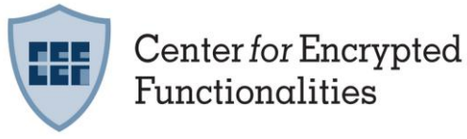
# Robust Transforming Combiners from iO to Functional Encryption

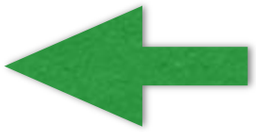




Prabhanjan Ananth
Aayush Jain
Amit Sahai

# Since 2013…

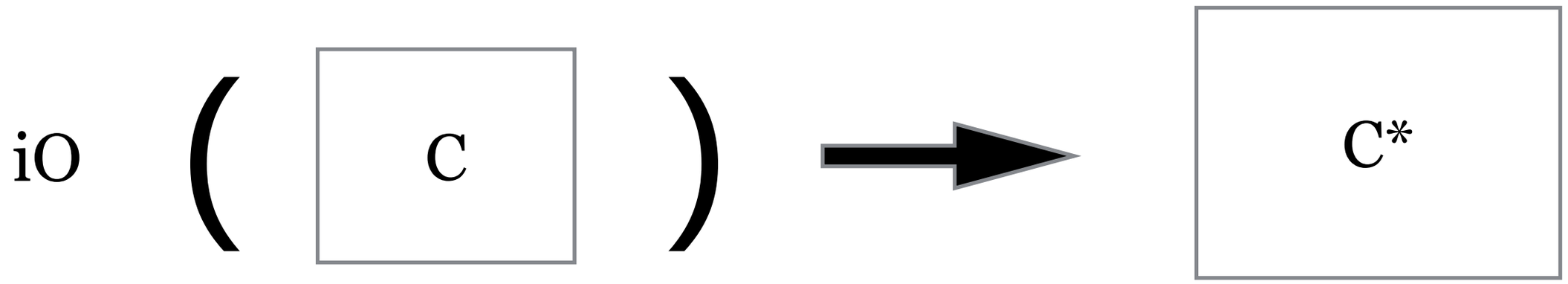
- Two-Round (Adaptive) Multi-Party Computation
- Instantiating Random Oracles
- Non-Interactive Multi-party Key Exchange
- Impossibility Results
- Theoretical Results (such as PPAD Hardness)
- Constant-Round Concurrent Zero Knowledge
- Separation Results for Circular Security
- Succinct Randomized Encodings
- Watermarking
- Patching

• • •

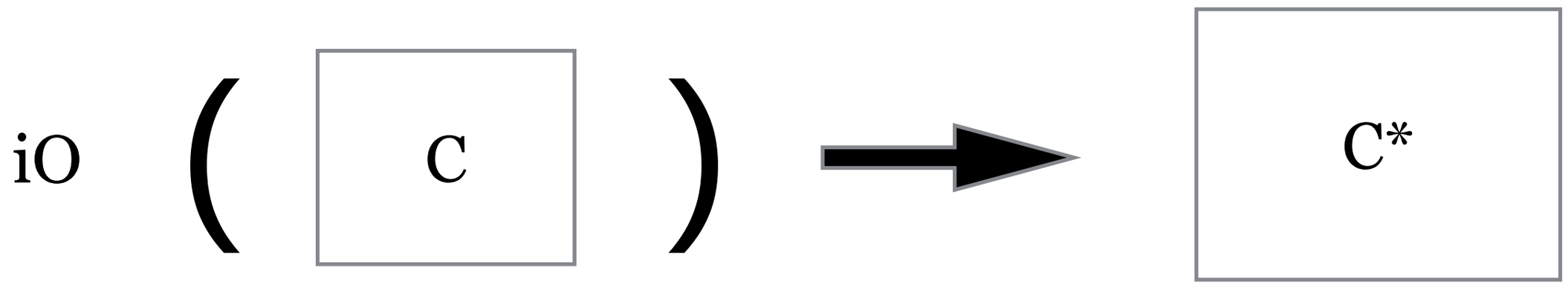


Indistinguishability Obfuscation (iO)/Functional Encryption

# What is iO?



# What is iO?




*Correctness*: for all x, $C^*(x) = C(x)$

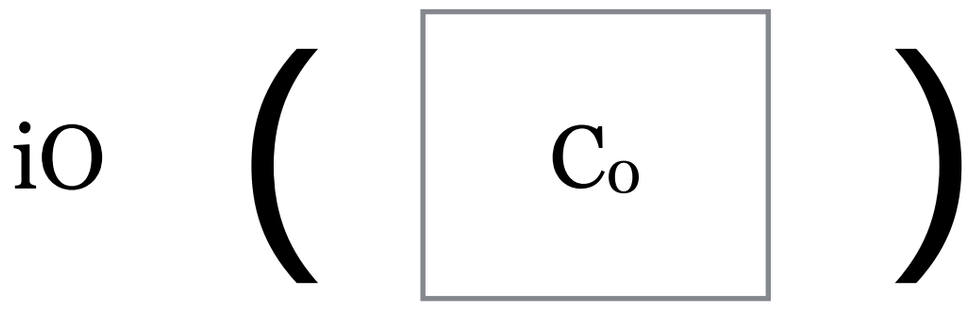
# What is iO?

$C_0$

≡

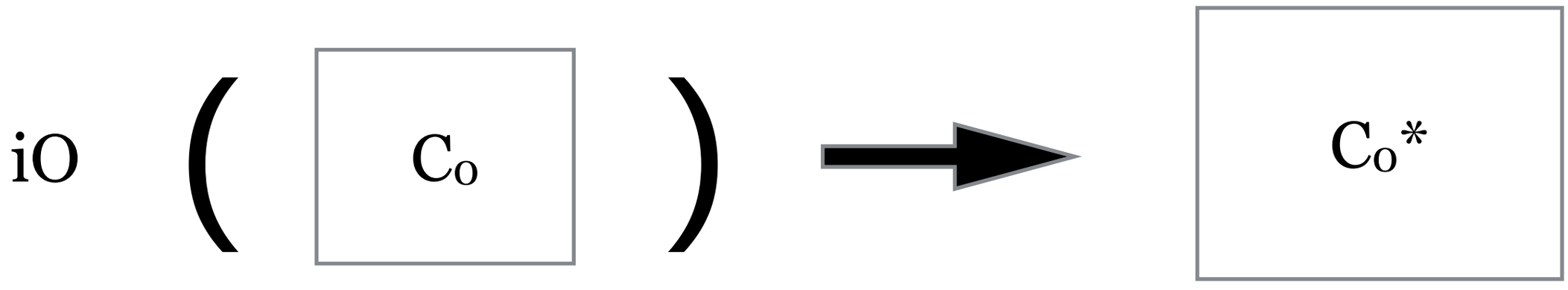
$C_1$

# What is iO?

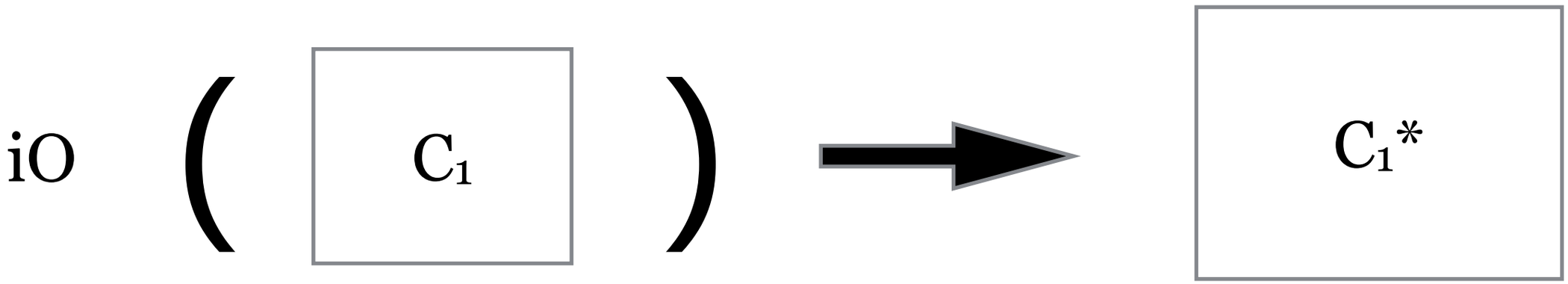



≡

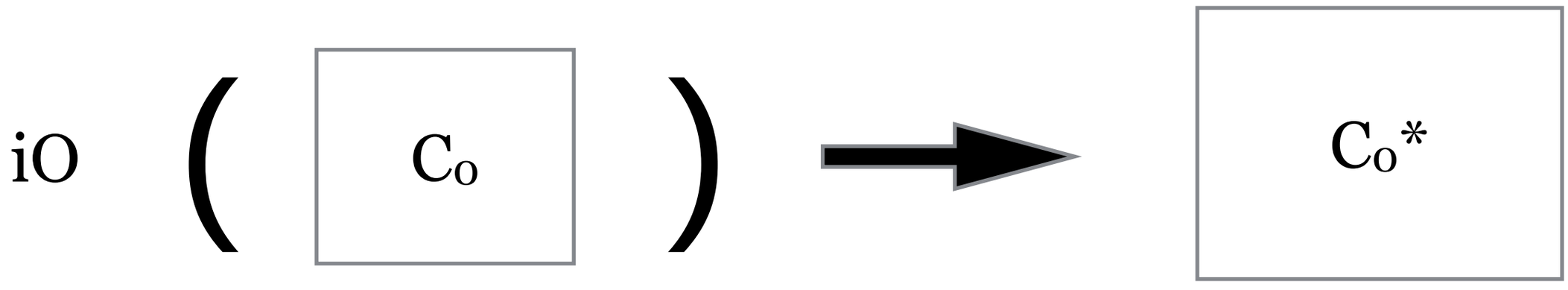
# What is iO?



