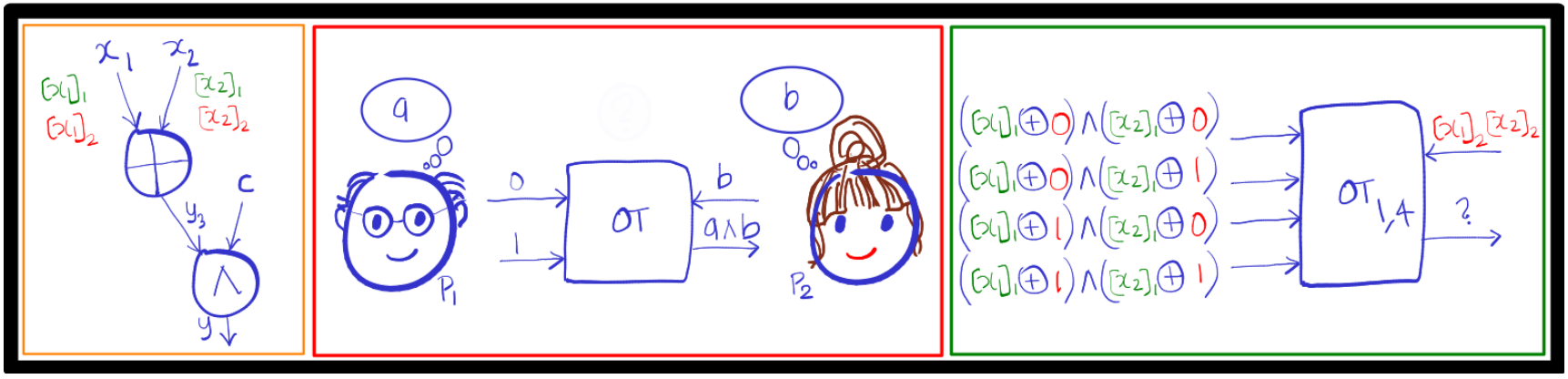


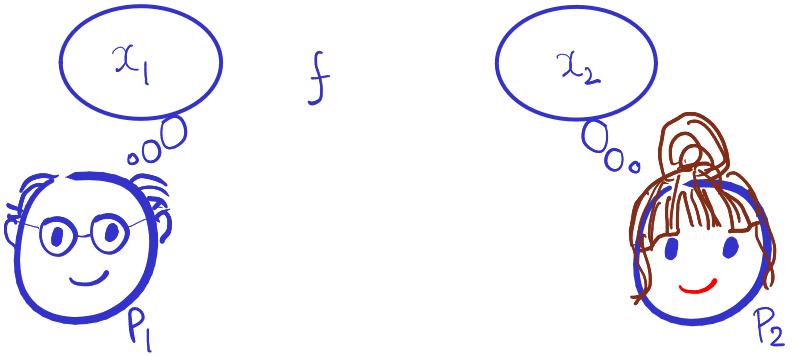
CS783: Theoretical Foundations of Cryptography

Lecture 18 (11/Oct/24)

Instructor: Chethan Kamath

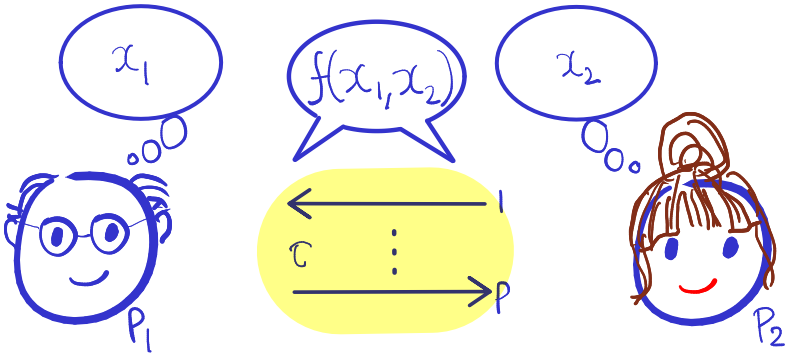
Recall from Last Lecture

- Task 6: Private computation of functions



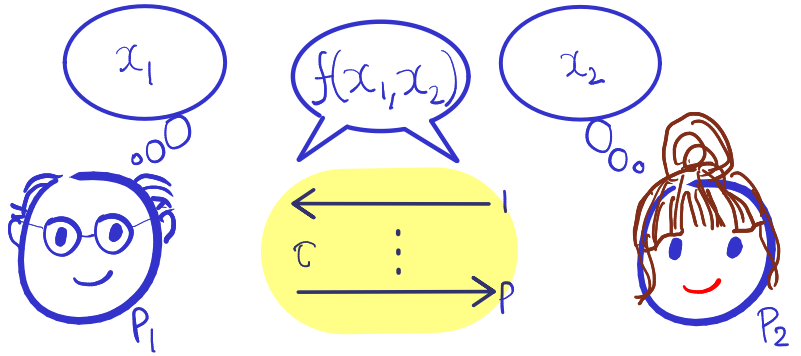
Recall from Last Lecture

- Task 6: Private computation of functions



Recall from Last Lecture

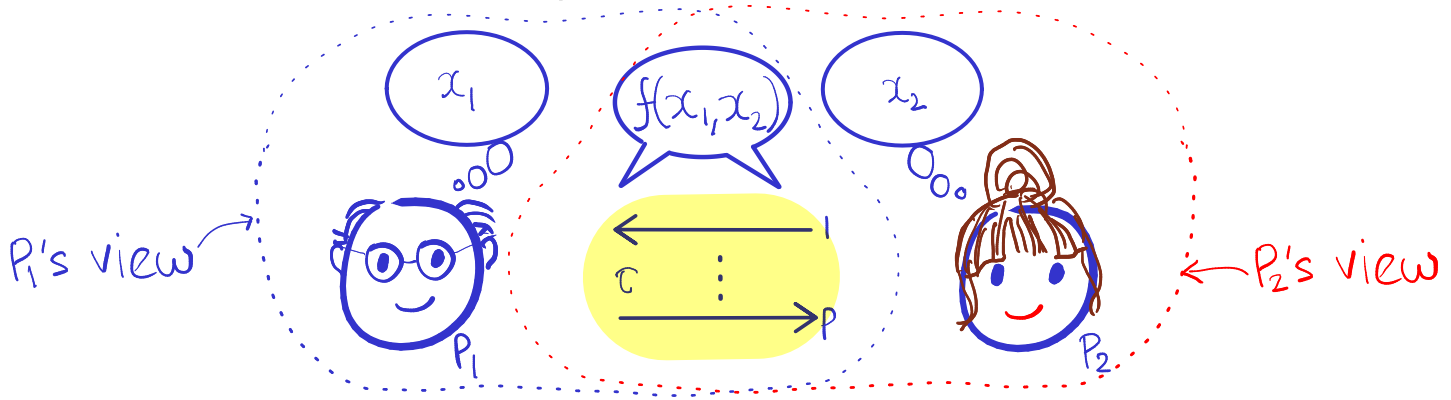
- Task 6: Private computation of functions



- Defined syntax and security for the two-party case (2PC)
  - Special case of 2PC: ZK proof
  - Extends the simulation paradigm

Recall from Last Lecture

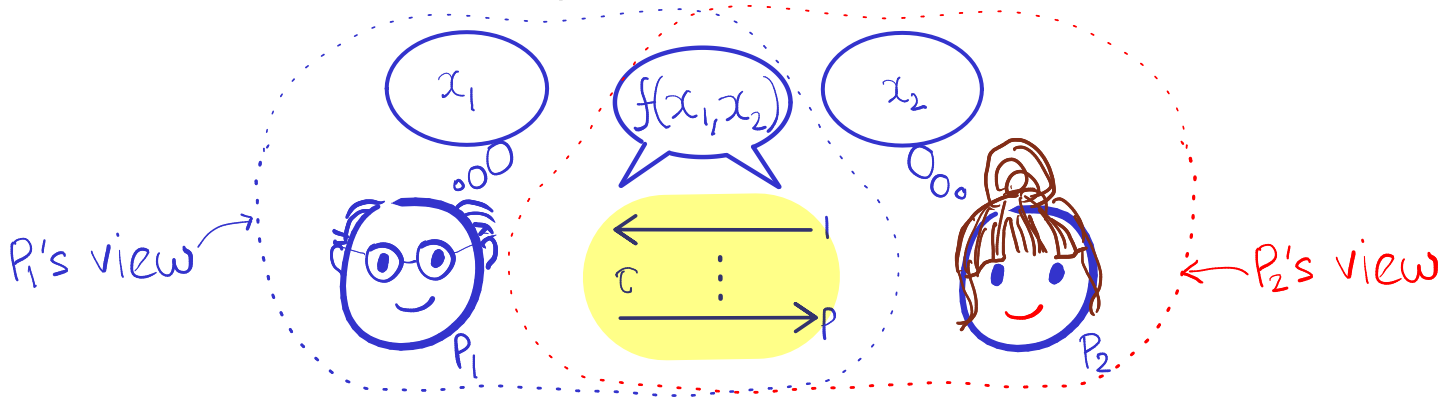
- Task 6: Private computation of functions



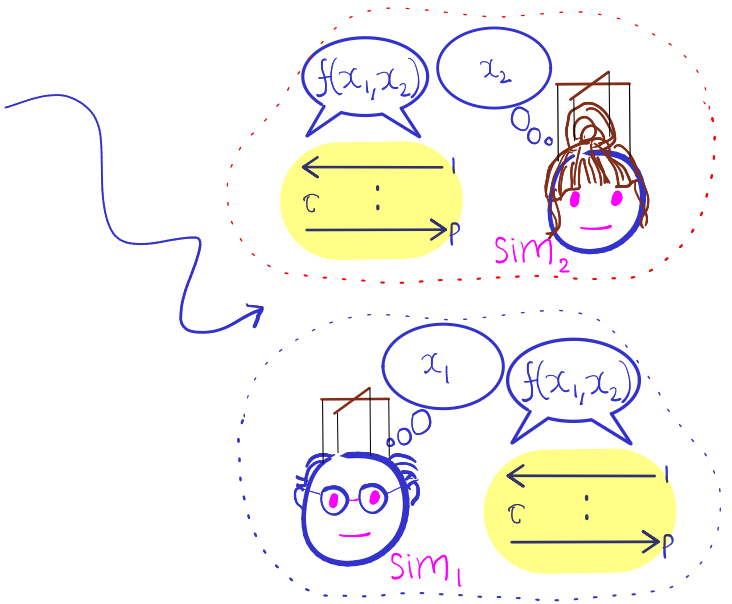
- Defined syntax and security for the two-party case (2PC)
  - Special case of 2PC: ZK proof
  - Extends the simulation paradigm

Recall from Last Lecture

- Task 6: Private computation of functions

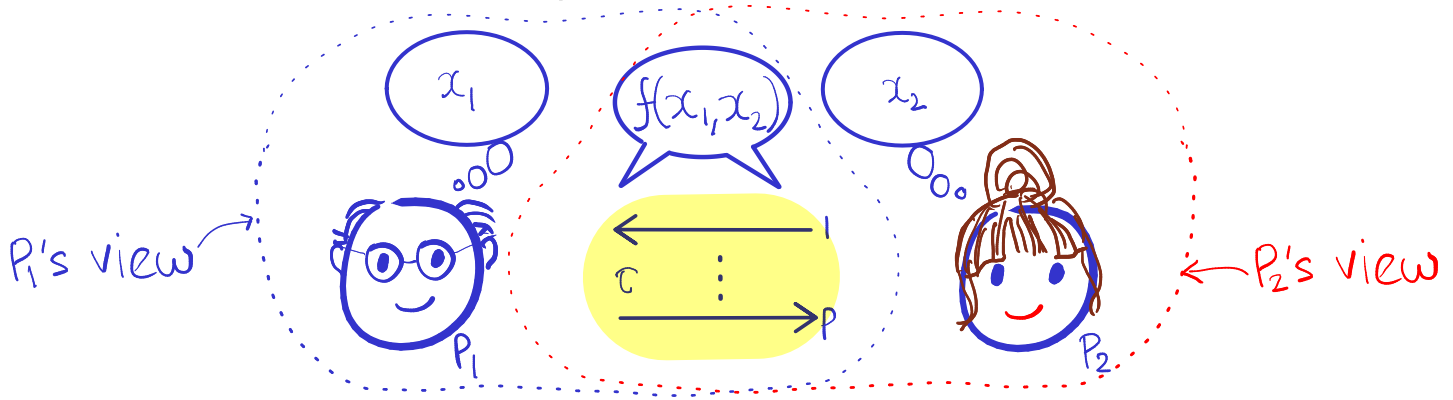


- Defined syntax and security for the two-party case (2PC)
  - Special case of 2PC: ZK proof
  - Extends the simulation paradigm

