, iv)$

Applications of odiO

odiO-*based structure-preserving compilers*

Designated verifier (DV) non-interactive argument systems

↓

Publicly verifiable (PV) non-interactive argument systems

Message authentication codes

↓

Signatures

IV-based symmetric key encryption

↓

Public key encryption

IV-based SKE

$\mathsf{Enc}(k, m; iv) = (c, iv)$

DV/PV non-interactive argument system

DV/PV non-interactive argument system

Designated Verifier (DV)

DV/PV non-interactive argument system

Designated Verifier (DV)

Syntax.

- Setup$(1^\lambda, \mathscr{R}) \to (crs, vk)$
- Prove$(crs, x, \omega) \to \pi$
- Verify$(vk, x, \pi) \to b$

DV/PV non-interactive argument system

Designated Verifier (DV)

Syntax.

- $\mathsf{Setup}(1^\lambda, \mathscr{R}) \to (crs, vk)$
- $\mathsf{Prove}(crs, x, \omega) \to \pi$
- $\mathsf{Verify}(vk, x, \pi) \to b$

Selective Soundness.

For every $x \notin \mathscr{L}$, every PPT adversary A, we have:

$$\mathbb{P}\left[\mathsf{Verify}(vk, x, \pi) = 1 \;\middle|\; \begin{array}{r}(crs, vk) \leftarrow \mathsf{Setup}(1^\lambda, \mathscr{R}) \\ \pi \leftarrow \mathsf{A}^{\mathsf{Verify}(vk,\cdot,\cdot)}(1^\lambda, crs, x)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

DV/PV non-interactive argument system

Designated Verifier (DV)

Syntax.

- $\mathsf{Setup}(1^\lambda, \mathscr{R}) \to (crs, vk)$
- $\mathsf{Prove}(crs, x, \omega) \to \pi$
- $\mathsf{Verify}(vk, x, \pi) \to b$

Selective Soundness.

For every $x \notin \mathscr{L}$, every PPT adversary A, we have:

$$\mathbb{P}\left[\mathsf{Verify}(vk, x, \pi) = 1 \;\middle|\; \begin{array}{r}(crs, vk) \leftarrow \mathsf{Setup}(1^\lambda, \mathscr{R}) \\ \pi \leftarrow \mathsf{A}^{\mathsf{Verify}(vk,\cdot,\cdot)}(1^\lambda, crs, x)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

Publicly Verifiable (PV)

Syntax.

- $\mathsf{Setup}(1^\lambda, \mathscr{R}) \to crs$
- $\mathsf{Prove}(crs, x, \omega) \to \pi$
- $\mathsf{Verify}(crs, x, \pi) \to b$

Selective Soundness.

For every $x \notin \mathscr{L}$, every PPT adversary A, we have:

$$\mathbb{P}\left[\mathsf{Verify}(vk, x, \pi) = 1 \;\middle|\; \begin{array}{r}(crs, vk) \leftarrow \mathsf{Setup}(1^\lambda, \mathscr{R}) \\ \pi \leftarrow \mathsf{A}(1^\lambda, crs, x)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

odiO-based Compiler

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*}$

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*}$

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*}$

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Sampler

$$\mathsf{S}_{x^*}(1^\lambda) \rightarrow (C_0 = C^{\mathsf{ver}}_{vk^*}, C_1 = C^{\mathsf{ver}}_{vk^*,x^*}, \alpha = crs^*)$$

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*}$

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Sampler

$$\mathsf{S}_{x^*}(1^\lambda) \rightarrow (C_0 = C^{\mathsf{ver}}_{vk^*}, C_1 = C^{\mathsf{ver}}_{vk^*,x^*}, \alpha = crs^*)$$

First Step: $\forall x^* \notin \mathscr{L}$, S_{x^*} is an **odiO**-sampler.

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*}$

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Sampler

$$\mathsf{S}_{x^*}(1^\lambda) \to (C_0 = C^{\mathsf{ver}}_{vk^*}, C_1 = C^{\mathsf{ver}}_{vk^*,x^*}, \alpha = crs^*)$$

First Step: $\forall x^* \notin \mathscr{L}$, S_{x^*} is an **odiO**-sampler.

Otherwise, the underlying (DV) Π^* is not sound.

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*}$

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Sampler

$$\mathsf{S}_{x^*}(1^\lambda) \rightarrow (C_0 = C^{\mathsf{ver}}_{vk^*}, C_1 = C^{\mathsf{ver}}_{vk^*,x^*}, \alpha = crs^*)$$

First Step: $\forall x^* \notin \mathscr{L}$, S_{x^*} is an **odiO**-sampler.

Otherwise, the underlying (DV) Π^* is not sound.

Here, we need oracles.

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*}$

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Sampler

$$\mathsf{S}_{x^*}(1^\lambda) \to (C_0 = C^{\mathsf{ver}}_{vk^*}, C_1 = C^{\mathsf{ver}}_{vk^*,x^*}, \alpha = crs^*)$$

First Step: $\forall x^* \notin \mathscr{L}$, S_{x^*} is an **odiO**-sampler.

Second Step: Replace $C^{\mathsf{ver}}_{vk^*}$ with $C^{\mathsf{ver}}_{vk^*,x^*}$.