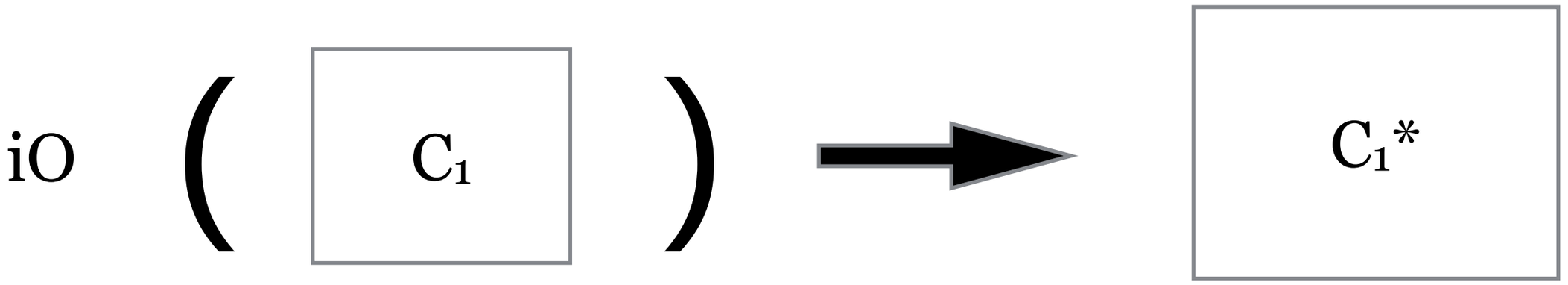

≡



# What is iO?




$\equiv$ *Security*: $\approx_c$



# Functional Encryption

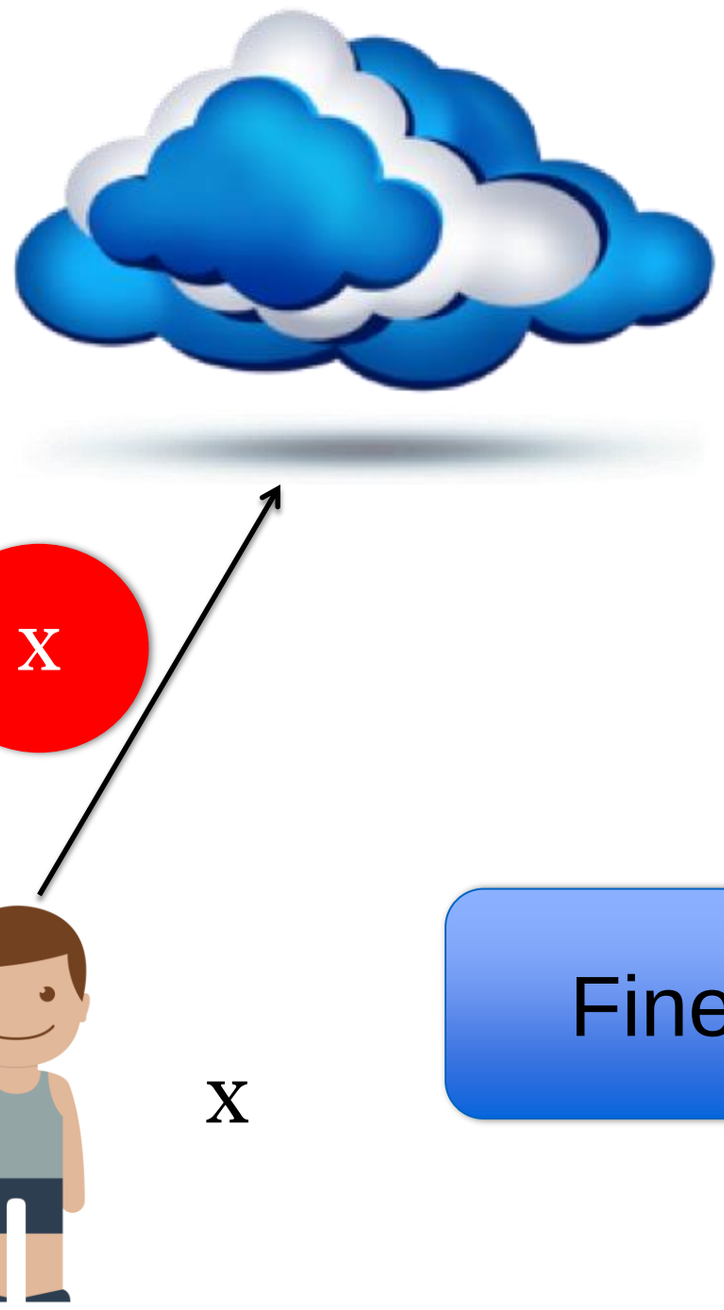
[SW'05,GGHRSW13]

x

Fine Grained Access to Private Data

# Functional Encryption

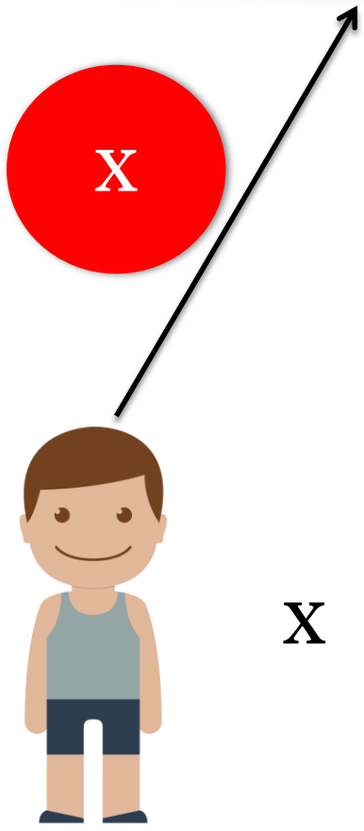
[SW'05,GGHRSW13]




Fine Grained Access to Private Data

# Functional Encryption

[SW'05,GGHRSW13]

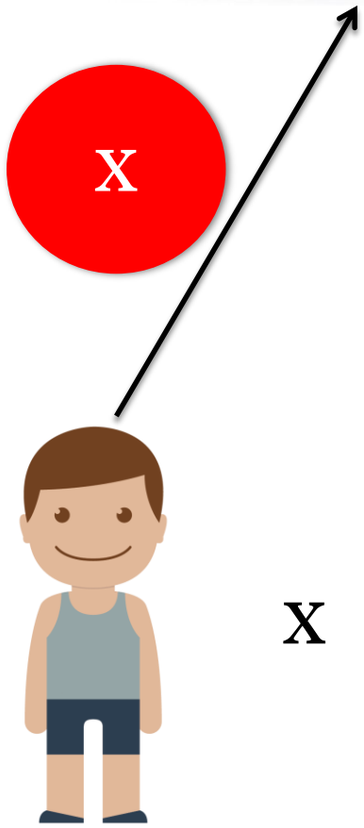



Fine Grained Access to Private Data

# Functional Encryption

[SW'05,GGHRSW13]

MSK


x

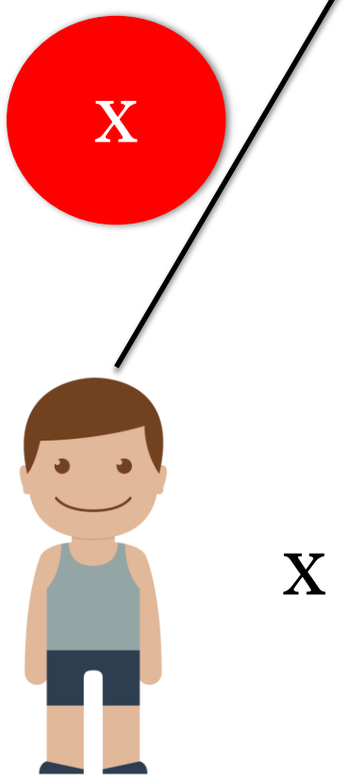
Fine Grained Access to Private Data

# Functional Encryption

[SW'05,GGHRSW13]

f

MSK


x

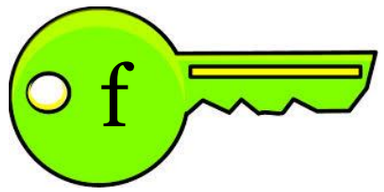
Fine Grained Access to Private Data

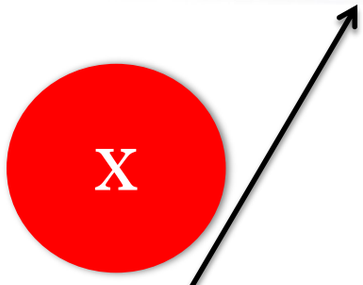
# Functional Encryption

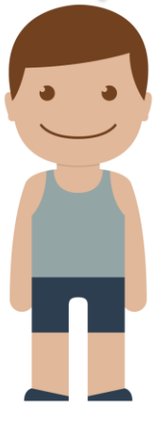
[SW'05,GGHRSW13]

MSK

f

x

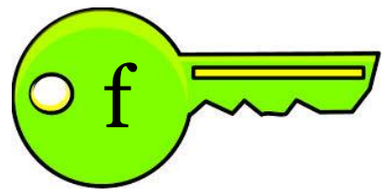
Fine Grained Access to Private Data

# Functional Encryption

[SW'05,GGHRSW13]