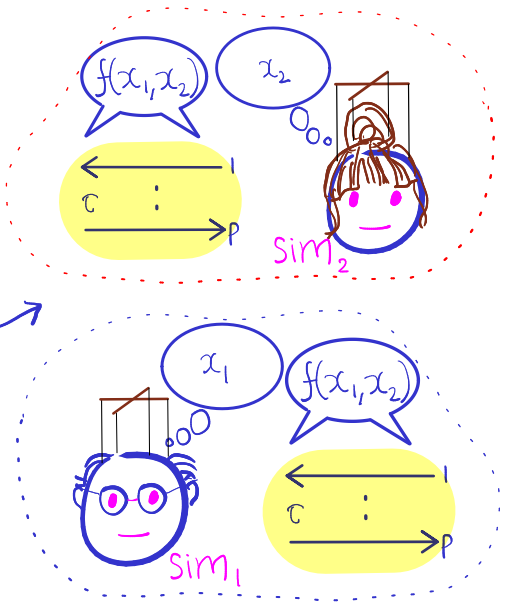


Recall from Last Lecture

- Task 6: Private computation of functions

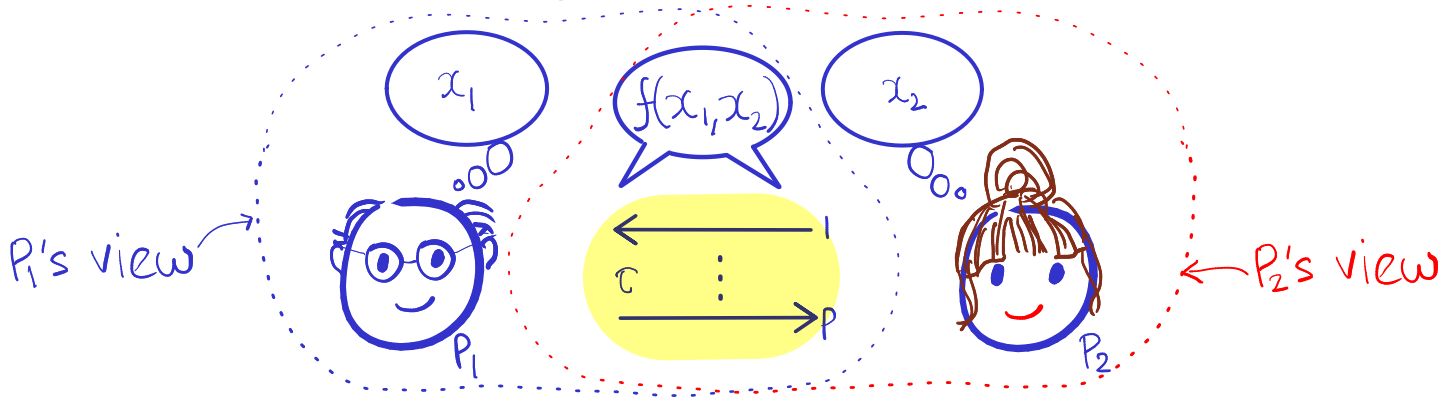


- Defined syntax and security for the two-party case (2PC)
  - Special case of 2PC: ZK proof
  - Extends the simulation paradigm
- Extended to multi-party case (MPC)

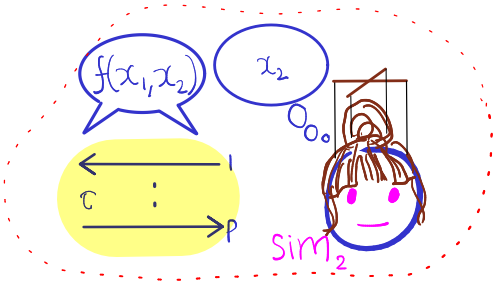


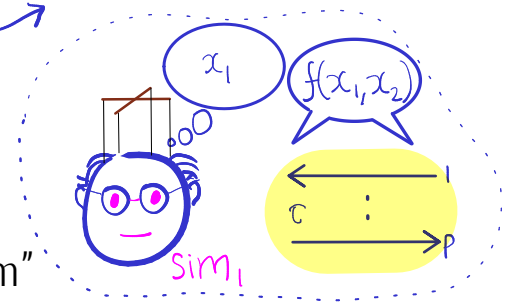
Recall from Last Lecture

- Task 6: Private computation of functions

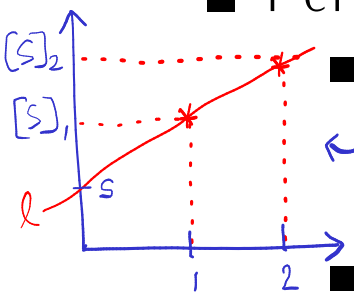


- Defined syntax and security for the two-party case (2PC)
  - Special case of 2PC: ZK proof
  - Extends the simulation paradigm
- Extended to multi-party case (MPC)





- Perfectly-private MPC for *linear* functions
  - Key tool: threshold secret sharing (TSS)
    - Construction: Shamir's TSS
    - Linearity: "sum of shares → shares of sum"
  - Key idea: "compute over shares"



Plan for Today's Lecture...

- What about perfectly-private MPC for *general* functions?

Plan for Today's Lecture...

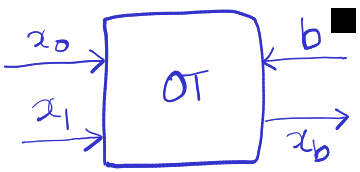
- What about perfectly-private MPC for *general* functions?
  -  Perfectly-private 2PC for *general* functions is impossible!
    - Counter-example: $\wedge$

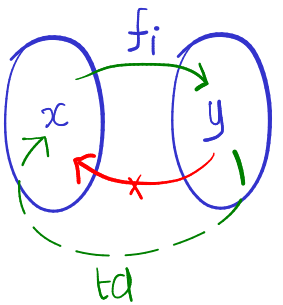
Plan for Today's Lecture...

- What about perfectly-private MPC for *general* functions?
  -  Perfectly-private 2PC for *general* functions is impossible!
    - Counter-example: $\wedge$
- What do we do?

Plan for Today's Lecture...

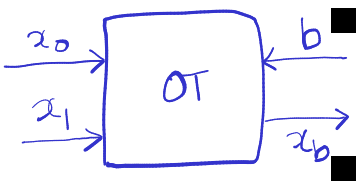
- What about perfectly-private MPC for *general* functions?
  - ⚠ Perfectly-private 2PC for *general* functions is impossible!
    - Counter-example: $\wedge$
- What do we do? Relax to *computational* privacy
  - New cryptographic primitive: oblivious transfer (OT)
    - Trapdoor permutation (TDP) $\rightarrow$ OT

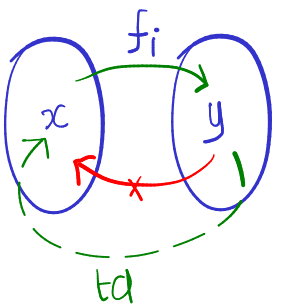




Plan for Today's Lecture...

- What about perfectly-private MPC for *general* functions?
  - ⚠ Perfectly-private 2PC for *general* functions is impossible!
    - Counter-example: $\wedge$

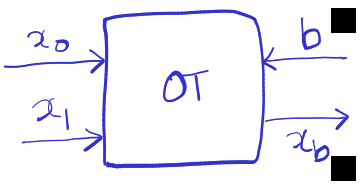


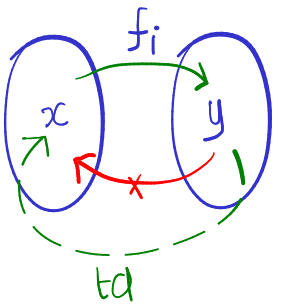


- What do we do? Relax to *computational* privacy
  - New cryptographic primitive: oblivious transfer (OT)
    - Trapdoor permutation (TDP) $\rightarrow$ OT
  - OT $\rightarrow$ computationally-private 2PC for general functions over $\mathbb{F}_2$ (GMW protocol)

Plan for Today's Lecture...

- What about perfectly-private MPC for *general* functions?
  - ⚠ Perfectly-private 2PC for *general* functions is impossible!
    - Counter-example: $\wedge$





- What do we do? Relax to *computational* privacy
  - New cryptographic primitive: oblivious transfer (OT)
    - Trapdoor permutation (TDP) $\rightarrow$ OT
  - OT $\rightarrow$ computationally-private 2PC for general functions over $\mathbb{F}_2$ (GMW protocol)

- Note: perfectly-private MPC for general functions *is* possible *with honest majority*, i.e., if $t < n/2$ (BGW protocol)
  - E.g., three-input $\wedge$ can be computed with perfect privacy if only one of the parties corrupt

Plan for Today's Lecture...

General *template*:

1. Identify the task
2. Come up with precise threat model M (a.k.a security model)
   - Adversary/Attack: What are the adversary's capabilities?
   - Security Goal: What does it mean to be secure?
3. Construct a scheme Π
4. Formally prove that Π in secure in model M

Plan for Today's Lecture...

General *template*:

1. Identify the task → private computation of (arbitrary) functions for two parties
2. Come up with precise threat model M (a.k.a security model)
   - Adversary/Attack: What are the adversary's capabilities?
   - Security Goal: What does it mean to be secure?
3. Construct a scheme Π
4. Formally prove that Π in secure in model M

Plan for Today's Lecture...

General *template*:

1. Identify the task
2. Come up with precise threat model M (a.k.a security model)
   - Adversary/Attack: What are the adversary's capabilities?
   - Security Goal: What does it mean to be secure?
3. Construct a scheme Π
4. Formally prove that Π in secure in model M

Plan for Today's Lecture...

General *template*:

1. Identify the task → private computation of (arbitrary) functions for two parties
2. Come up with precise threat model M (a.k.a security model) → semi-honest model
   - Adversary/Attack: What are the adversary's capabilities? → "static corruption"
   - Security Goal: What does it mean to be secure? → "computational privacy"
3. Construct a scheme Π → GMW protocol
4. Formally prove that Π in secure in model M
   ↳ OT ⇒ simulation

Plan for Today's Lecture

1. Computing $\wedge$ with Perfect Privacy is Impossible
2. New Cryptographic Primitive: Oblivious Transfer (OT)
3. Goldreich-Micali-Wigderson (GMW) Protocol

Plan for Today's Lecture

1. Computing $\wedge$ with Perfect Privacy is Impossible
2. New Cryptographic Primitive: Oblivious Transfer (OT)
3. Goldreich–Micali–Wigderson (GMW) Protocol

Can we Extend Π to Arbitrary Functions?

- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Can we Extend Π to Arbitrary Functions?

- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Can we Extend Π to Arbitrary Functions?

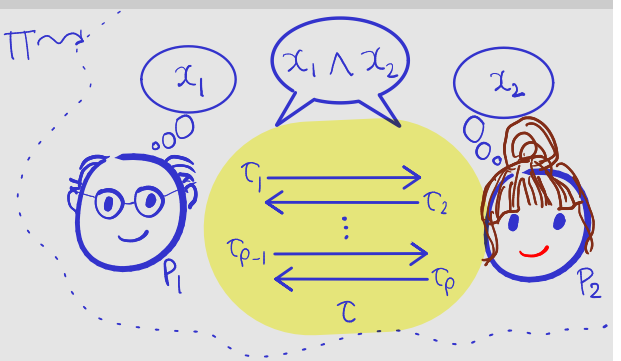
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ $= (\tau_1, \ldots, \tau_p)$ on inputs x_1 & x_2 (over coins of π)



Can we Extend Π to Arbitrary Functions?

- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

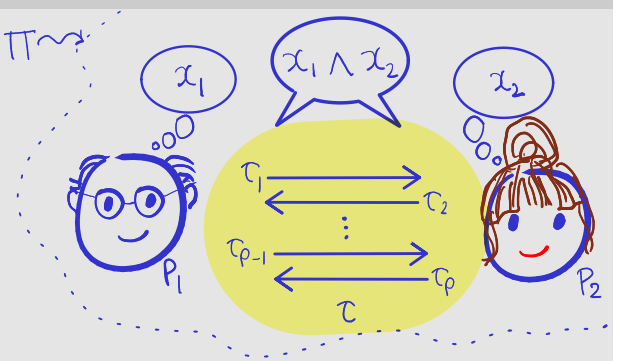
Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ $= (\tau_1, \ldots, \tau_p)$ on inputs x_1 & x_2 (over coins of π)

Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β



Can we Extend Π to Arbitrary Functions?

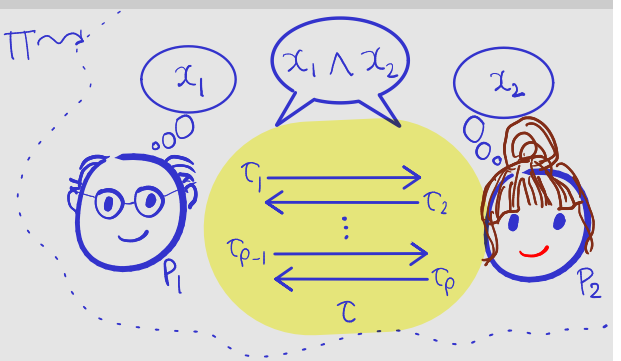
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ (over coins of π) on inputs x_1 & x_2 $\quad (\tau = (\tau_1, \ldots, \tau_p))$



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

Proof:

$$P_{x_1,x_2}(\tau) = \prod_{r=1}^{p} \Pr\left[\tau_r \mid \tau_{r-1}, \ldots \tau_1, x_1, x_2\right]$$

$$= \prod_{\substack{r=1 \\ r \text{ odd}}}^{p} \Pr\left[\tau_r \mid \tau_{r-1}, \ldots \tau_1, x_1, x_2\right] \cdot \prod_{\substack{r=1 \\ r \text{ even}}}^{p} \Pr\left[\tau_r \mid \tau_{r-1}, \ldots \tau_1, x_1, x_2\right]$$

Can we Extend Π to Arbitrary Functions?

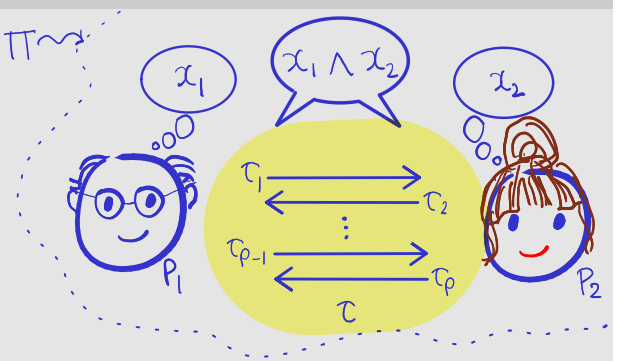
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ (over coins of π) on inputs x_1 & x_2 $= (\tau_1, \ldots, \tau_p)$



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

Proof:

$$P_{x_1,x_2}(\tau) = \prod_{r=1}^{p} \Pr\left[\tau_r \mid \tau_{r-1}, \ldots \tau_1, x_1, x_2\right]$$

$$= \prod_{r=1}^{p} \Pr\left[\tau_r \mid \tau_{r-1}, \ldots \tau_1, x_1, x_2\right] \cdot \prod_{r=1}^{p} \Pr\left[\tau_r \mid \tau_{r-1}, \ldots \tau_1, x_1, x_2\right]$$

r odd $\leftarrow P_1$ speaks; r even $\leftarrow P_2$ speaks

Can we Extend Π to Arbitrary Functions?

- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

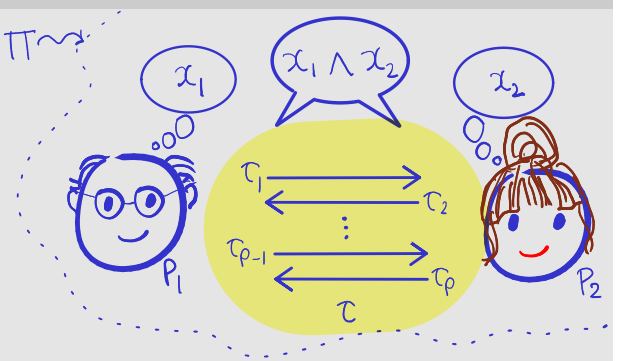
Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

over coins of π $\quad = (r_1, \ldots, r_\rho)$

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ on inputs x_1 & x_2



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

Proof:

$$P_{x_1,x_2}(\tau) = \prod_{r=1}^{\rho} \Pr\left[r_r \mid r_{r-1}, \ldots r_1, x_1, x_2\right]$$

$$= \prod_{r=1}^{\rho} \Pr\left[r_r \mid r_{r-1}, \ldots r_1, x_1, \_\right] \cdot \prod_{r=1}^{\rho} \Pr\left[r_r \mid r_{r-1}, \ldots r_1, \_, x_2\right]$$

r odd ← P_1 speaks

r even ← P_2 speaks

Can we Extend Π to Arbitrary Functions?

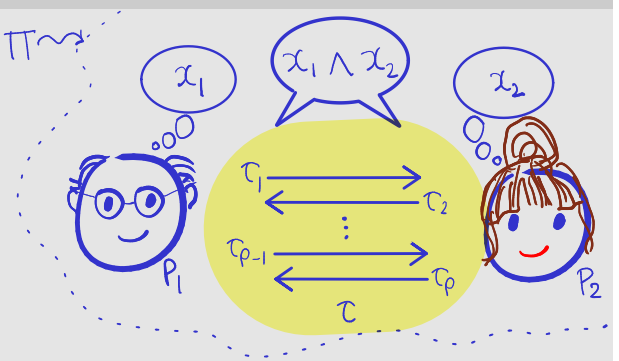
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ $= (\tau_1, \ldots, \tau_\rho)$ on inputs x_1 & x_2 (over coins of π)



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

Proof:

$$P_{x_1,x_2}(\tau) = \prod_{r=1}^{\rho} \Pr\left[\tau_r \mid \tau_{r-1}, \ldots \tau_1, x_1, x_2\right]$$

$$= \prod_{r=1}^{\rho} \Pr\left[\tau_r \mid \tau_{r-1}, \ldots \tau_1, x_1, \_\right] \cdot \prod_{r=1}^{\rho} \Pr\left[\tau_r \mid \tau_{r-1}, \ldots \tau_1, \_, x_2\right]$$

r odd $\leftarrow P_1$ speaks (α) ; r even $\leftarrow P_2$ speaks (β)

Can we Extend Π to Arbitrary Functions?

- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

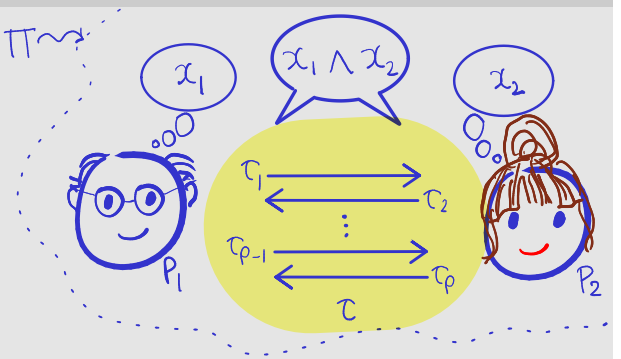
There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is $\tau = (\tau_1, \ldots, \tau_p)$ on inputs x_1 & x_2 (over coins of π)

Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

By perfect privacy, $\forall \tau \quad P_{10}(\tau) = P_{00}(\tau) = P_{01}(\tau)$



Can we Extend Π to Arbitrary Functions?

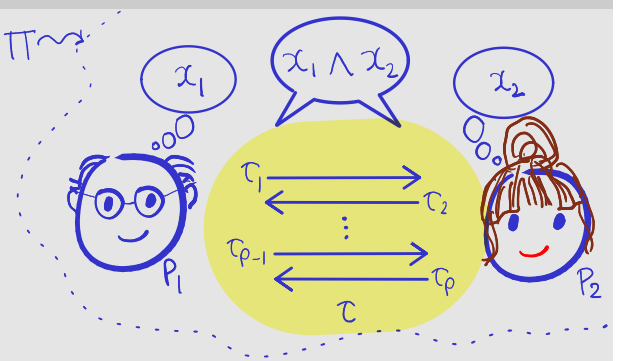
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ ($= (\tau_1, \ldots, \tau_p)$) over coins of π on inputs x_1 & x_2



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

By perfect privacy, $\forall \tau \quad P_{10}(\tau) = P_{00}(\tau) = P_{01}(\tau)$

$$\alpha(\tau,1)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,0) \overset{*}{=} \alpha(\tau,0)\cdot\beta(\tau,1)$$

Can we Extend Π to Arbitrary Functions?

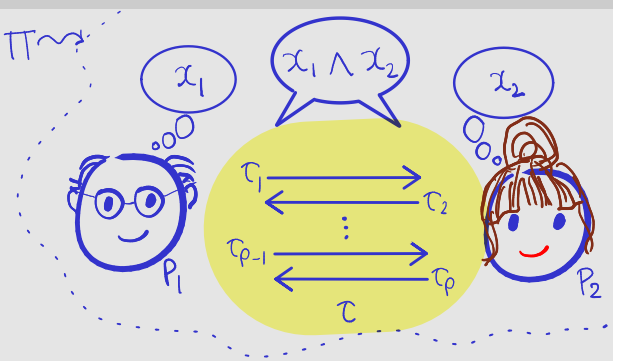
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ $= (\tau_1, \ldots, \tau_p)$ on inputs x_1 & x_2 (over coins of π)



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

By perfect privacy, $\forall \tau \quad P_{10}(\tau) = P_{00}(\tau) = P_{01}(\tau)$

$$\alpha(\tau,1)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,1) \quad (*)$$

$$\alpha(\tau,1) = \alpha(\tau,0) \;\&\; \beta(\tau,0) = \beta(\tau,1)$$

Can we Extend Π to Arbitrary Functions?

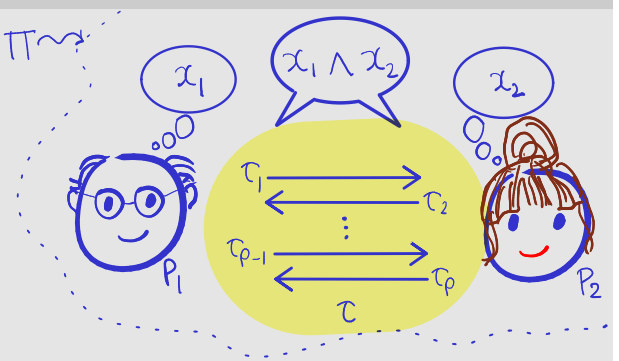
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ (over coins of π) on inputs x_1 & x_2 $= (\tau_1, \ldots, \tau_p)$



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

By perfect privacy, $\forall \tau \quad P_{10}(\tau) = P_{00}(\tau) = P_{01}(\tau)$

$$\Downarrow *$$

$$\alpha(\tau,1)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,1)$$

$$\Downarrow$$

$$\alpha(\tau,1) = \alpha(\tau,0) \;\&\; \beta(\tau,0) = \beta(\tau,1)$$

$$\Downarrow$$

$$P_{11}(\tau) = \alpha(\tau,1)\beta(\tau,1) = \alpha(\tau,0)\beta(\tau,0) = P_{00}(\tau)$$

Can we Extend Π to Arbitrary Functions?

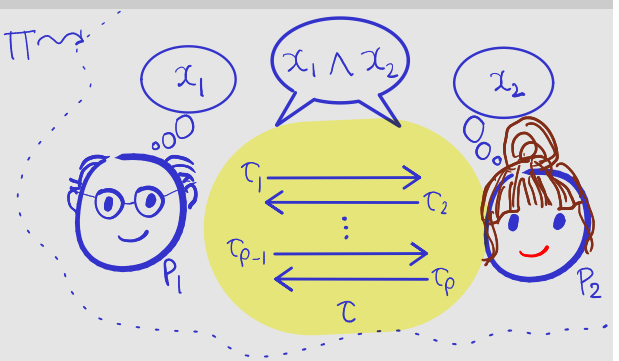
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ (over coins of π) $= (\tau_1, \ldots, \tau_p)$ on inputs x_1 & x_2



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

By perfect privacy, $\forall \tau \quad P_{10}(\tau) = P_{00}(\tau) = P_{01}(\tau)$

$\Downarrow *$

$$\alpha(\tau,1)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,1)$$

$\Downarrow$

$$\alpha(\tau,1) = \alpha(\tau,0) \;\&\; \beta(\tau,0) = \beta(\tau,1)$$

$\Downarrow$

$$P_{11}(\tau) = \alpha(\tau,1)\beta(\tau,1) = \alpha(\tau,0)\beta(\tau,0) = P_{00}(\tau) \implies \pi \text{ incorrect!}$$

Can we Extend Π to Arbitrary Functions?

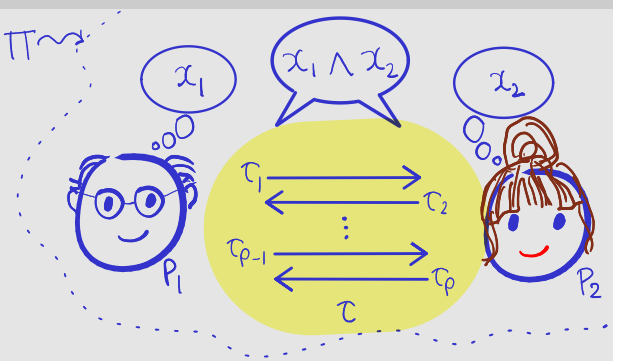
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ (over coins of π) on inputs x_1 & x_2 $= (\tau_1, \ldots, \tau_p)$



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

By perfect privacy, $\forall \tau \quad P_{10}(\tau) = P_{00}(\tau) = P_{01}(\tau)$

$$\Downarrow *$$

$$\alpha(\tau,1)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,1)$$

$$\Downarrow$$

$$\alpha(\tau,1) = \alpha(\tau,0) \;\&\; \beta(\tau,0) = \beta(\tau,1)$$

$$\Downarrow$$

$$P_{11}(\tau) = \alpha(\tau,1)\beta(\tau,1) = \alpha(\tau,0)\beta(\tau,0) = P_{00}(\tau) \implies \pi \text{ incorrect!}$$

□

? Where does this argument fail for $\oplus$?

Can we Extend Π to Arbitrary Functions?

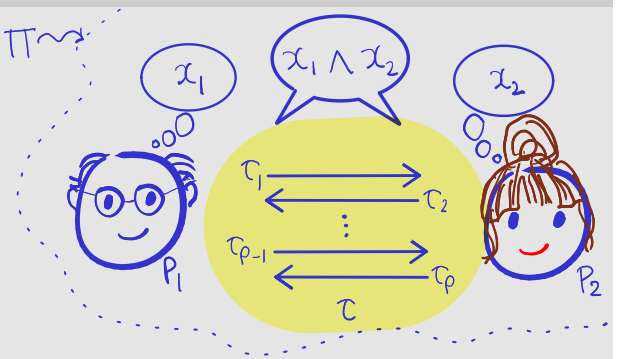
- Need to deal with $\wedge$ over $\mathbb{F}_2$/multiplication over $\mathbb{F}_p$

Claim 1

There does not exist a perfectly-private 2PC protocol Π for $\wedge$

Proof (Idea: $\exists$ perfectly private $\Pi \implies \Pi$ incorrect).

$P_{x_1,x_2}(\tau) :=$ Prob. that transcript is τ ($= (\tau_1, \ldots, \tau_p)$) over coins of π on inputs x_1 & x_2



Subclaim: For any π, $P_{x_1,x_2}(\tau) = \alpha(\tau, x_1) \cdot \beta(\tau, x_2)$ for some α, β

By perfect privacy, $\forall \tau \quad P_{10}(\tau) = P_{00}(\tau) = P_{01}(\tau)$

$\Downarrow *$

$\alpha(\tau,1)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,0) = \alpha(\tau,0)\cdot\beta(\tau,1)$

$\Downarrow$

$\alpha(\tau,1) = \alpha(\tau,0)$ & $\beta(\tau,0) = \beta(\tau,1)$

$\Downarrow$

$P_{11}(\tau) = \alpha(\tau,1)\beta(\tau,1) = \alpha(\tau,0)\beta(\tau,0) = P_{00}(\tau) \implies \pi$ incorrect!

□

? Where does this argument fail for $\oplus$?