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*}$

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Retu… erif… $vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Sampler

$\mathsf{S}_{x^*}(1^\lambda) \rightarrow (C_0 = C^{\mathsf{ver}}_{vk^*}, C_1 = C^{\mathsf{ver}}_{vk^*,x^*}, \alpha = crs^*)$

First Step: $\forall x^* \notin \mathscr{L}$, S_{x^*} is an **odiO**-sampler.

Second Step: Replace $C^{\mathsf{ver}}_{vk^*}$ with $C^{\mathsf{ver}}_{vk^*,x^*}$.

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. ~~$\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$~~
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*,x^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

~~Circuit $C^{\mathsf{ver}}_{vk^*}$~~

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Retu… erif… $vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Sampler

$$\mathsf{S}_{x^*}(1^\lambda) \to (C_0 = C^{\mathsf{ver}}_{vk^*}, C_1 = C^{\mathsf{ver}}_{vk^*,x^*}, \alpha = crs^*)$$

First Step: $\forall x^* \notin \mathscr{L}$, S_{x^*} is an **odiO**-sampler.

Second Step: Replace $C^{\mathsf{ver}}_{vk^*}$ with $C^{\mathsf{ver}}_{vk^*,x^*}$.

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. ~~$\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$~~
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*,x^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*}$

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Retu… erif… $vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Sampler

$$\mathsf{S}_{x^*}(1^\lambda) \to (C_0 = C^{\mathsf{ver}}_{vk^*}, C_1 = C^{\mathsf{ver}}_{vk^*,x^*}, \alpha = crs^*)$$

First Step: $\forall x^* \notin \mathscr{L}$, S_{x^*} is an **odiO**-sampler.

Second Step: Replace $C^{\mathsf{ver}}_{vk^*}$ with $C^{\mathsf{ver}}_{vk^*,x^*}$.

Otherwise, we can distinguish $\mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$ and $\mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*,x^*})$.

odiO-based Compiler

Compiler

Setup$(1^\lambda, \mathscr{R})$:

1. $(crs^*, vk^*) \leftarrow \mathsf{Setup}^*(1^\lambda, \mathscr{R})$
2. ~~$\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*})$~~
2. $\widetilde{C} \leftarrow \mathsf{Obf}(1^\lambda, C^{\mathsf{ver}}_{vk^*,x^*})$
3. Return $crs = (crs^*, \widetilde{C})$

Prove(crs, x, ω) :

1. Return $\pi \leftarrow \mathsf{Prove}(crs^*, x, \omega)$

Verify(crs, x, π) :

1. Return $\widetilde{C}(x, \pi)$

~~Circuit $C^{\mathsf{ver}}_{vk^*}$~~

$C^{\mathsf{ver}}_{vk^*}(x, \pi)$:

1. Retu… erif… $vk^*, x, \pi)$

Circuit $C^{\mathsf{ver}}_{vk^*,x^*}$

$C^{\mathsf{ver}}_{vk^*,x^*}(x, \pi)$:

1. If $x = x^*$, return 0.
2. Return $\mathsf{Verify}^*(vk^*, x, \pi)$

Sampler

$$\mathsf{S}_{x^*}(1^\lambda) \rightarrow (C_0 = C^{\mathsf{ver}}_{vk^*}, C_1 = C^{\mathsf{ver}}_{vk^*,x^*}, \alpha = crs^*)$$

First Step: $\forall x^* \notin \mathscr{L}$, S_{x^*} is an **odiO**-sampler.

Second Step: Replace $C^{\mathsf{ver}}_{vk^*}$ with $C^{\mathsf{ver}}_{vk^*,x^*}$.

Third Step: A's advantage is 0.

Other Results

Other Results

***oiO**-based structure-preserving compilers*

Key Indistinguishable symmetric key encryption

↓

Public key encryption

Other Results

***oiO**-based structure-preserving compilers*

Key Indistinguishable symmetric key encryption

↓

Public key encryption

General Purpose **odiO/oiO**-obfuscators do not exist (unconditionally)

There exists an **odiO/oiO**-sampler that cannot be **odiO/oiO**-obfuscated.

Other Results

***oiO**-based structure-preserving compilers*

Key Indistinguishable symmetric key encryption

↓

Public key encryption

General Purpose **odiO/oiO**-obfuscators do not exist (unconditionally)

There exists an **odiO/oiO**-sampler that cannot be **odiO/oiO**-obfuscated.

Key Ind. SKE → PKE is impossible (unconditionally)

Does not exist an obfuscator Obf able to implement this compiler.

Applications of odiO

***odiO**-based structure-preserving compilers*

Designated verifier (DV) non-interactive argument systems

↓

Publicly verifiable (PV) non-interactive argument systems

Message authentication codes

↓

Signatures

IV-based symmetric key encryption

↓

Public key encryption

IV-based SKE

$\mathsf{Enc}(k, m; iv) = (c, iv)$

Applications of odiO

***odiO**-based structure-preserving compilers*

Designated verifier (DV) non-interactive argument systems

↓

Publicly verifiable (PV) non-interactive argument systems

Message authentication codes

↓

Signatures

IV-based symmetric key encryption

↓

Public key encryption

+

Puncturable PRF

IV-based SKE

$\mathsf{Enc}(k, m; iv) = (c, iv)$

Thank You!