MSK

f


x

$Dec(\ f\ ,\ x\ ) = f(x)$

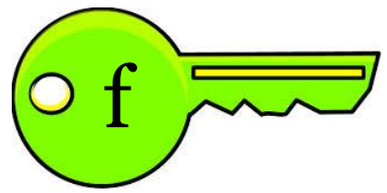
Fine Grained Access to Private Data

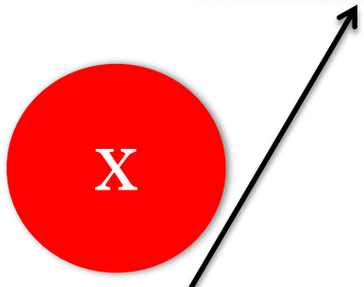
# Functional Encryption

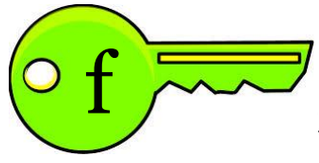
[SW'05,GGHRSW13]

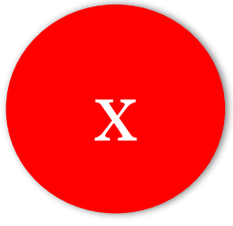
MSK

f

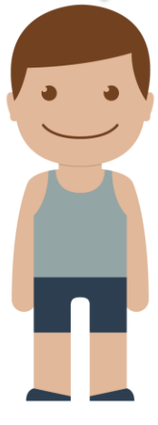




Dec( 

, 

 ) = f(x)



x

$SK_f$ should not allow adversary to compute anything other than f(x) !

# Known Constructions?

[GGHRSW’13, BGKPS’14, Zim’15, GLSW’15, AB’15, GMMSSZ’16, LV’16, L’16, AS’17, LT’17….]

Are all candidates of iO broken?

NO!

Are all candidates of iO broken?

**NO!**

We have several unbroken iO candidates,
including with proofs of security in various models.

# Our Goal

Find a iO candidate that is secure
*even if **only** one of the candidates is secure.*

# Our Goal

Find a iO candidate that is secure
*even if **only** one of the candidates is secure.*

Problem Statement:
Given any set of iO candidates, find a candidate that is secure
even if **only** one of the candidates is secure.

# Our Goal

Find a iO candidate that is secure
*even if **only** one of the candidates is secure.*

Problem Statement:
Given any set of iO candidates, find a candidate that is secure
even if **only** one of the candidates is secure.

**iO combiner**

# Our Goal

Find a iO candidate that is secure
*even if **only** one of the candidates is secure*.

Problem Statement:
Given any set of iO candidates, find a candidate that is secure
even if **only** one of the candidates is secure.

**Robust iO combiner:**

*In fact we only require the secure candidate to be correct*
*All other candidates can violate correctness*
**[AJNSY16, FHNS16]**

# Robust iO Combiners

*Let* **P** = ($P_1$, …, $P_n$) *be any n iO candidates*

# Robust iO Combiners

*Let* **P** = ($P_1$, …, $P_n$) *be any n iO candidates*

- RCiO.Obf( **P** , C ) outputs C*.

# Robust iO Combiners

*Let* **P** = ($P_1$, …, $P_n$) *be any n iO candidates*

- RCiO.Obf( **P** , C ) outputs C*.
- RCiO.Eval( **P** , C*, x ) outputs y.

# Robust iO Combiners

*Let* **P** = ($P_1$, …, $P_n$) *be any n iO candidates*

- RCiO.Obf( **P** , C ) outputs C*.
- RCiO.Eval( **P** , C*, x ) outputs y.

If there exists i in [n] such that $P_i$ is correct and secure:

# Robust iO Combiners

*Let* **P** = ($P_1$, …, $P_n$) *be any n iO candidates*

- RCiO.Obf( **P** , C ) outputs C*.
- RCiO.Eval( **P** , C*, x ) outputs y.

If there exists i in [n] such that $P_i$ is correct and secure:

**Correctness:** y = C(x)

# Robust iO Combiners

*Let* **P** = ($P_1$, …, $P_n$) *be any n iO candidates*

- RCiO.Obf( **P** , C ) outputs C*.
- RCiO.Eval( **P** , C*, x ) outputs y.

*If there exists i in [n] such that* $P_i$ is correct and secure *:*

**Security:** *If $C_0$ is equivalent to $C_1$,*

RCiO.Obf( **P** , $C_0$) $\approx_c$ RCiO.Obf( **P** , $C_1$)

# Implications

Robust iO combiners imply **universal iO [AJNSY’16]**

# Implications

Robust iO combiners imply **universal iO [AJNSY'16]**

**Universal iO:**

A scheme P is a universal iO scheme

if *iO exists* then *P is a secure iO scheme*

# Previous Work

# Previous Work

- **AJNSY16** gave candidate construction of a robust combiner from DDH/LWE.
- Required one candidate to be sub-exponentially secure.
- **FHNS16** considers the case of combining unconditionally.

# Previous Work

- **AJNSY16** gave candidate construction of a robust combiner from DDH/LWE.
- Required one candidate to be sub-exponentially secure.
- **FHNS16** considers the case of combining unconditionally.

## Questions?

- Can we achieve some applications of iO if the secure candidate is polynomially secure?
- Can we weaken the assumptions to rely on only one-way functions?

# This Work

**Theorem 1 (Combiner -> Robust Combiner):**
*Given:*

- *An iO Combiner* **AND**
- *One-way function f,*

*we construct a robust iO combiner*

# This Work

**Theorem 1 (Combiner -> Robust Combiner):**
*Given:*

- *An iO Combiner* **AND**
- *One-way function f,*

*we construct a robust iO combiner*

Previously, as observed in AJNSY'16 and BV'15, this result required sub-exponential DDH/LWE and the underlying candidate to be sub-exponentially secure

# This Work

# This Work

**Theorem 2:** *Given:*

- *N correct iO Candidates (with one secure)*

  ***AND***

- *Any one-way function F,*

*we construct a compact FE scheme with complexity poly(k,N) and polynomial security loss.*

# This Work

**Theorem 2:** *Given:*

- *N correct iO Candidates (with one secure)*

  ***AND***

- *Any one-way function F,*

*we construct a compact FE scheme with complexity poly(k,N) and polynomial security loss.*

**Corollary [AJ15,BV15]:** *There exists (sub-exponential) universal iO if sub-exponential one-way functions exist.*

# This Work

**Theorem 2:** *Given:*

- *N correct iO Candidates (with one secure)*

  ***AND***

- *Any one-way function F,*

*we construct a compact FE scheme with complexity poly(k,N) and polynomial security loss.*

Transforming Combiners

**Corollary [AJ15,BV15]:** *There exists (sub-exponential) universal iO if sub-exponential one-way functions exist.*

# Transforming Combiners

Given N candidates of primitive $A=(A_1,..,A_N)$,
such that one $A_i$ is secure and correct,
construct secure primitive B
with efficiency polynomial in N.

# Transforming Combiners

Given N candidates of primitive $A=(A_1,..,A_N)$,
such that one $A_i$ is secure and correct,
construct secure primitive B
with efficiency polynomial in N.

We show:
There exists a transforming robust combiner from iO to Functional Encryption.
This also yields any primitive implied by FE (such as NIKE. [GPSZ17])

# Technical Overview

# Combiner to Robust Combiner: Idea 1

# Combiner to Robust Combiner: Idea 1

- For each obfuscation candidate P, construct modified candidate P' that "self-checks for correctness":

# Combiner to Robust Combiner: Idea 1

- For each obfuscation candidate P, construct modified candidate P' that "self-checks for correctness":

P'(C) works as follows:

# Combiner to Robust Combiner: Idea 1

- For each obfuscation candidate P, construct modified candidate P' that "self-checks for correctness":

P'(C) works as follows:

1. Compute P(C)=C*

# Combiner to Robust Combiner: Idea 1

- For each obfuscation candidate P, construct modified candidate P' that "self-checks for correctness":

P'(C) works as follows:

1. Compute P(C)=C*
2. Sample $x_1, x_2,..,x_L$, where $L = k^2$

# Combiner to Robust Combiner: Idea 1

- For each obfuscation candidate P, construct modified candidate P' that "self-checks for correctness":

P'(C) works as follows:

1. Compute P(C)=C*
2. Sample $x_1, x_2,..,x_L$, where $L = k^2$
3. Check if $C^*(x_i)=C(x_i)$ for all i

# Combiner to Robust Combiner: Idea 1

- For each obfuscation candidate P, construct modified candidate P' that "self-checks for correctness":

P'(C) works as follows:

1. Compute P(C)=C*
2. Sample $x_1, x_2,..,x_L$, where $L = k^2$
3. Check if $C^*(x_i)=C(x_i)$ for all i
4. If any check fails, output C, otherwise output C*

# Combiner to Robust Combiner: Idea 1

- For each obfuscation candidate P, construct modified candidate P' that "self-checks for correctness":

$$\Pr_{\{x,\text{coins}(P)\}}[C^*(x)=C(x)] \geq 1 - 1/k$$

P'(C) works as follows:

1. Compute P(C)=C*
2. Sample $x_1, x_2,..,x_L$, where $L = k^2$
3. Check if $C^*(x_i)=C(x_i)$ for all i
4. If any check fails, output C, otherwise output C*

# Combiner to Robust Combiner: Idea 1

- For each obfuscation candidate P, construct modified candidate P' that "self-checks for correctness":

$$\Pr_{\{x,coins(P)\}}[C^*(x)=C(x)] \geq 1 - 1/k$$

Secure candidate is unchanged as it is correct.