Plan for Today's Lecture

1. Computing $\wedge$ with Perfect Privacy is Impossible
2. New Cryptographic Primitive: Oblivious Transfer (OT)
3. Goldreich-Micali-Wigderson (GMW) Protocol

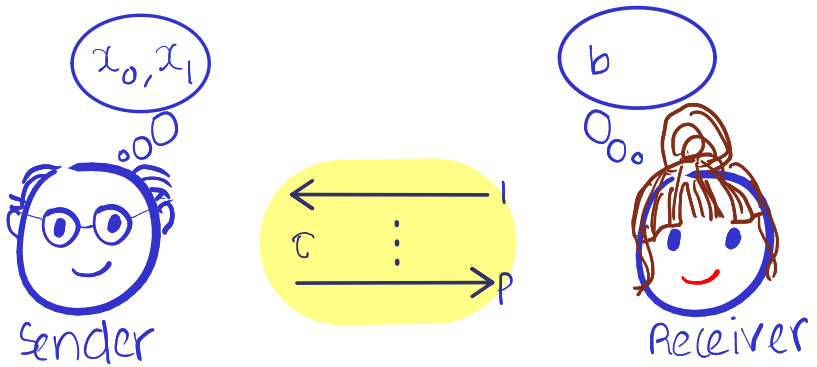
Oblivious Transfer (OT)

- Intuitively: 2PC protocol for "choice function/multiplexer"

$f_b(x_0, x_1) := x_b = (1-b)x_0 + b x_1$

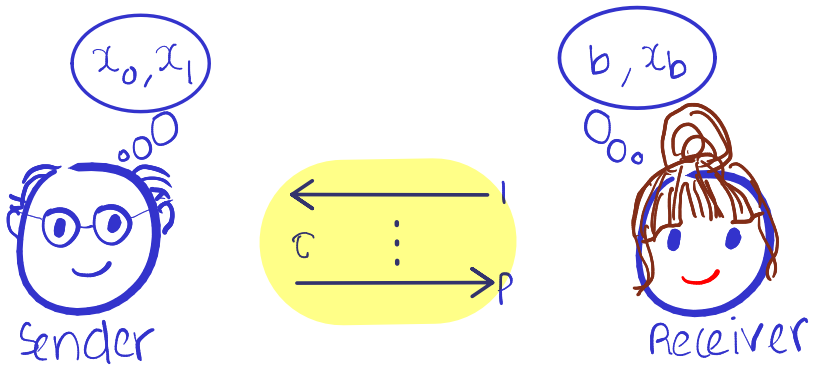
Oblivious Transfer (OT)

- Intuitively: 2PC protocol for "choice function/multiplexer" $f_b(x_0, x_1) := x_b = (1-b)x_0 + b x_1$



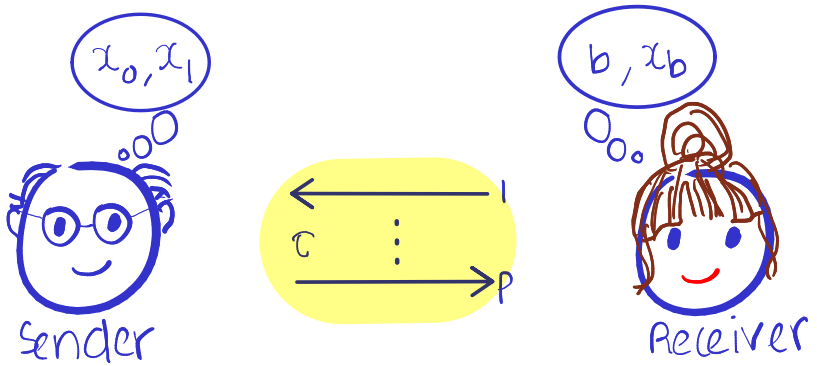
Oblivious Transfer (OT)

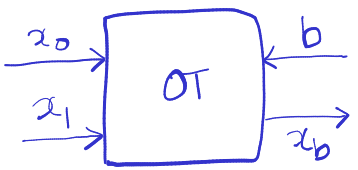
- Intuitively: 2PC protocol for "choice function/multiplexer" $f_b(x_0, x_1) := x_b = (1-b)x_0 + b x_1$



Oblivious Transfer (OT)

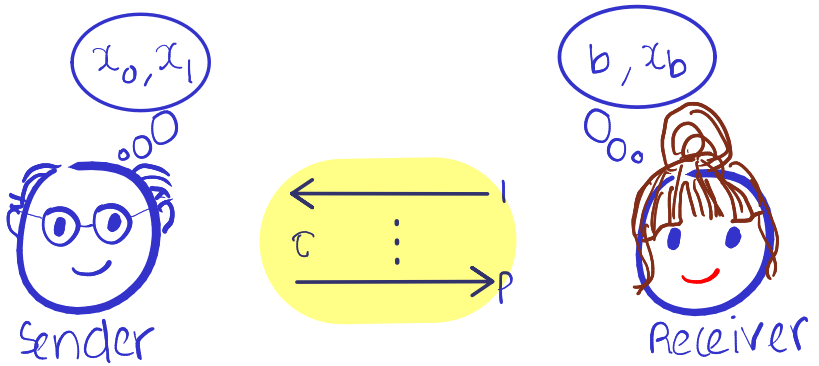
- Intuitively: 2PC protocol for "choice function/multiplexer" $f_b(x_0, x_1) := x_b = (1-b)x_0 + b x_1$



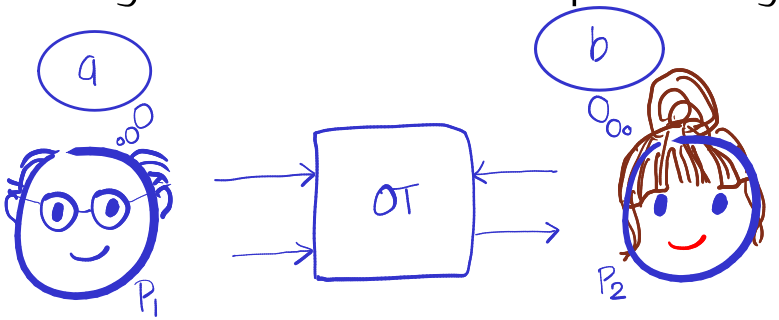


Oblivious Transfer (OT)

- Intuitively: 2PC protocol for "choice function/multiplexer" $f_b(x_0, x_1) := x_b = (1-b)x_0 + b x_1$

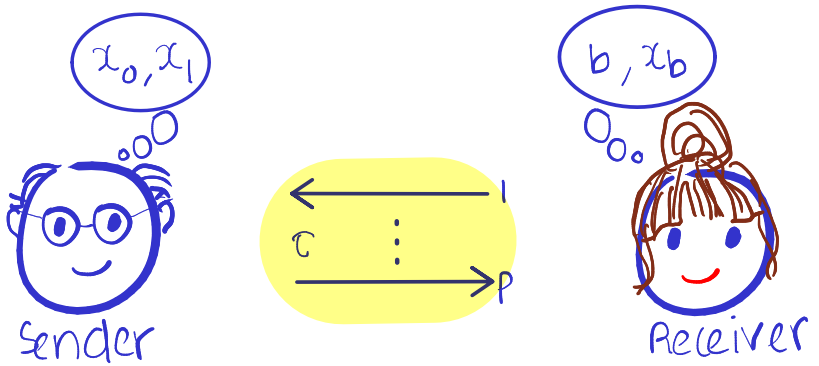


- Why is it useful? E.g., can be used to privately compute $\wedge$

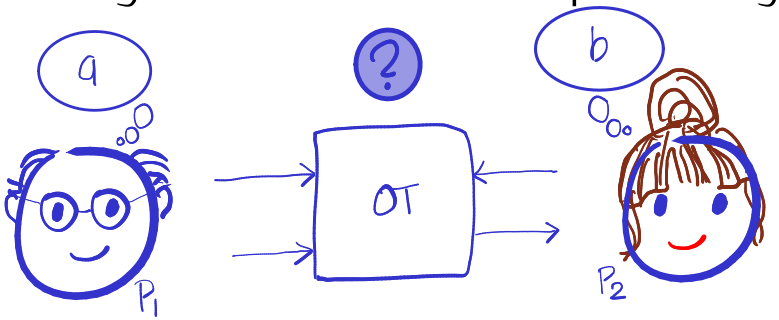


Oblivious Transfer (OT)

- Intuitively: 2PC protocol for "choice function/multiplexer" $f_b(x_0, x_1) := x_b = (1-b)x_0 + b x_1$

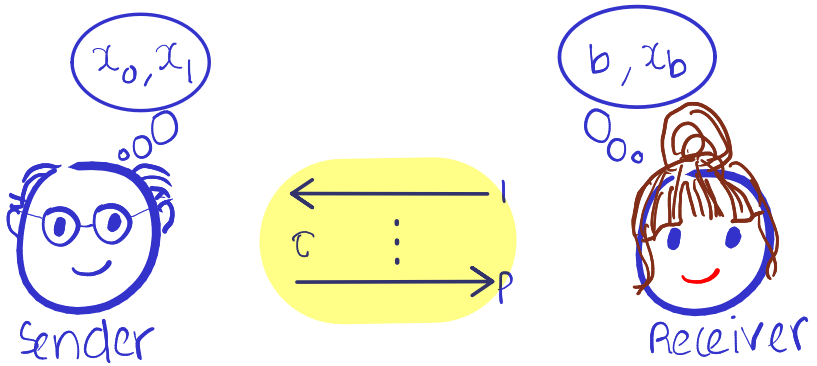


- Why is it useful? E.g., can be used to privately compute $\wedge$

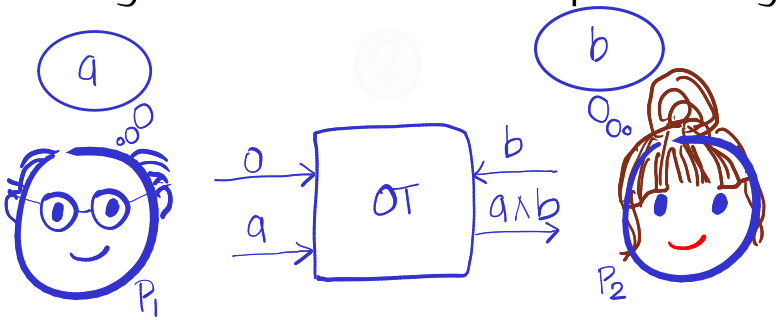


Oblivious Transfer (OT)

- Intuitively: 2PC protocol for "choice function/multiplexer" $f_b(x_0, x_1) := x_b = (1-b)x_0 + b x_1$

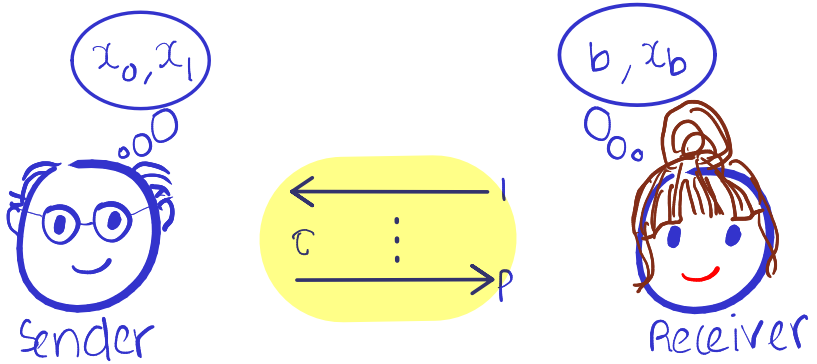


- Why is it useful? E.g., can be used to privately compute $\wedge$

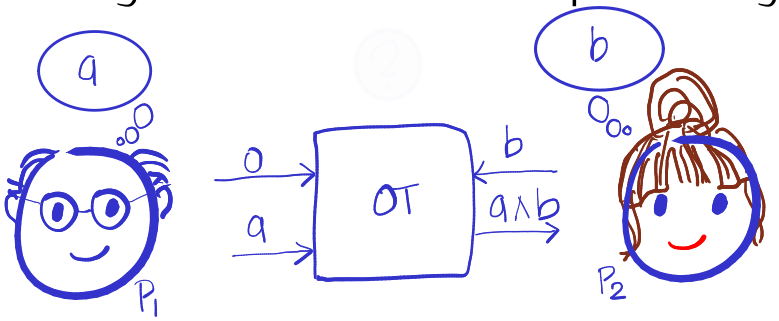


Oblivious Transfer (OT)

- Intuitively: 2PC protocol for "choice function/multiplexer" $f_b(x_0, x_1) := x_b = (1-b)x_0 + b x_1$



- Why is it useful? E.g., can be used to privately compute $\wedge$



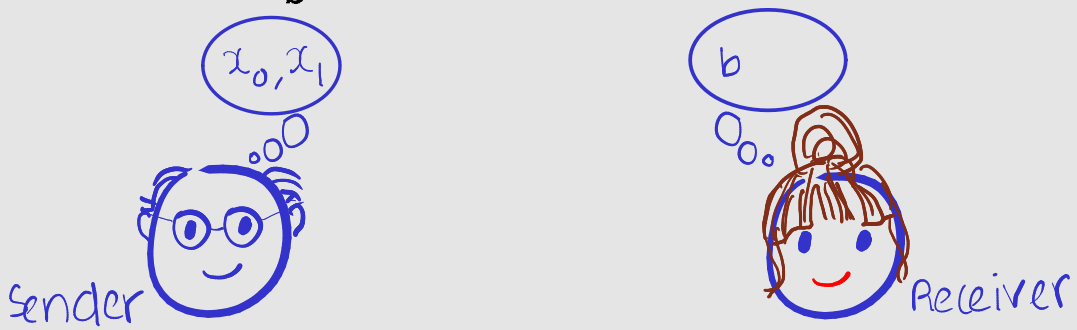
Exercise 1 (Hint: Need to invoke OT multiple times)

Implement comparison operator $\leq$ for n-bit integers using OT

Syntax and Security of OT

Defintion 1 (Oblivious Transfer)