P'(C) works as follows:

1. Compute P(C)=C*
2. Sample $x_1, x_2,..,x_L$, where $L = k^2$
3. Check if $C^*(x_i)=C(x_i)$ for all i
4. If any check fails, output C, otherwise output C*

# Removing dependency on x: Idea 2

# Removing dependency on x: Idea 2

# Removing dependency on x: Idea 2

"Encrypt Inputs" [BV'15]

# Removing dependency on x: Idea 2

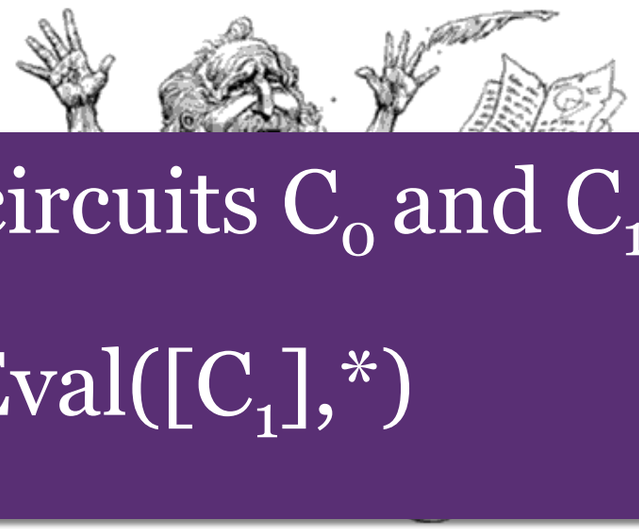
"Encrypt Inputs" [BV'15]

- Consider a "special" circuit garbling scheme with an additional property.

# Removing dependency on x: Idea 2

"Encrypt Inputs" [BV'15]

- Consider a "special" circuit garbling scheme with an additional property.



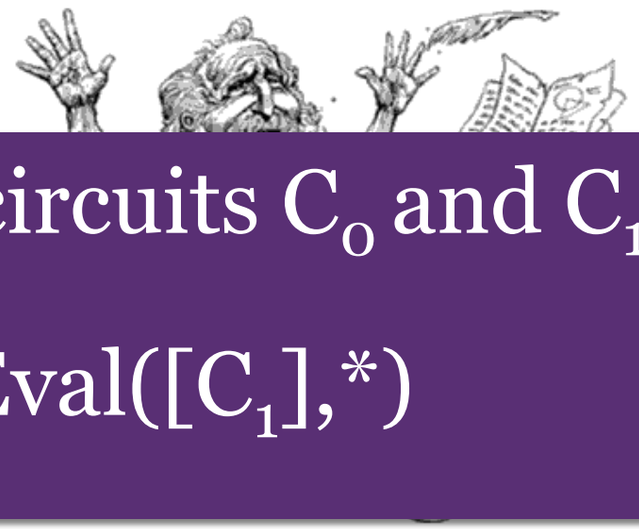
For any equivalent circuits $C_0$ and $C_1$

$\mathrm{Eval}([C_0],*) \cong \mathrm{Eval}([C_1],*)$

# Removing dependency on x: Idea 2

"Encrypt Inputs" [BV'15]

- Consider a "special" circuit garbling scheme with an additional property.



For any equivalent circuits $C_0$ and $C_1$

$\mathrm{Eval}([C_0],*) \cong \mathrm{Eval}([C_1],*)$

- Such garbled circuits can be constructed from one-way functions.

# Combining Ideas

# Combining Ideas

1. Use the modified obfuscator to obfuscate Eval([C],*)
2. Release the encoding key MSK to the evaluator.

# Combining Ideas

For any x,
$\Pr_{\{coins(P)\}}[C^*(x)=C(x)] \geq 1-2/k$

1. Use the modified obfuscator to obfuscate Eval([C],*)
2. Release the encoding key MSK to the evaluator.

# Combining Ideas

For any x,
$\Pr_{\{coins(P)\}}[C^*(x)=C(x)] \geq 1-2/k$

1. Use the modified obfuscator to obfuscate Eval([C],*)
2. Release the encoding key MSK to the evaluator.

Perform BPP Amplification to
get almost correctness

# Theorem 2: Combining iO

**IDEA:**

# Theorem 2: Combining iO

**IDEA:**

- No candidate should get the circuit in the clear.

# Theorem 2: Combining iO

**IDEA:**

- No candidate should get the circuit in the clear.
- Every candidate should get a secret share of circuit C

# Theorem 2: Combining iO

**IDEA:**

- No candidate should get the circuit in the clear.
- Every candidate should get a secret share of circuit C
- On every input x, the candidates “jointly compute” C

# Theorem 2: Combining iO

**IDEA:**

- No candidate should get the circuit in the clear.
- Every candidate should get a secret share of circuit C
- On every input x, the candidates "jointly compute" C

# Theorem 2: Combining iO

**IDEA:**

- No candidate should get the circuit in the clear.
- Every candidate should get a secret share of circuit C
- On every input x, the candidates "jointly compute" C

How to do this?

**Use MPC Techniques!**

# Approach of AJNSY’16

# Approach of AJNSY'16

- Let C be the circuit to be obfuscated.

# Approach of AJNSY'16

- Let C be the circuit to be obfuscated.
- Use a non-interactive MPC.

# Approach of AJNSY'16

- Let C be the circuit to be obfuscated.
- Use a non-interactive MPC.
- Secret share circuit C into $C_1,\ldots,C_N$. Treat $C_i$ as input to $P_i$.

# Approach of AJNSY'16

- Let C be the circuit to be obfuscated.
- Use a non-interactive MPC.
- Secret share circuit C into $C_1,\ldots,C_N$. Treat $C_i$ as input to $P_i$.
- Obfuscate the circuit containing $C_i$ and the pre-processed state using candidate $P_i$

# Approach of AJNSY'16

- Let C be the circuit to be obfuscated.
- Use a non-interactive MPC.
- Secret share circuit C into $C_1,\ldots,C_N$. Treat $C_i$ as input to $P_i$.
- Obfuscate the circuit containing $C_i$ and the pre-processed state using candidate $P_i$

MPC satisfying such properties are based on assumptions such as LWE/DDH [MW'16,BGI'17]

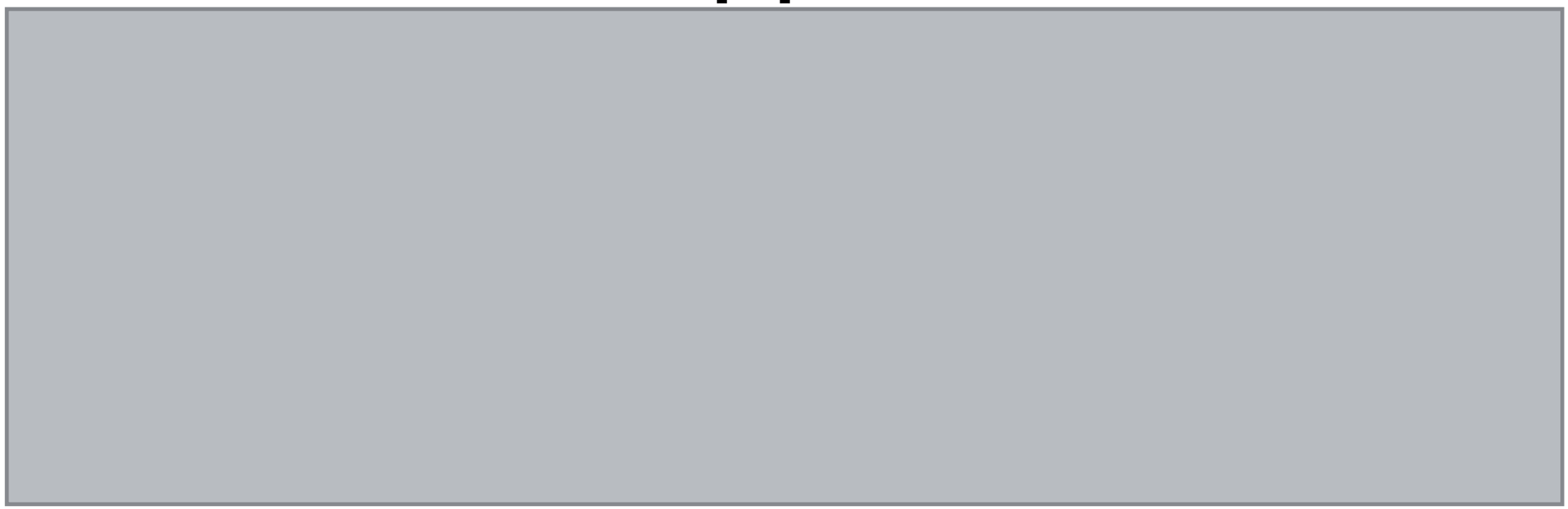
# Approach of AJNSY'16

- Let C be the circuit to be obfuscated.
- Use a non-interactive MPC.
- Secret share circuit C into $C_1,\ldots,C_N$. Treat $C_i$ as input to $P_i$.
- Obfuscate the circuit containing $C_i$ and the pre-processed state using candidate $P_i$

MPC satisfying such properties are based on assumptions such as LWE/DDH [MW'16,BGI'17]

Can we weaken assumptions by relying on interactive MPC?

# Our Approach



# Our Approach


THE FIRST IDEA.

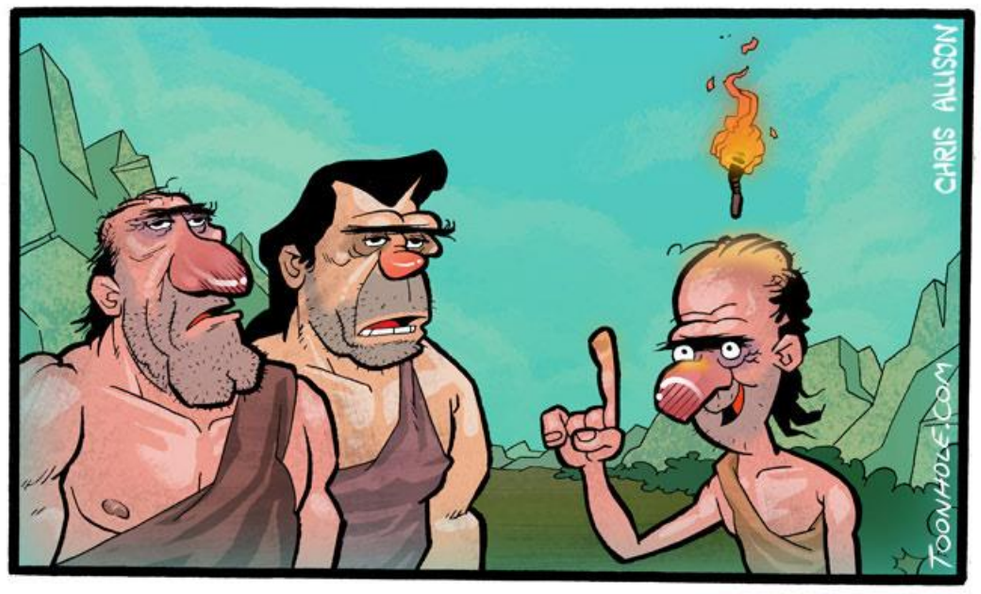
# Our Approach

- Secret share circuit to $(C_1,..,C_N)$ using additive secret sharing.


THE FIRST IDEA.

# Our Approach

- Secret share circuit to $(C_1,..,C_N)$ using additive secret sharing.
- Treat each candidate as a party in interactive MP Cprotocol.




THE FIRST IDEA.

# Our Approach

- Secret share circuit to $(C_1,..,C_N)$ using additive secret sharing.
- Treat each candidate as a party in interactive MP Cprotocol.
- Run the MPC protocol for $U(C_1+...+C_N, x)$ to learn $C(x)$


THE FIRST IDEA.

# How to evaluate MPC?

# How to evaluate MPC?

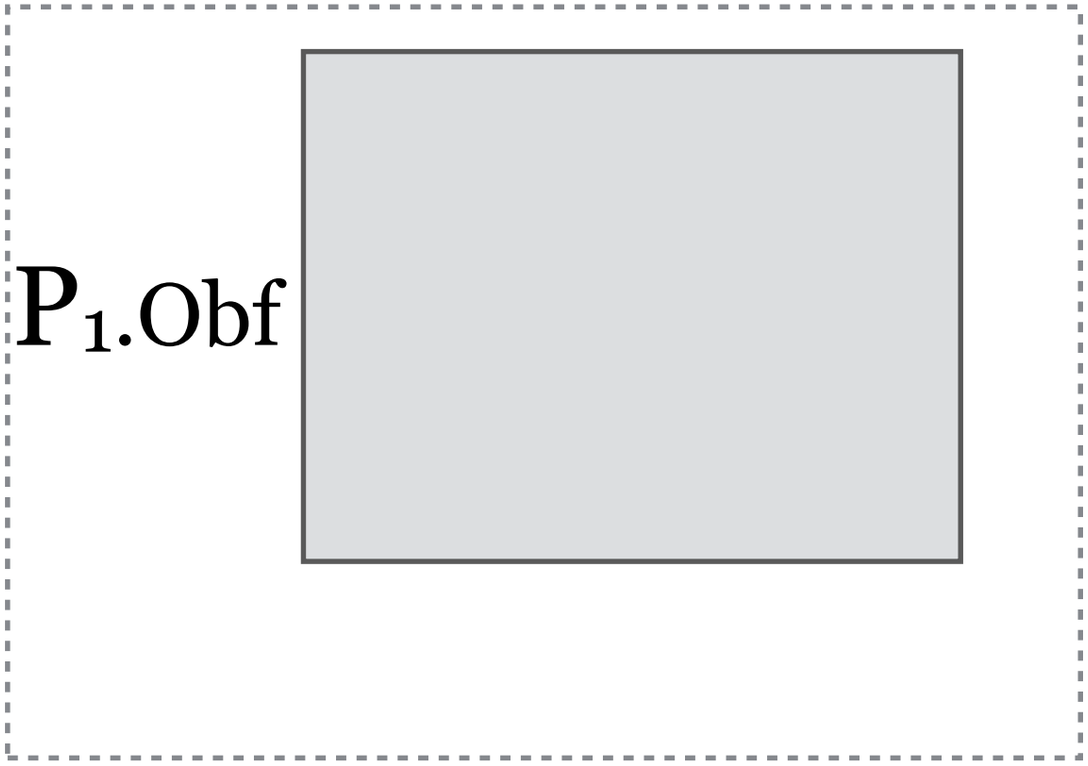
- Using candidate $P_i$ obfuscate NextMsg($C_i$, , *)

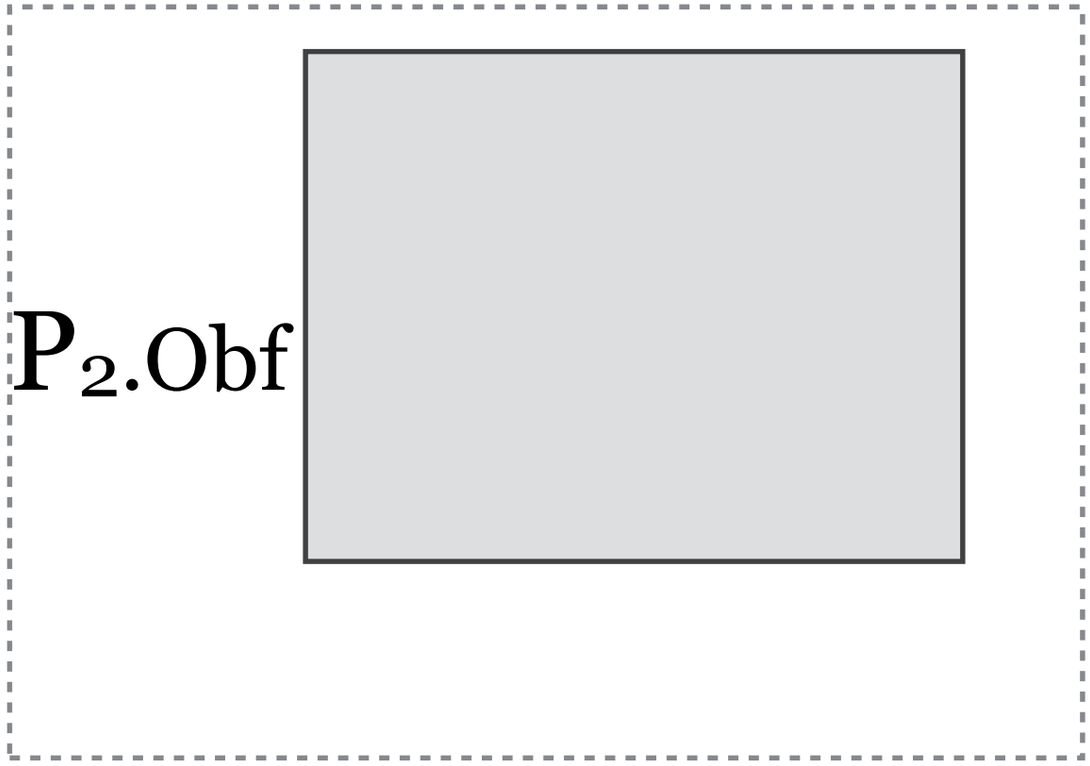
# How to evaluate MPC?