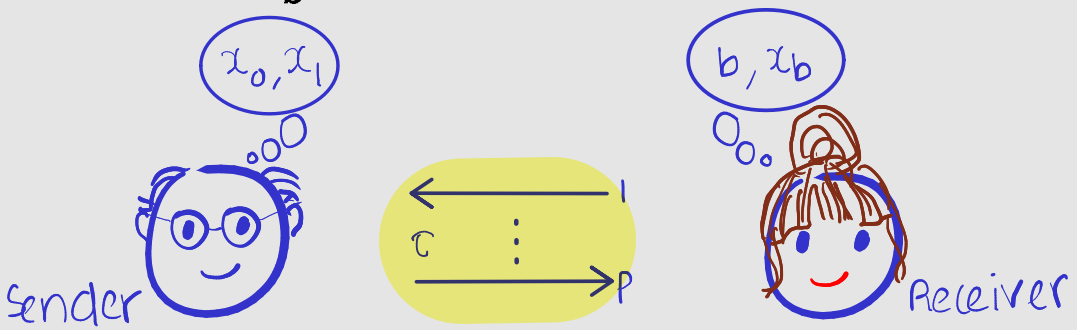
An interactive protocol $\Pi = (\mathsf{S}, \mathsf{R})$ *between a sender* S *with input bits* $x_0, x_1 \in \{0,1\}$ *and receiver* R *with choice bit* $b \in \{0,1\}$*, at the end of which* R *learns* x_b



Syntax and Security of OT

Defintion 1 (Oblivious Transfer)

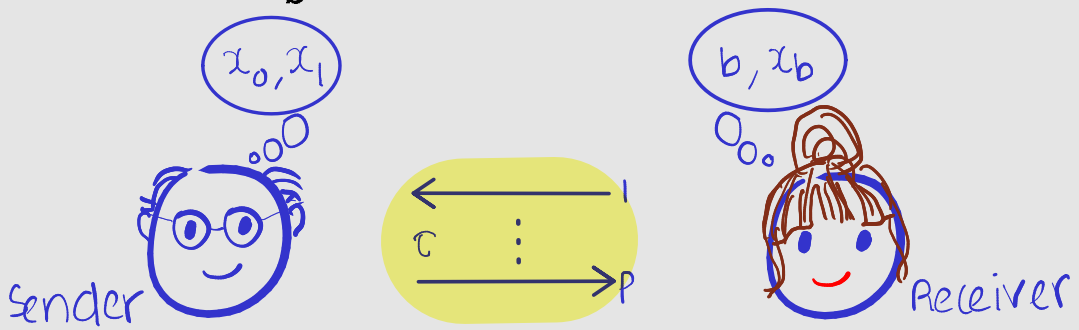
An interactive protocol $\Pi = (\mathsf{S}, \mathsf{R})$ *between a sender* S *with input bits* $x_0, x_1 \in \{0,1\}$ *and receiver* R *with choice bit* $b \in \{0,1\}$*, at the end of which* R *learns* x_b



Syntax and Security of OT

Defintion 1 (Oblivious Transfer)

An interactive protocol $\Pi = (\mathsf{S}, \mathsf{R})$ *between a sender* S *with input bits* $x_0, x_1 \in \{0,1\}$ *and receiver* R *with choice bit* $b \in \{0,1\}$*, at the end of which* R *learns* x_b

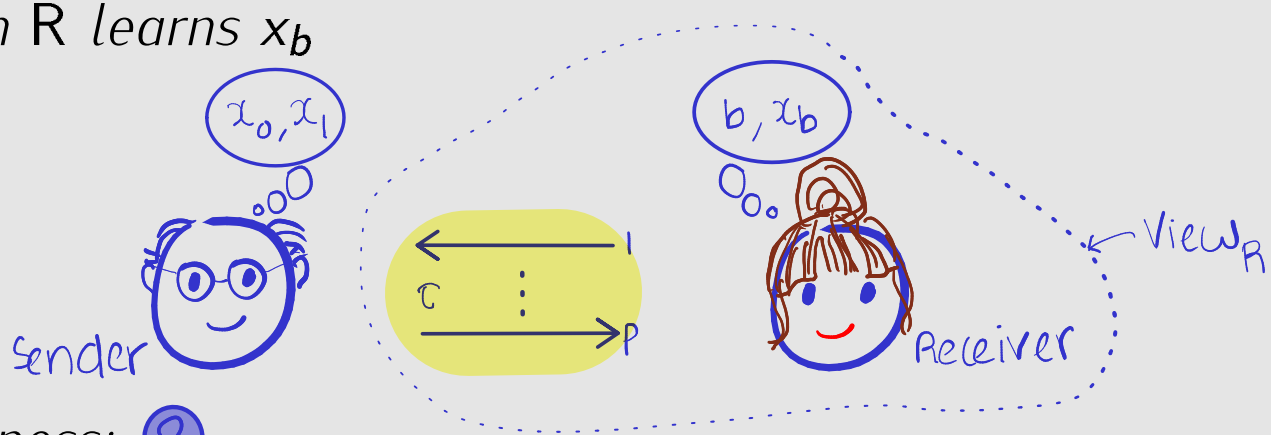


- *Correctness:* ?

Syntax and Security of OT

Defintion 1 (Oblivious Transfer)

An interactive protocol $\Pi = (\mathsf{S}, \mathsf{R})$ *between a sender* S *with input bits* $x_0, x_1 \in \{0,1\}$ *and receiver* R *with choice bit* $b \in \{0,1\}$*, at the end of which* R *learns* x_b



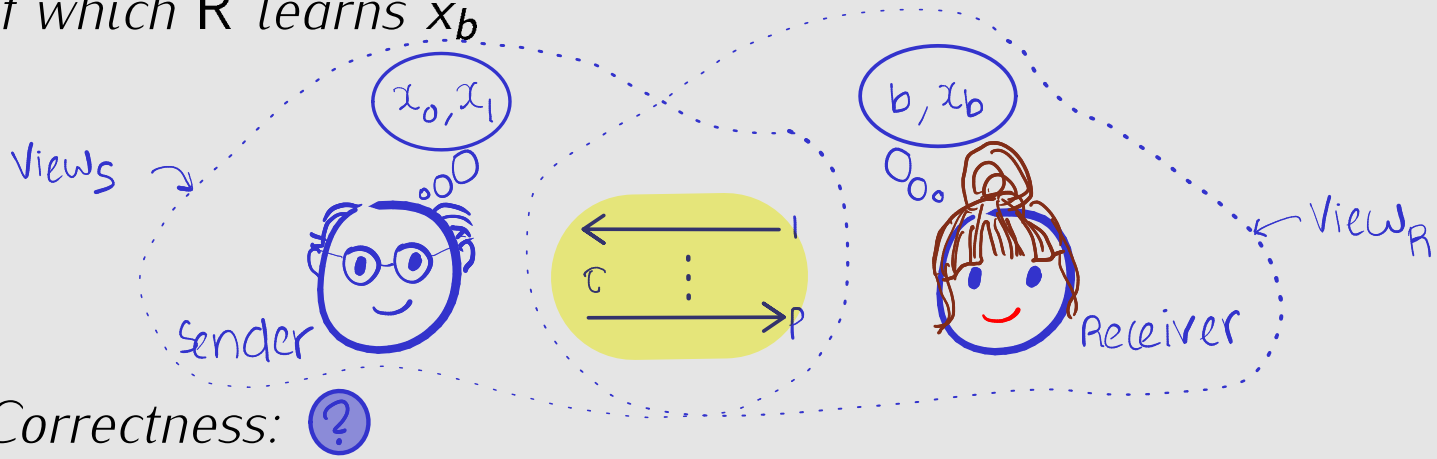
- *Correctness:*

- Sender privacy: (honest) R should not learn input x_{1-b}
  - $\exists$ simulator Sim_R that simulates $\text{View}_\mathsf{R}(\langle \mathsf{S}(x_0, x_1), \mathsf{R}(b) \rangle)$

Syntax and Security of OT

Defintion 1 (Oblivious Transfer)

An interactive protocol $\Pi = (\mathsf{S}, \mathsf{R})$ *between a sender* S *with input bits* $x_0, x_1 \in \{0,1\}$ *and receiver* R *with choice bit* $b \in \{0,1\}$*, at the end of which* R *learns* x_b



- *Correctness:*

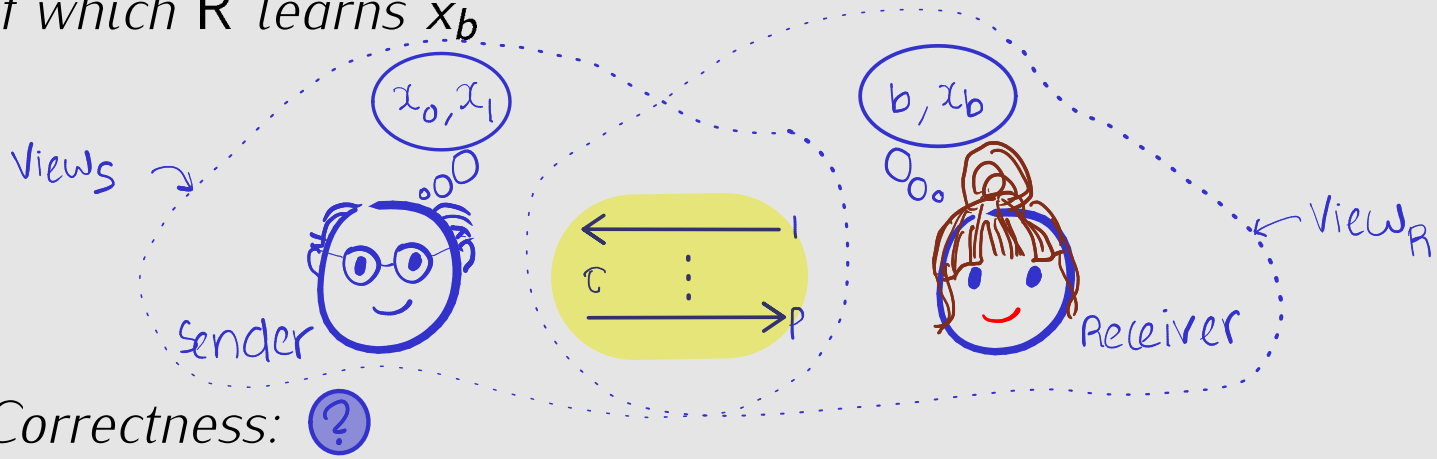
- Sender privacy: (honest) R should not learn input x_{1-b}
  - $\exists$ simulator Sim_R that simulates $\text{View}_\mathsf{R}(\langle \mathsf{S}(x_0, x_1), \mathsf{R}(b) \rangle)$
- Receiver privacy: (honest) S should not learn choice bit b
  - $\exists$ simulator Sim_S that simulates $\text{View}_\mathsf{S}(\langle \mathsf{S}(x_0, x_1), \mathsf{R}(b) \rangle)$

Syntax and Security of OT

1-out-of-2

Defintion 1 (Oblivious Transfer)

An interactive protocol $\Pi = (\mathsf{S}, \mathsf{R})$ *between a sender* S *with input bits* $x_0, x_1 \in \{0,1\}$ *and receiver* R *with choice bit* $b \in \{0,1\}$*, at the end of which* R *learns* x_b



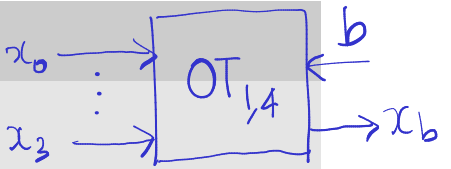
- *Correctness:* ?

- Sender privacy: (honest) R should not learn input x_{1-b}
  - $\exists$ simulator $\mathsf{Sim_R}$ that simulates $\text{View}_{\mathsf{R}}(\langle \mathsf{S}(x_0, x_1), \mathsf{R}(b) \rangle)$
- Receiver privacy: (honest) S should not learn choice bit b
  - $\exists$ simulator $\mathsf{Sim_S}$ that simulates $\text{View}_{\mathsf{S}}(\langle \mathsf{S}(x_0, x_1), \mathsf{R}(b) \rangle)$

Exercise 2

$S: x_0, x_1, x_2, x_3 \in \{0,1\}^\lambda \quad R: b \in \{0, \ldots, 3\}$

Construct a 1*-out-of-*4 *OT from a* 1*-out-of-*2 *OT.*



Recall Trapdoor Permutation (TDP) from Lecture 13...

- One-way permutation that is easy to invert given a "trapdoor"

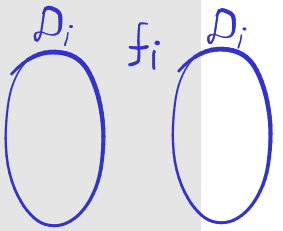
Recall Trapdoor Permutation (TDP) from Lecture 13...

- One-way permutation that is easy to invert given a "trapdoor"

Defintion 2 (Trapdoor (one-way) permutation (TDP) collection)

A collection of permutations $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ *is trapdoor one-way if*

1. *There is an efficient index+trapdoor sampling algorithm* Index



Recall Trapdoor Permutation (TDP) from Lecture 13...

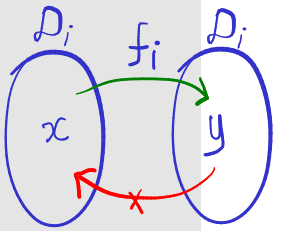
- One-way permutation that is easy to invert given a "trapdoor"

Defintion 2 (Trapdoor (one-way) permutation (TDP) collection)

A collection of permutations $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ *is trapdoor one-way if*

1. *There is an efficient index+trapdoor sampling algorithm* Index
2. *Each* f_i, $i \in \mathcal{I}$, *is efficiently computable*
3. *For all PPT inverters* Inv, *the following is negligible:*

$$p(n) := \Pr_{\substack{(i,\tau) \leftarrow \mathsf{Index}(1^n) \\ x \leftarrow \mathcal{D}_i}} [\mathsf{Inv}(\mathsf{f}_i(x)) \in \mathsf{f}_i^{-1}(\mathsf{f}_i(x))]$$



Recall Trapdoor Permutation (TDP) from Lecture 13...