- Using candidate $P_i$ obfuscate NextMsg($C_{i,}$ , *)

# How to evaluate MPC?

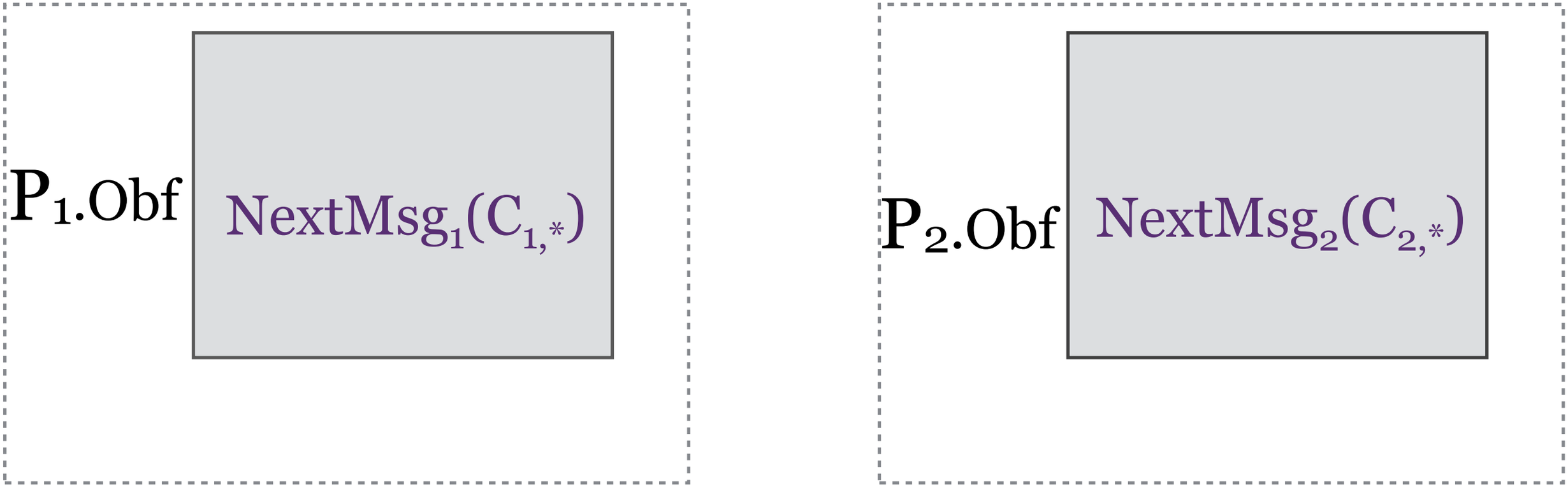
- Using candidate $P_i$ obfuscate NextMsg($C_i$, , *)



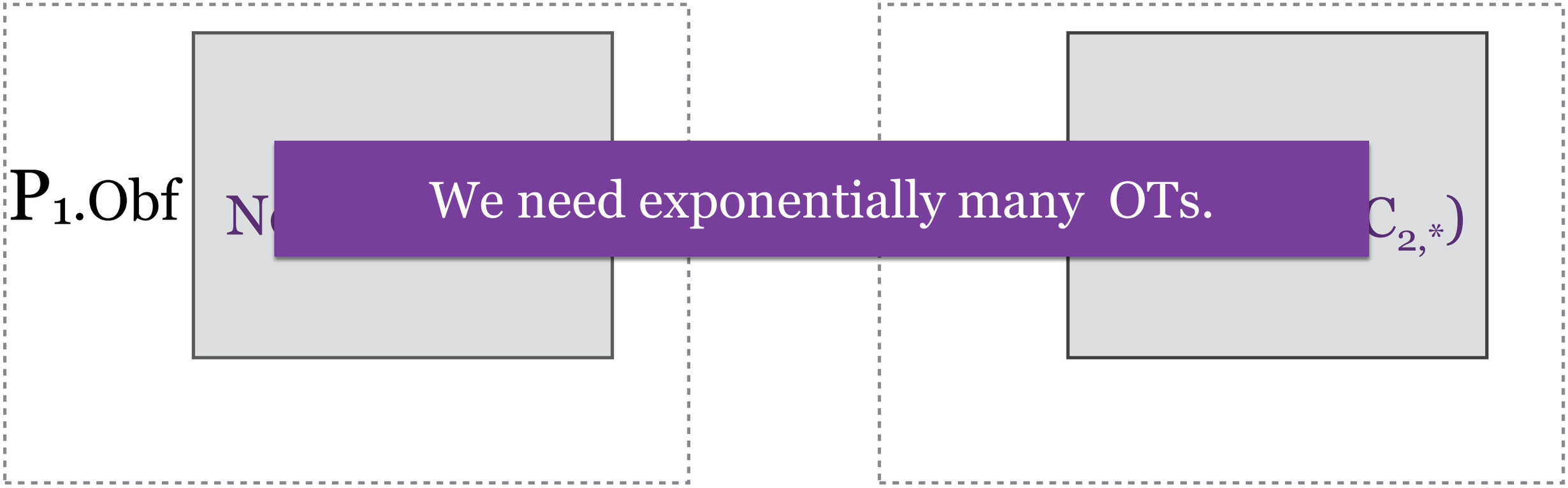
# How to evaluate MPC?

- Using candidate $P_i$ obfuscate $NextMsg(C_{i,}, *)$



# How to evaluate MPC?

- Using candidate $P_i$ obfuscate NextMsg($C_{i,}$ , *)



# (Random) OT

$P_1$

$P_2$

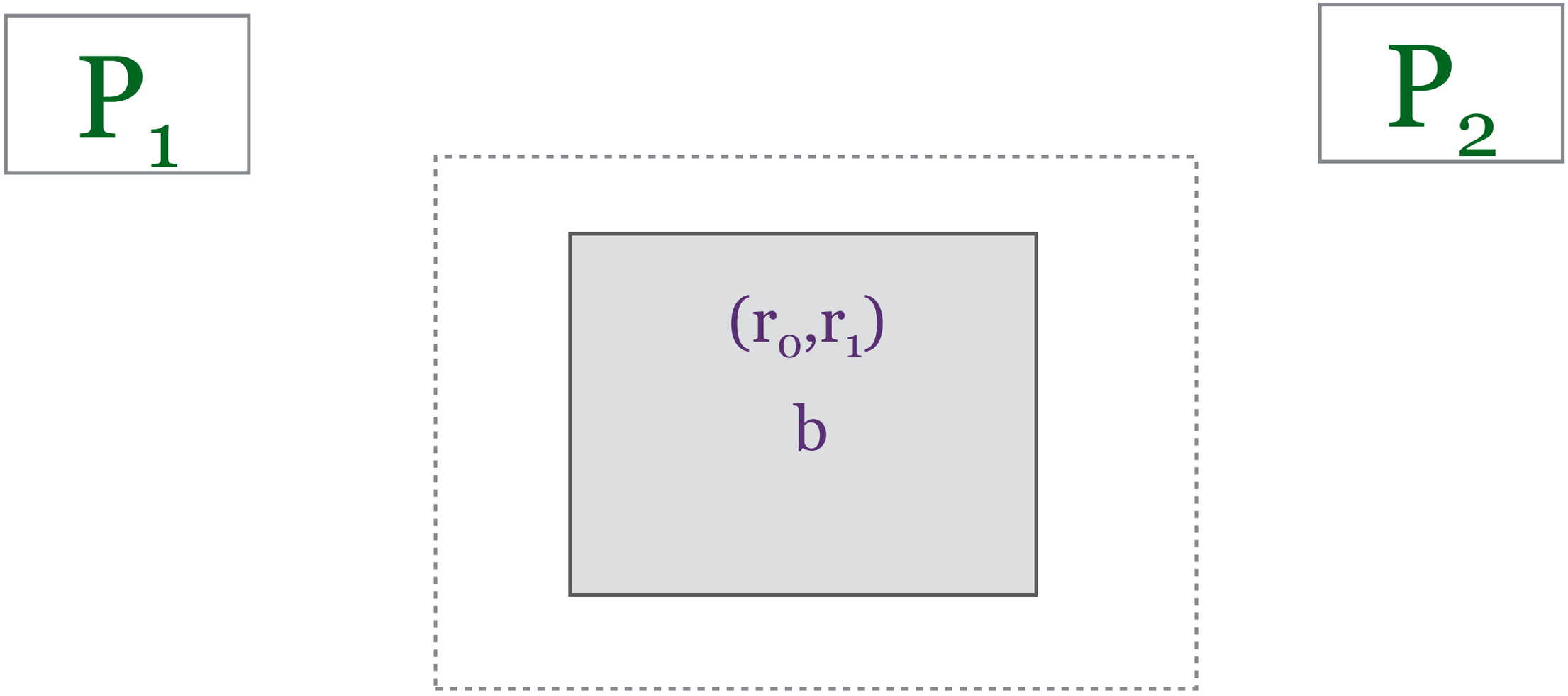
# (Random) OT

$P_1$

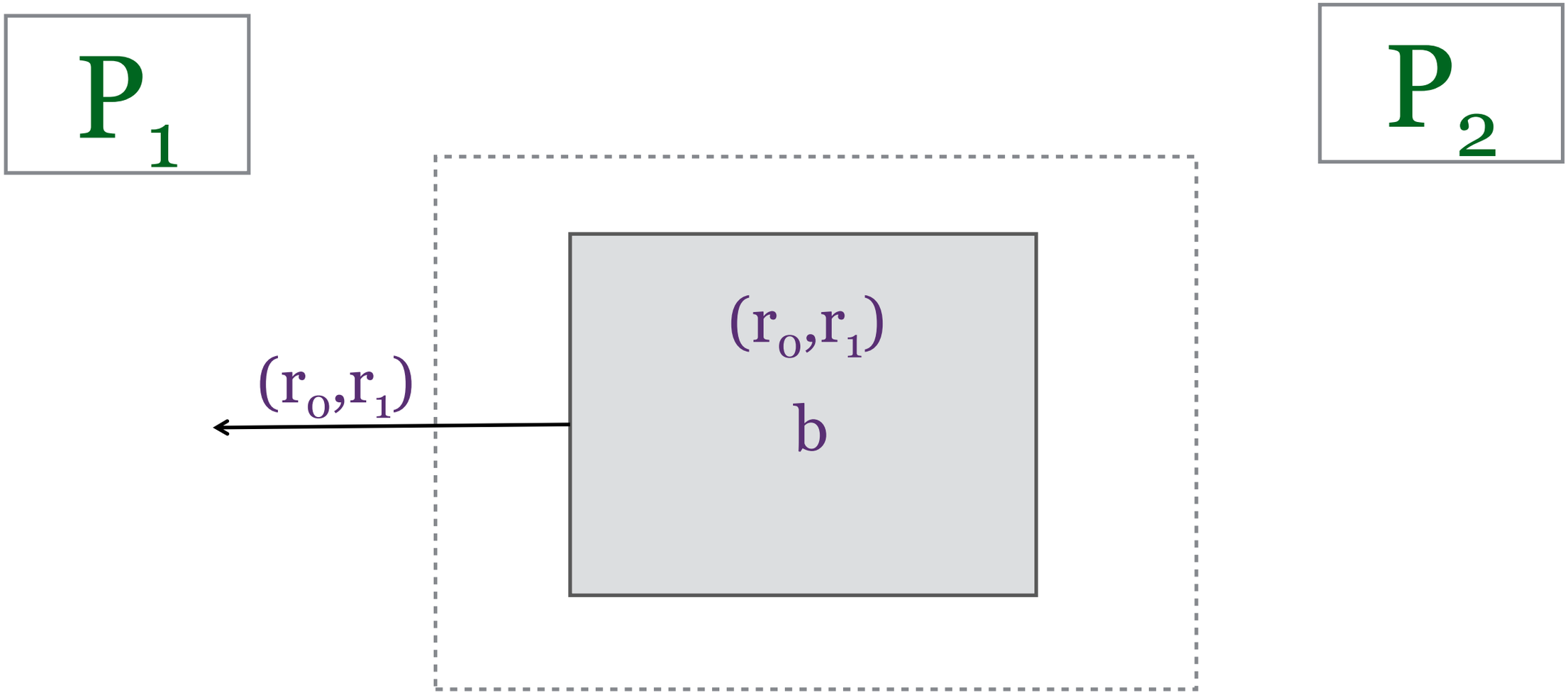
$P_2$

$(r_0, r_1)$

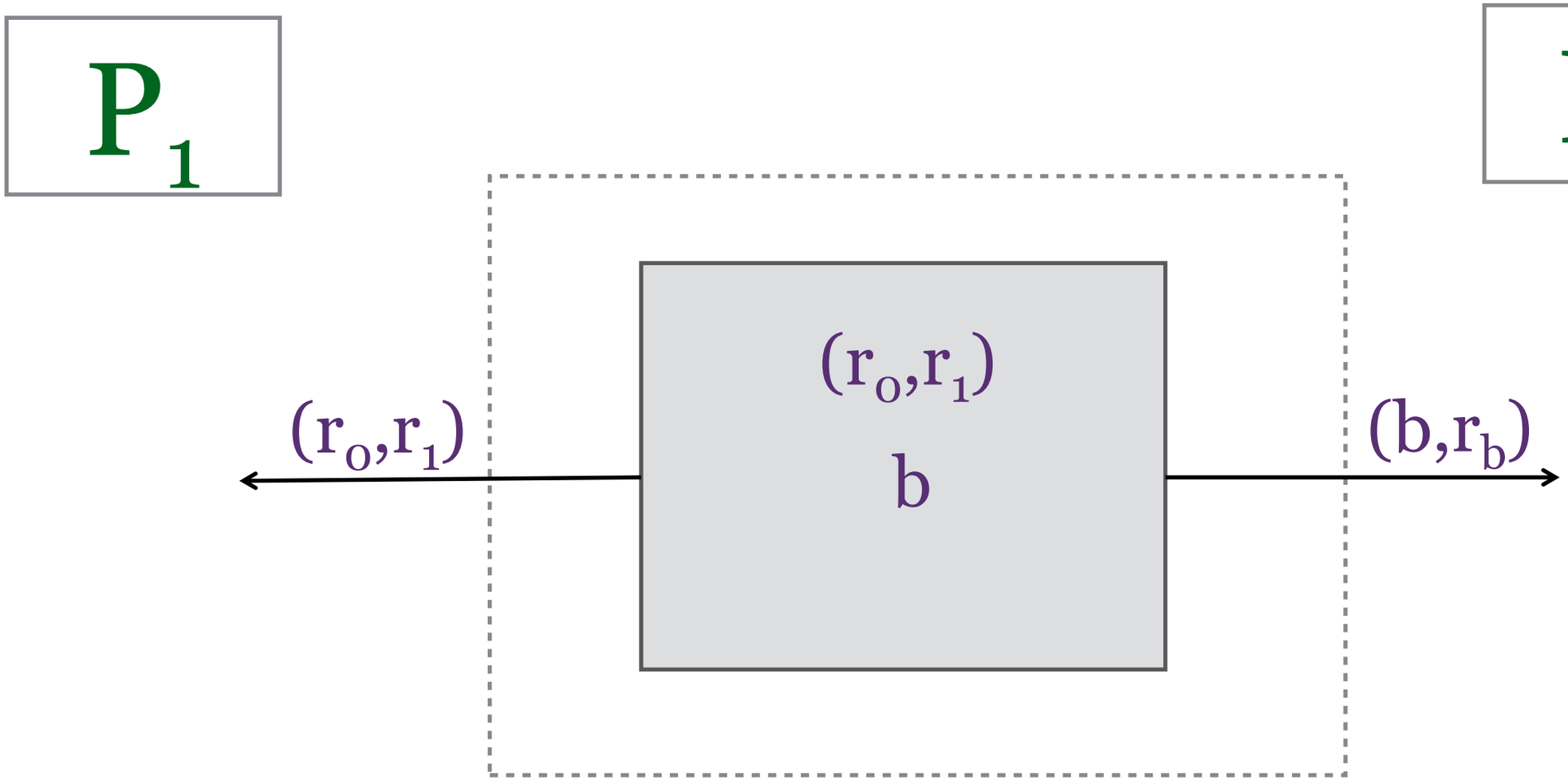
# (Random) OT



# (Random) OT



# (Random) OT



# How to Implement OT?

# How to Implement OT?

- Use any OT protocol? Assumptions are stronger.

# How to Implement OT?

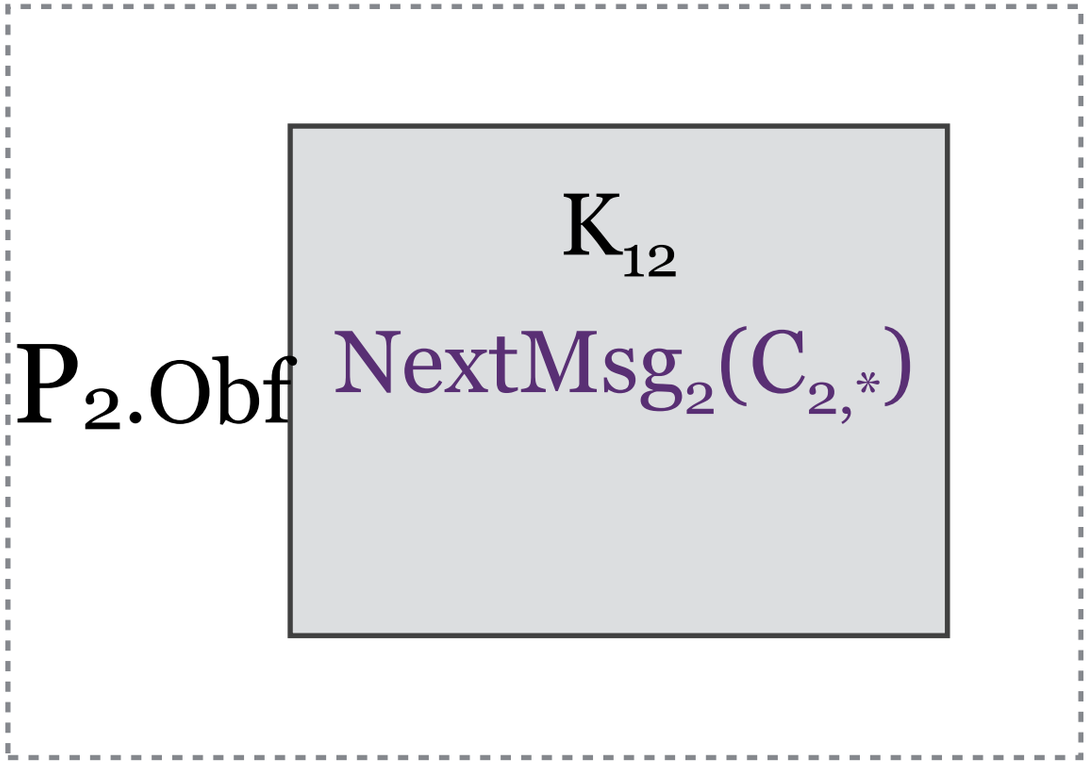
- Use any OT protocol? Assumptions are stronger.
- Pre-process random OTs. Exponential pre-processing required.