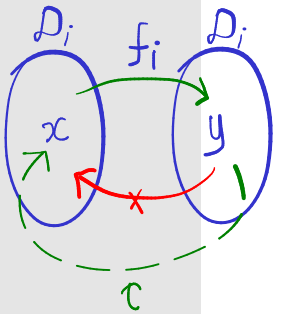
- One-way permutation that is easy to invert given a "trapdoor"

Defintion 2 (Trapdoor (one-way) permutation (TDP) collection)

A collection of permutations $\mathsf{f} = \{\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i\}_{i \in \mathcal{I} \subseteq \{0,1\}^*}$ *is trapdoor one-way if*

1. *There is an efficient index+trapdoor sampling algorithm* Index
2. *Each* f_i, $i \in \mathcal{I}$, *is efficiently computable*
3. *For all PPT inverters* Inv, *the following is negligible:*

$$p(n) := \Pr_{\substack{(i,\tau) \leftarrow \mathsf{Index}(1^n) \\ x \leftarrow \mathcal{D}_i}} [\mathsf{Inv}(\mathsf{f}_i(x)) \in \mathsf{f}_i^{-1}(\mathsf{f}_i(x))]$$



4. f_i^{-1} *can be efficiently computed given trapdoor* τ *for* i

Recall Trapdoor Permutation (TDP) from Lecture 13...

- Example: RSA TDP $\mathsf{f}_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times$ defined as

$$\mathsf{f}_{N,e}(x) := x^e \bmod N$$

  - $\mathsf{f}_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption
  - The trapdoor is $d := e^{-1} \bmod (p-1)(q-1)$

Recall Trapdoor Permutation (TDP) from Lecture 13...

- Example: RSA TDP $\mathsf{f}_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times$ defined as

$$\mathsf{f}_{N,e}(x) := x^e \bmod N$$

  - $\mathsf{f}_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption
  - The trapdoor is $d := e^{-1} \bmod (p-1)(q-1)$

bit

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ $\leftarrow$ TDP $\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i$)

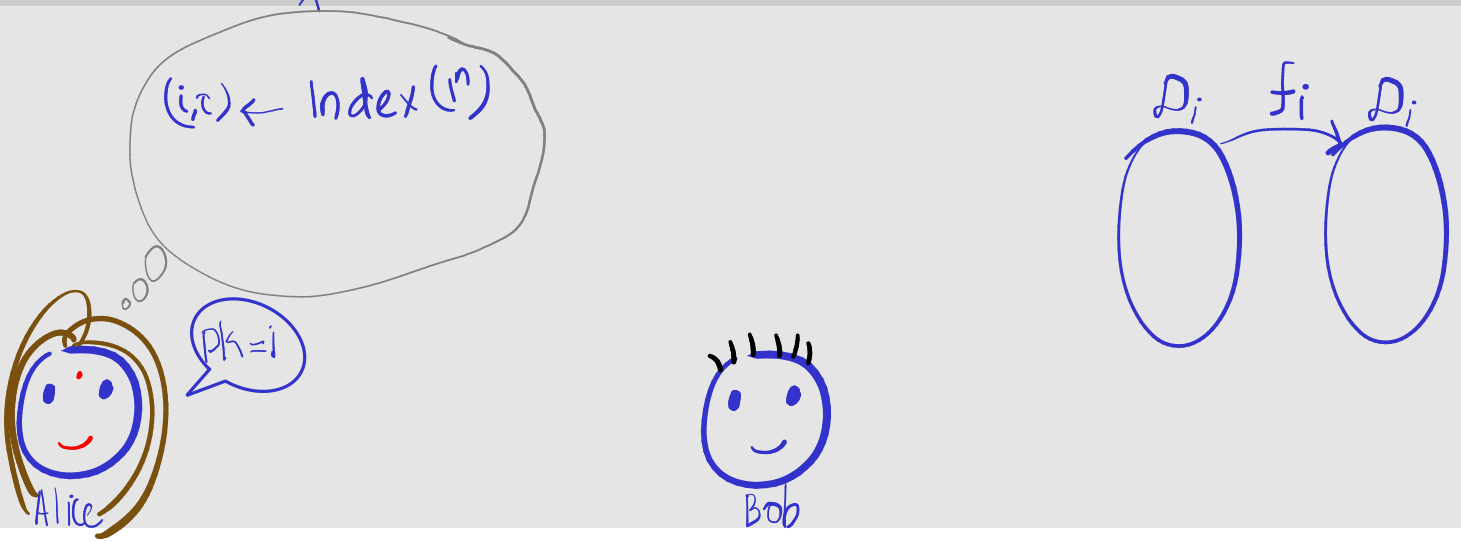
Recall Trapdoor Permutation (TDP) from Lecture 13...

- Example: RSA TDP $\mathsf{f}_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times$ defined as

$$\mathsf{f}_{N,e}(x) := x^e \bmod N$$

  - $\mathsf{f}_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption
  - The trapdoor is $d := e^{-1} \bmod (p-1)(q-1)$

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ $\leftarrow$ TDP $\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i$)



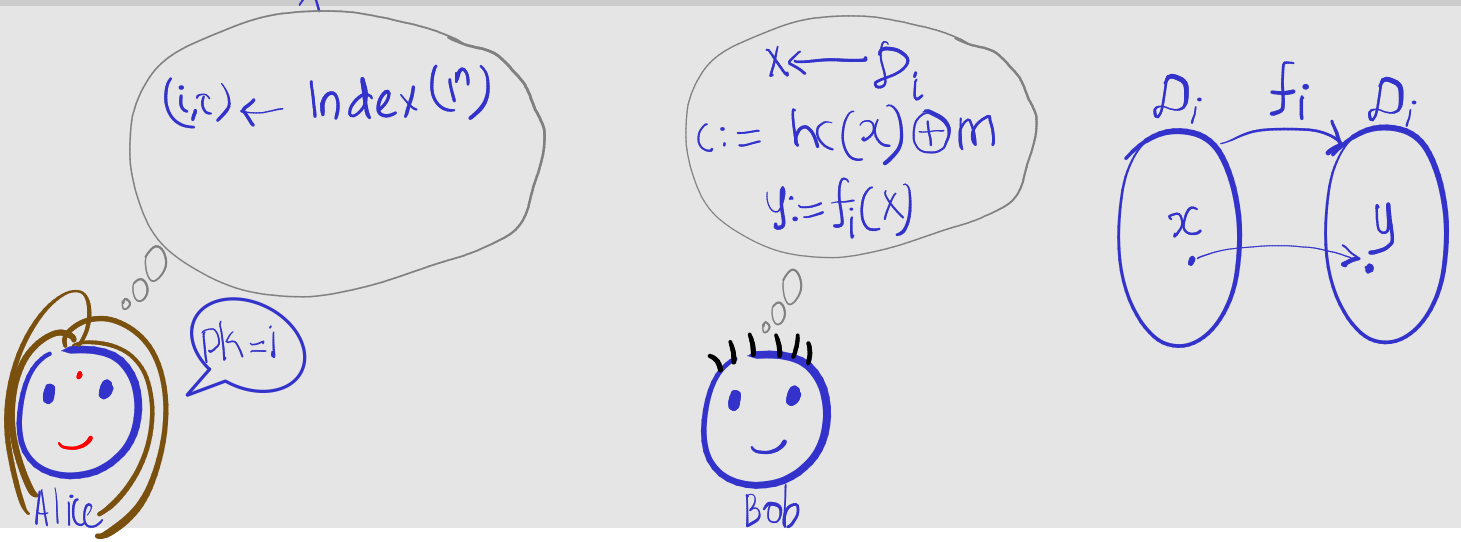
Recall Trapdoor Permutation (TDP) from Lecture 13...

- Example: RSA TDP $\mathsf{f}_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times$ defined as

$$\mathsf{f}_{N,e}(x) := x^e \bmod N$$

  - $\mathsf{f}_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption
  - The trapdoor is $d := e^{-1} \bmod (p-1)(q-1)$

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ $\leftarrow$ TDP $\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i$)



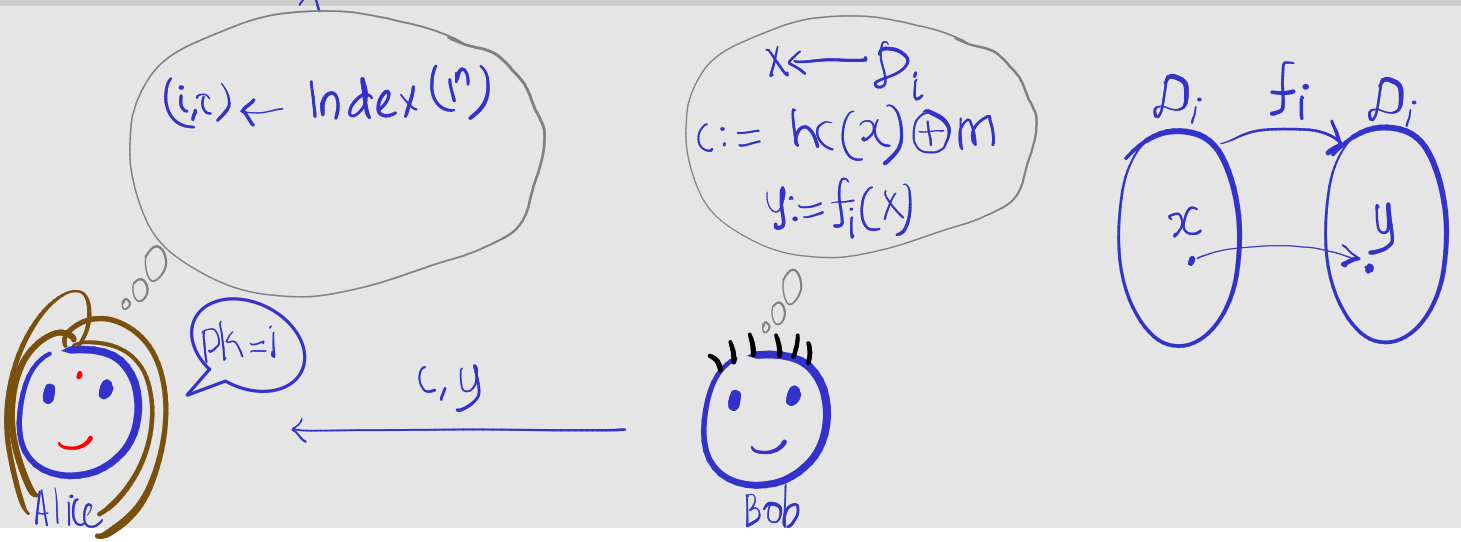
Recall Trapdoor Permutation (TDP) from Lecture 13...

- Example: RSA TDP $\mathsf{f}_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times$ defined as

$$\mathsf{f}_{N,e}(x) := x^e \bmod N$$

  - $\mathsf{f}_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption
  - The trapdoor is $d := e^{-1} \bmod (p-1)(q-1)$

Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ $\leftarrow$ TDP $\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i$)



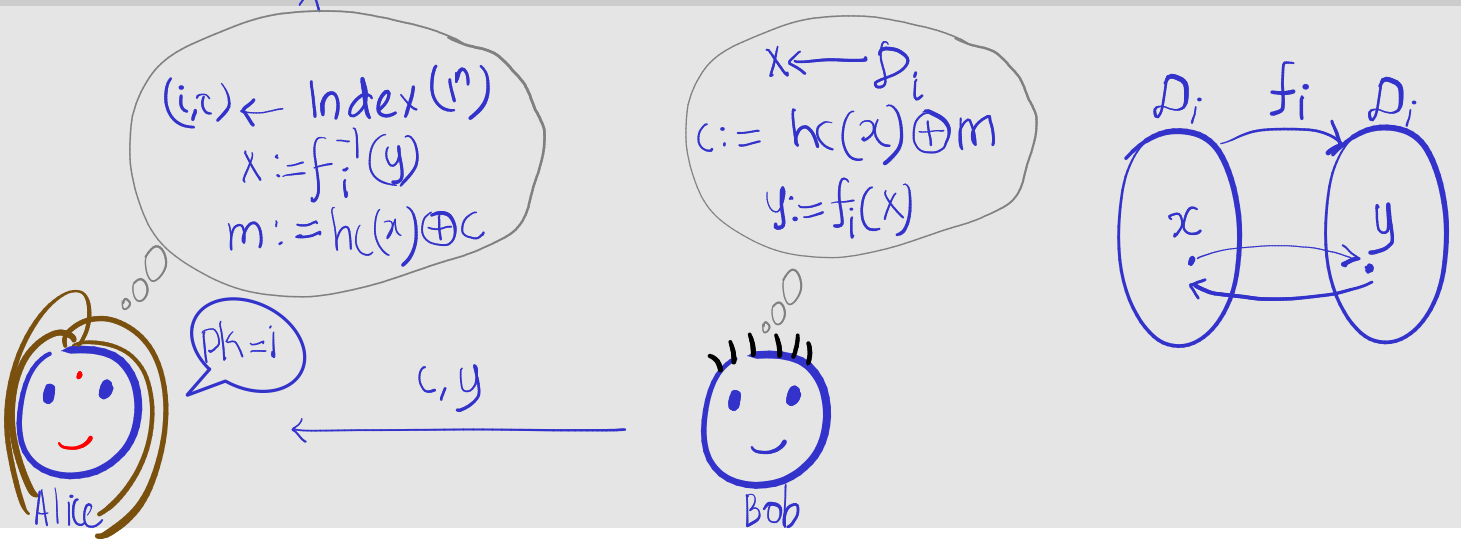
Recall Trapdoor Permutation (TDP) from Lecture 13...