# How to Implement OT?

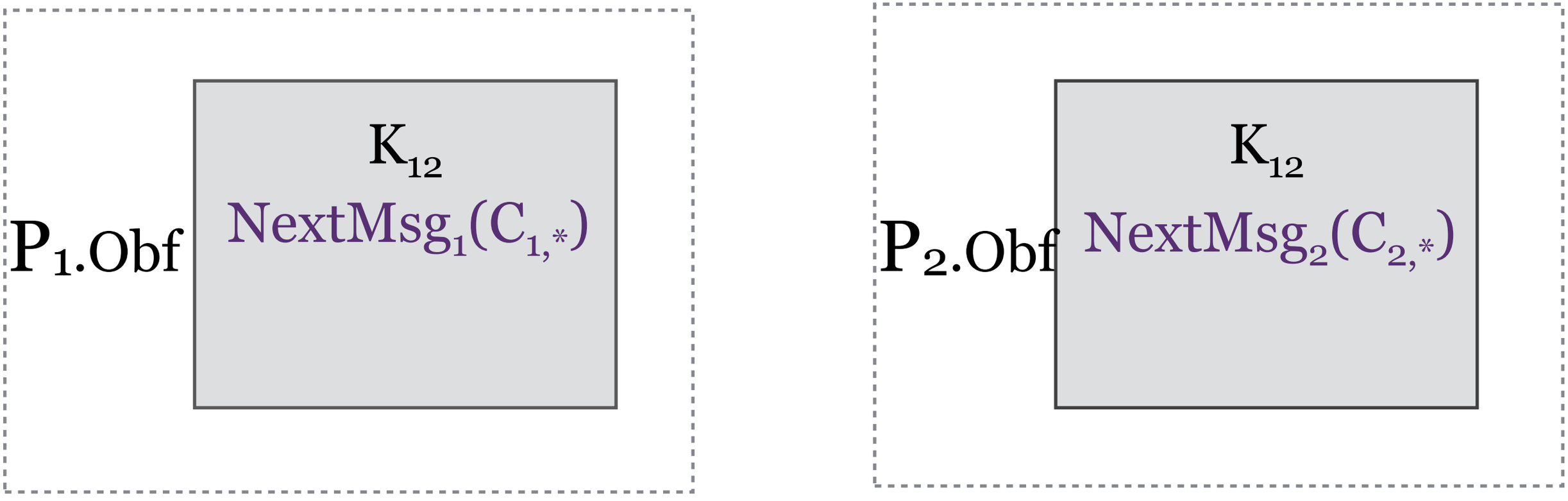
- Use any OT protocol? Assumptions are stronger.
- Pre-process random OTs. Exponential pre-processing required.
- Use PRF keys to generate OTs on the fly.

# Using PRF keys

# Using PRF keys

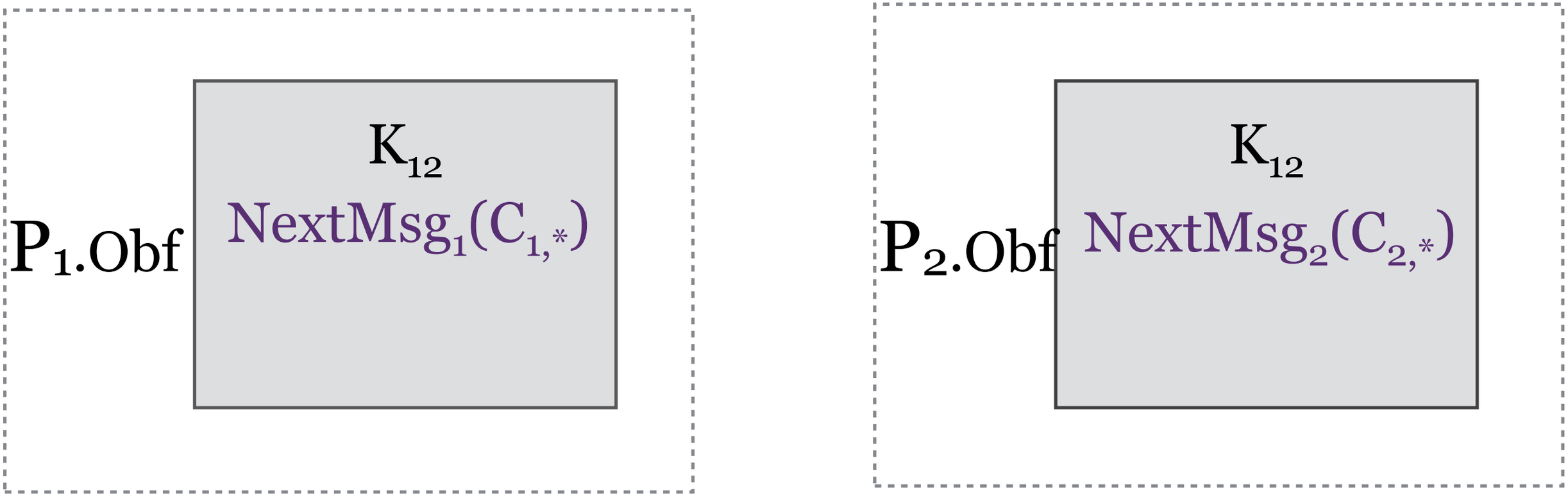


# Using PRF keys



# Using PRF keys

But the PRF key $K_{i,j}$ is obfuscated individually by both candidates $P_i$ and $P_j$



# Using PRF keys

But the PRF key $K_{i,j}$ is obfuscated individually by both candidates $P_i$ and $P_j$

$P_1$.Obf

$K_{12}$

$NextMsg_1(C_{1,*})$

$P_2$.Obf

$K_{12}$

$NextMsg_2(C_{2,*})$

# Using PRF keys

But the PRF key $K_{i,j}$ is obfuscated individually by both candidates $P_i$ and $P_j$

$P_1$.Obf

$K_{12}$ ✗

$NextMsg_1(C_{1,*})$

$P_2$.Obf

$K_{12}$

$NextMsg_2(C_{2,*})$

# Using PRF keys

But the PRF key $K_{i,j}$ is obfuscated individually by both candidates $P_i$ and $P_j$

$P_1$.Obf

$K_{12}$ ✗

$NextMsg_1(C_{1,*})$

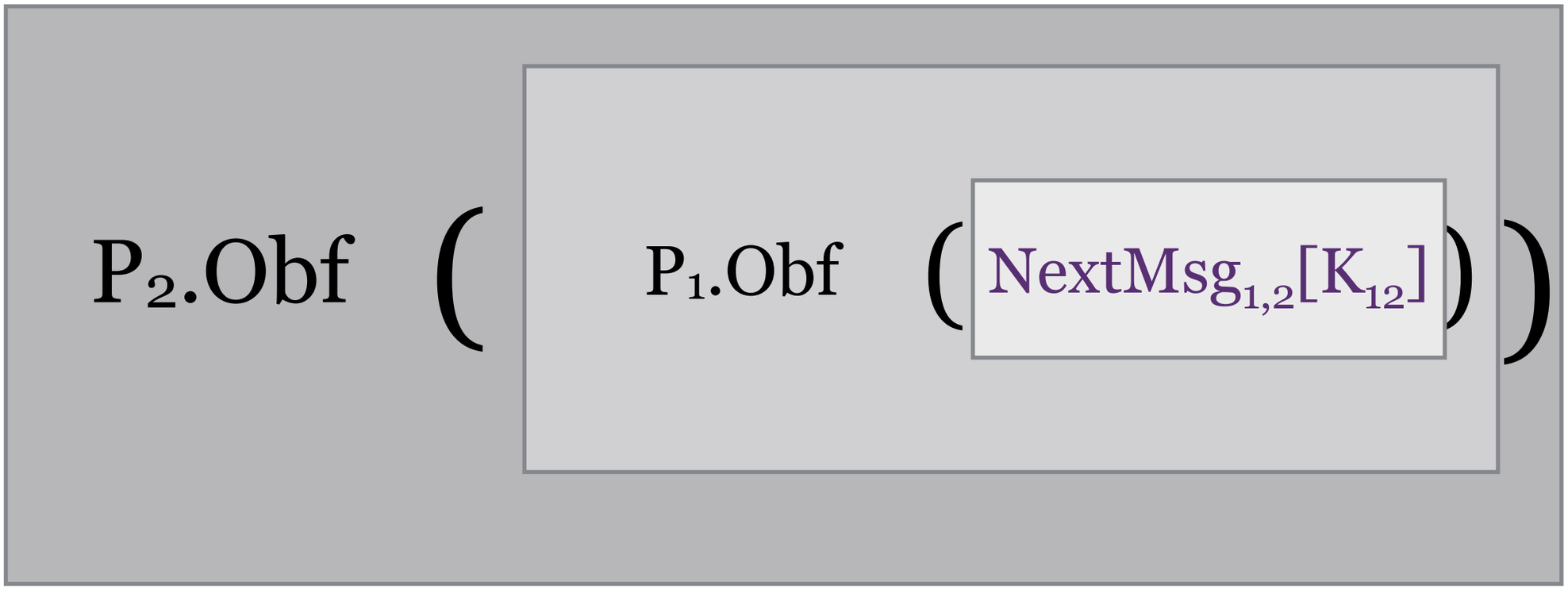
$P_2$.Obf

$K_{12}$ ✗

$NextMsg_2(C_{2,*})$

# Our Fix: Onion Combiner

# Our Fix: Onion Combiner



# Further Ideas

# Further Ideas

- Several other problems: Handling malicious candidates, resetting attacks, avoiding stronger assumptions, …

# Further Ideas

- Several other problems: Handling malicious candidates, resetting attacks, avoiding stronger assumptions, …
- FE allows us to avoid input-by-input arguments, allows us to use only polynomial hardness.

# Further Ideas


- Several other problems: Handling malicious candidates, resetting attacks, avoiding stronger assumptions, …
- FE allows us to avoid input-by-input arguments, allows us to use only polynomial hardness.

# Open Questions

# Open Questions

1. **iO Combiner from polynomial hardness**

# Open Questions

1. **iO Combiner from polynomial hardness**
2. **Combiner for poly–hard Functional Encryption from OWF/DDH**