- Example: RSA TDP $\mathsf{f}_{N,e} : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times$ defined as

$$\mathsf{f}_{N,e}(x) := x^e \bmod N$$

  - $\mathsf{f}_{N,e}$ is permutation when $GCD(e, (p-1)(q-1)) = 1$
  - One-way by RSA assumption
  - The trapdoor is $d := e^{-1} \bmod (p-1)(q-1)$

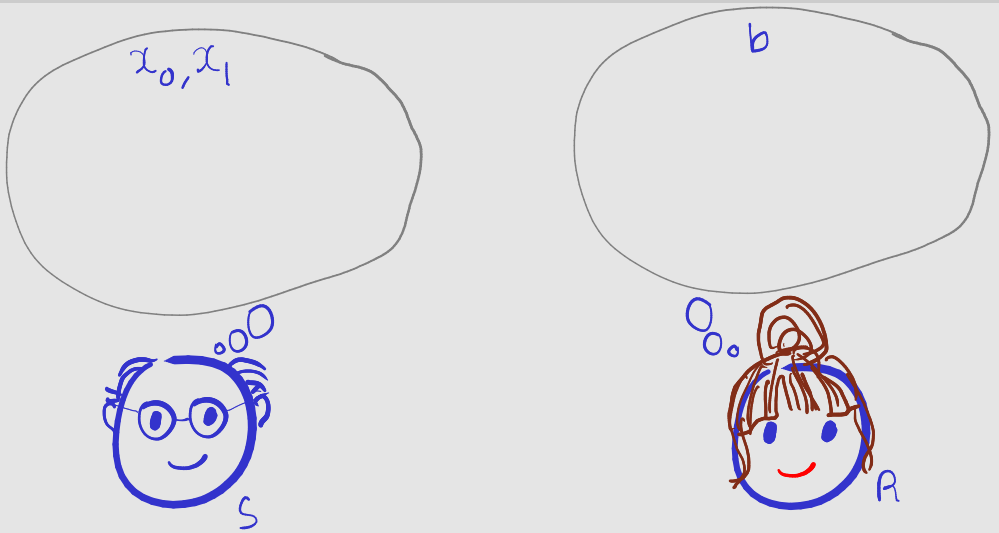
Construction 1 (PKE $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ $\leftarrow$ TDP $\mathsf{f}_i : \mathcal{D}_i \to \mathcal{D}_i$)



TDP $\rightarrow$ OT...

How to tweak Construction 1 to get TDP $\rightarrow$ OT?

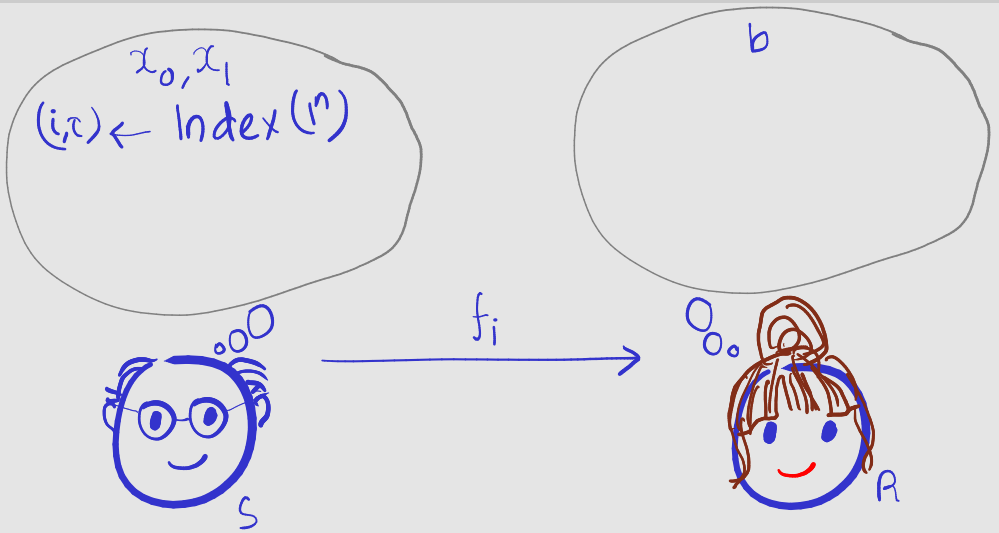
Protocol 1 (OT $\Pi = (\mathsf{S}, \mathsf{R}) \leftarrow$ TDP $\mathsf{f}_i : \mathcal{D}_i \rightarrow \mathcal{D}_i$)

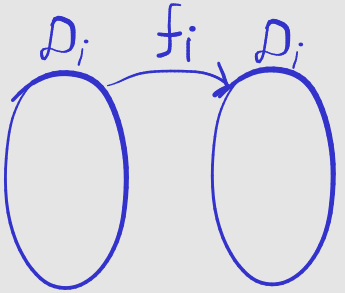


TDP → OT...

How to tweak Construction 1 to get TDP → OT?

Protocol 1 (OT $\Pi = (S, R) \leftarrow$ TDP $f_i : \mathcal{D}_i \to \mathcal{D}_i$)



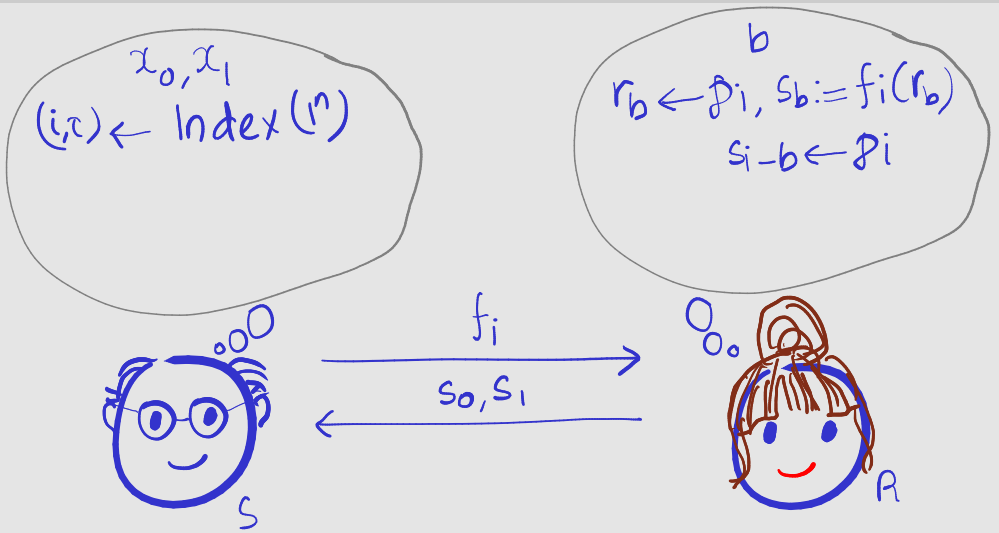


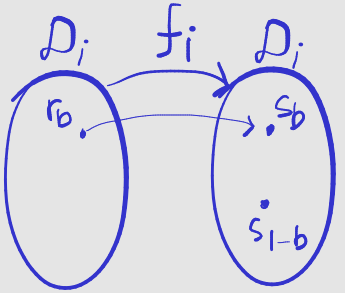
1 *Sender: sample* (f_i, f_i^{-1}) *and send* f_i *to receiver*

TDP → OT...

How to tweak Construction 1 to get TDP → OT?

Protocol 1 (OT $\Pi = (S, R)$ ← TDP $f_i : \mathcal{D}_i \to \mathcal{D}_i$)



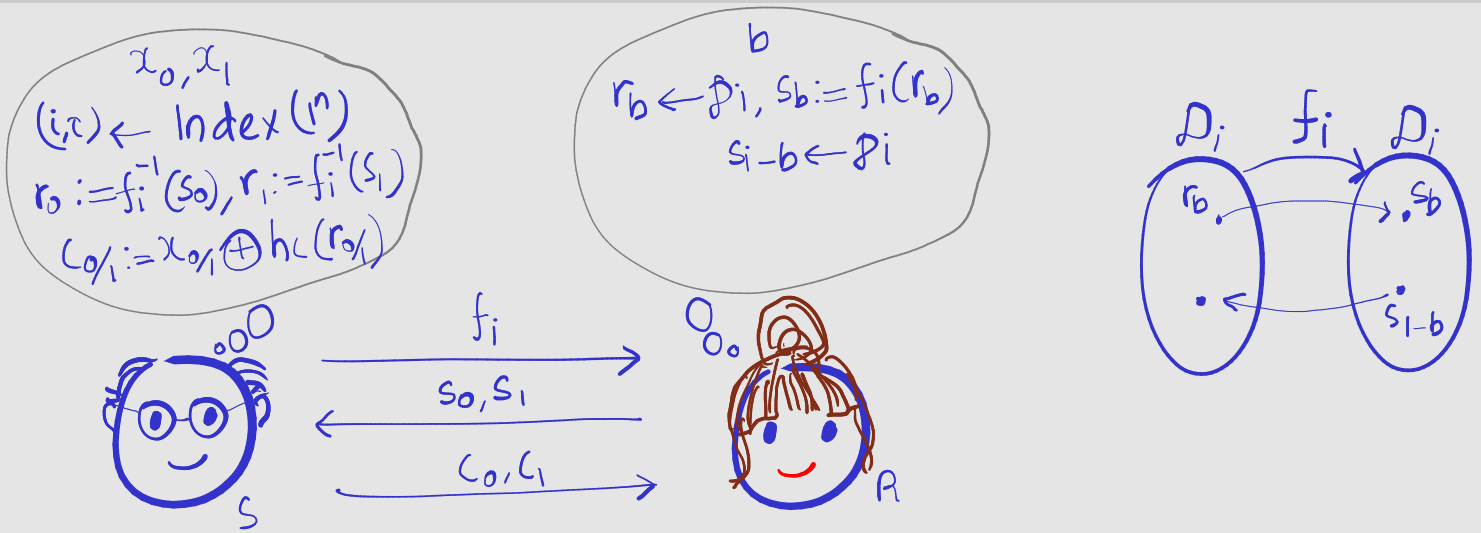


1. *Sender: sample* (f_i, f_i^{-1}) *and send* f_i *to receiver*
2. *Receiver:*
   1. *Sample* $r_b \leftarrow \mathcal{D}_i$ *and set* $s_b := f_i(r_b)$*; sample* $s_{1-b} \leftarrow \mathcal{D}_i$
   2. *Send* (s_0, s_1) *to sender*

TDP → OT...

How to tweak Construction 1 to get TDP → OT?

Protocol 1 (OT $\Pi = (S, R) \leftarrow$ TDP $f_i : \mathcal{D}_i \to \mathcal{D}_i$)

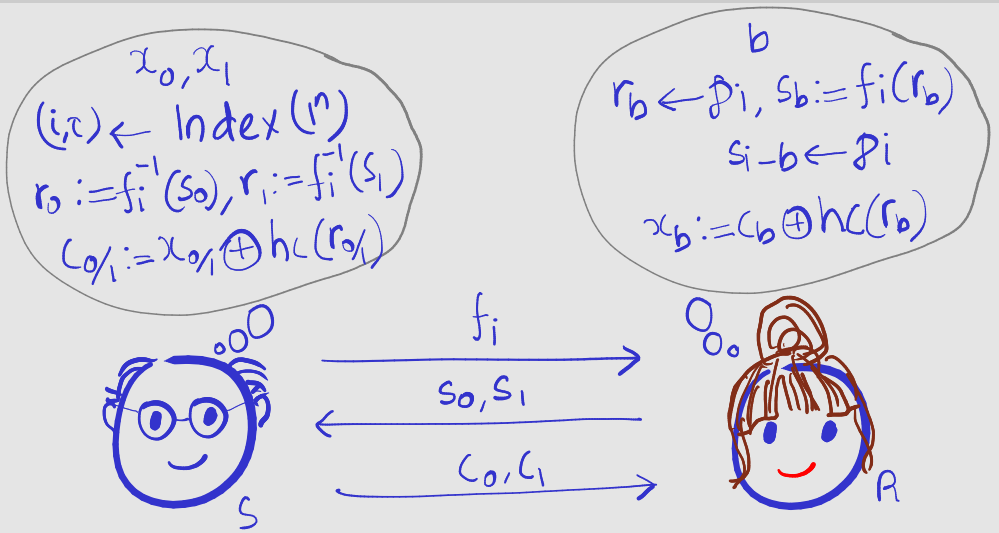


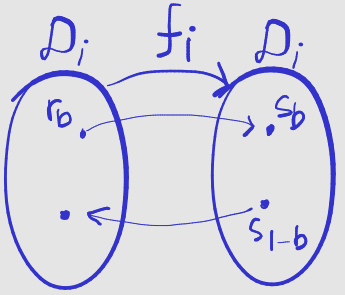
1. *Sender: sample* (f_i, f_i^{-1}) *and send* f_i *to receiver*
2. *Receiver:*
   1. *Sample* $r_b \leftarrow \mathcal{D}_i$ *and set* $s_b := f_i(r_b)$*; sample* $s_{1-b} \leftarrow \mathcal{D}_i$
   2. *Send* (s_0, s_1) *to sender*
3. *Sender:*
   1. *Compute* $r_0 := f_i^{-1}(s_0)$ *and* $r_1 := f_i^{-1}(s_1)$
   2. *Send* $(c_0 := x_0 \oplus \text{hc}(r_0), c_1 := x_1 \oplus \text{hc}(r_1))$

TDP → OT

How to tweak Construction 1 to get TDP → OT?

Protocol 1 (OT $\Pi = (S, R) \leftarrow$ TDP $f_i : \mathcal{D}_i \to \mathcal{D}_i$)





1. *Sender: sample* (f_i, f_i^{-1}) *and send* f_i *to receiver*
2. *Receiver:*
   1. *Sample* $r_b \leftarrow \mathcal{D}_i$ *and set* $s_b := f_i(r_b)$*; sample* $s_{1-b} \leftarrow \mathcal{D}_i$
   2. *Send* (s_0, s_1) *to sender*
3. *Sender:*
   1. *Compute* $r_0 := f_i^{-1}(s_0)$ *and* $r_1 := f_i^{-1}(s_1)$
   2. *Send* $(c_0 := x_0 \oplus \text{hc}(r_0), c_1 := x_1 \oplus \text{hc}(r_1))$
4. *Receiver: Output* $c_b \oplus \text{hc}(r_b)$

TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy). i) construct Sim_R ii) show real view ≈ simulation

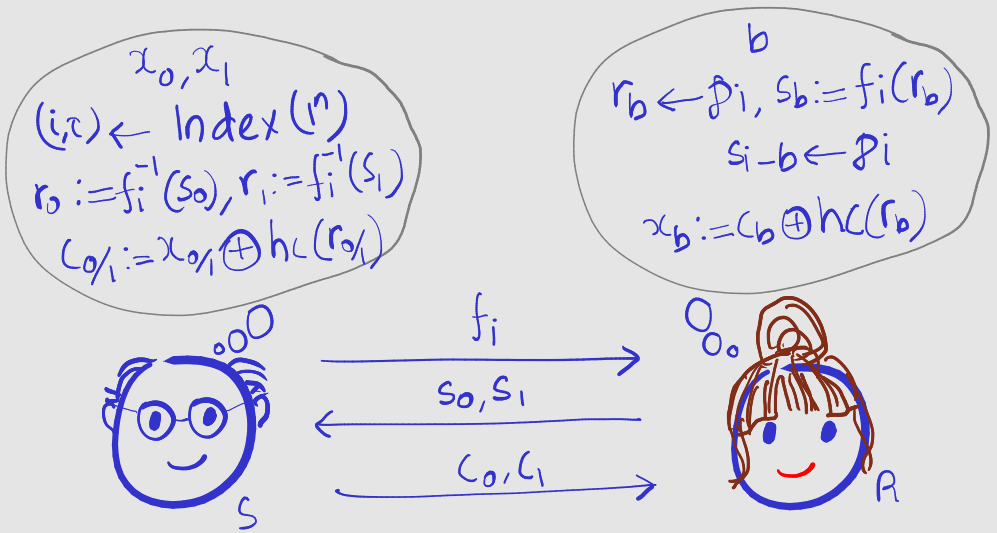
□

TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy). 1) construct Sim_R 2) show real view ≈ simulation



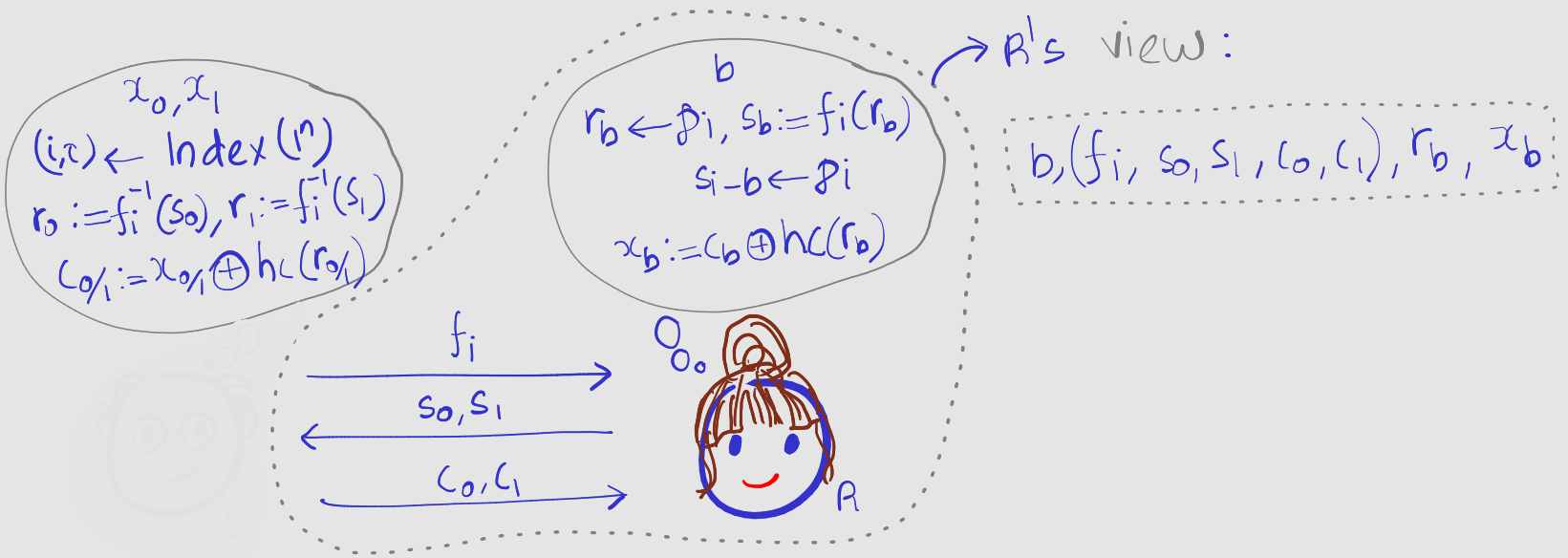
□

TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy).



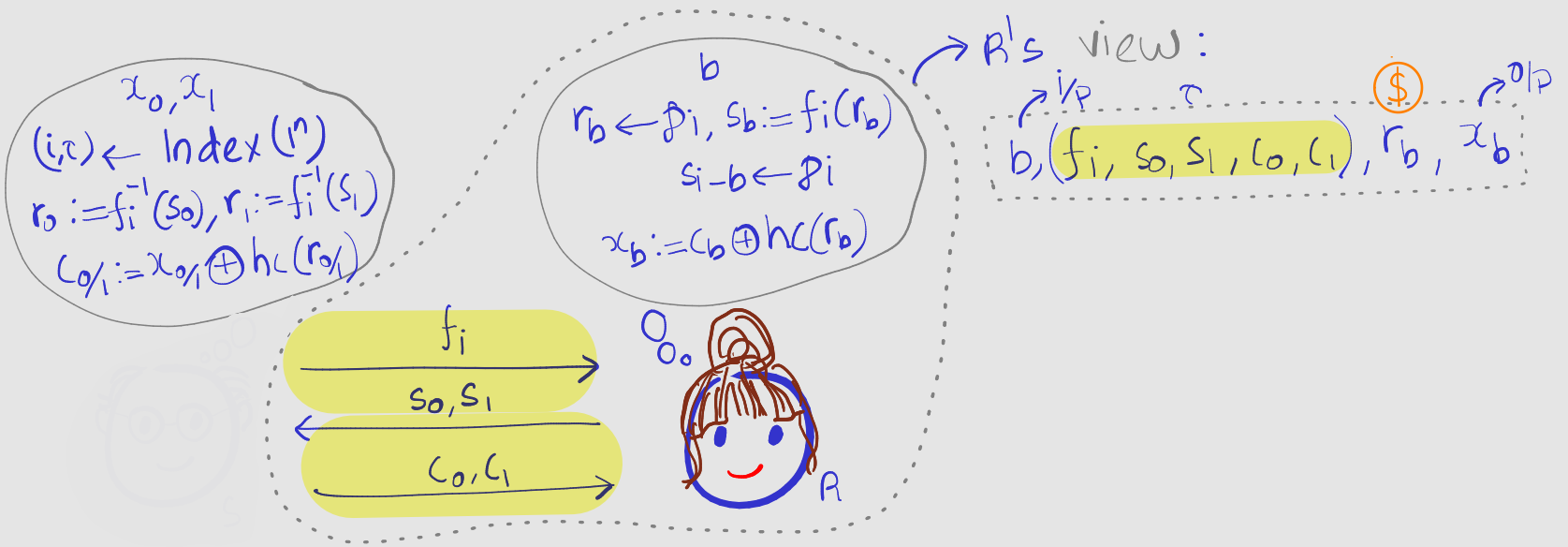
□

TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy).



□

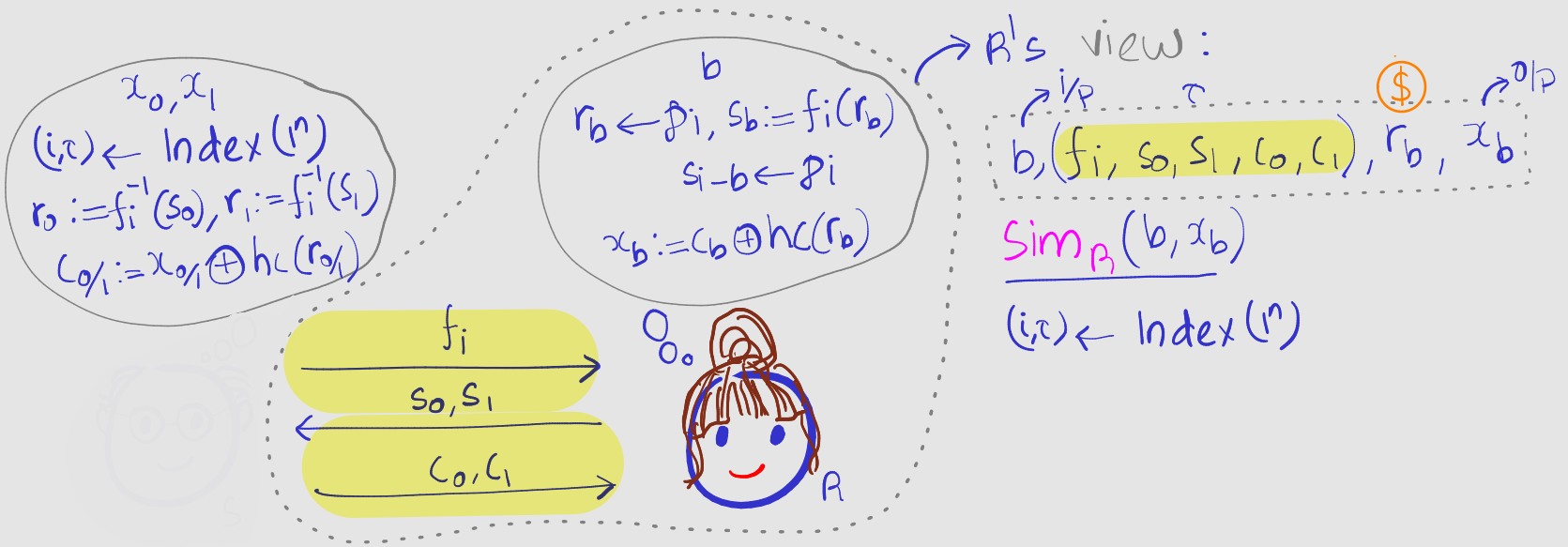
TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy).

1) Construct Sim_R 2) Show real view ≈ simulation



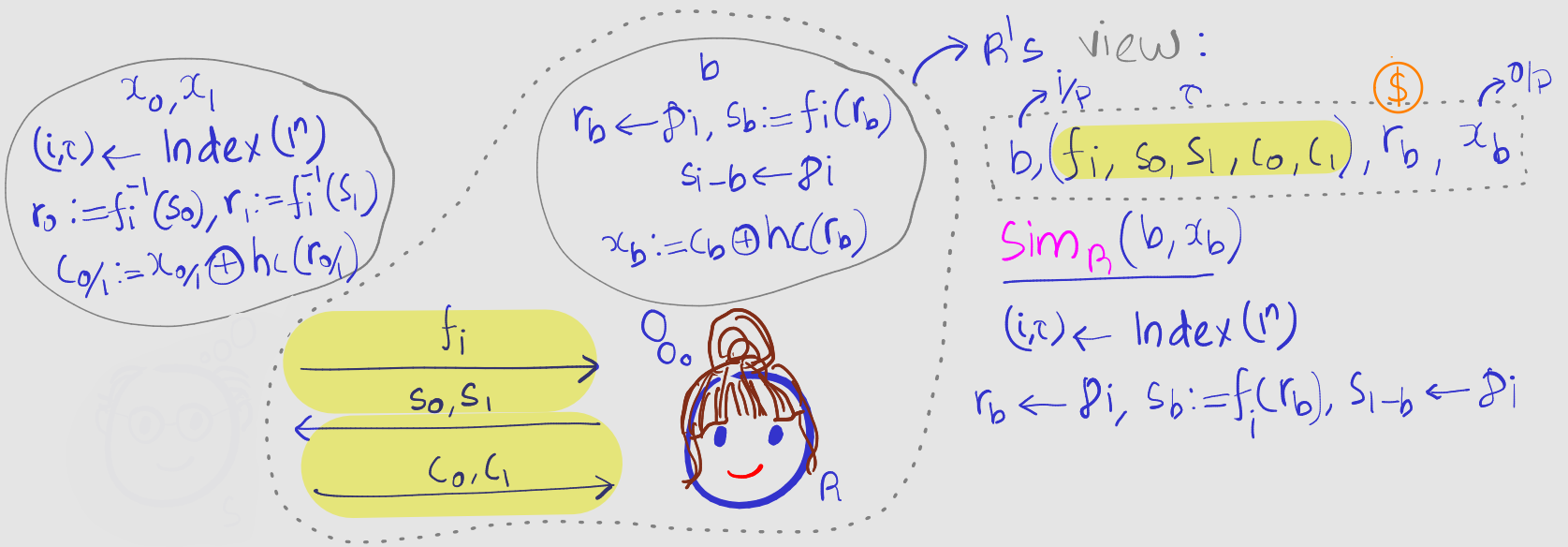
□

TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy).



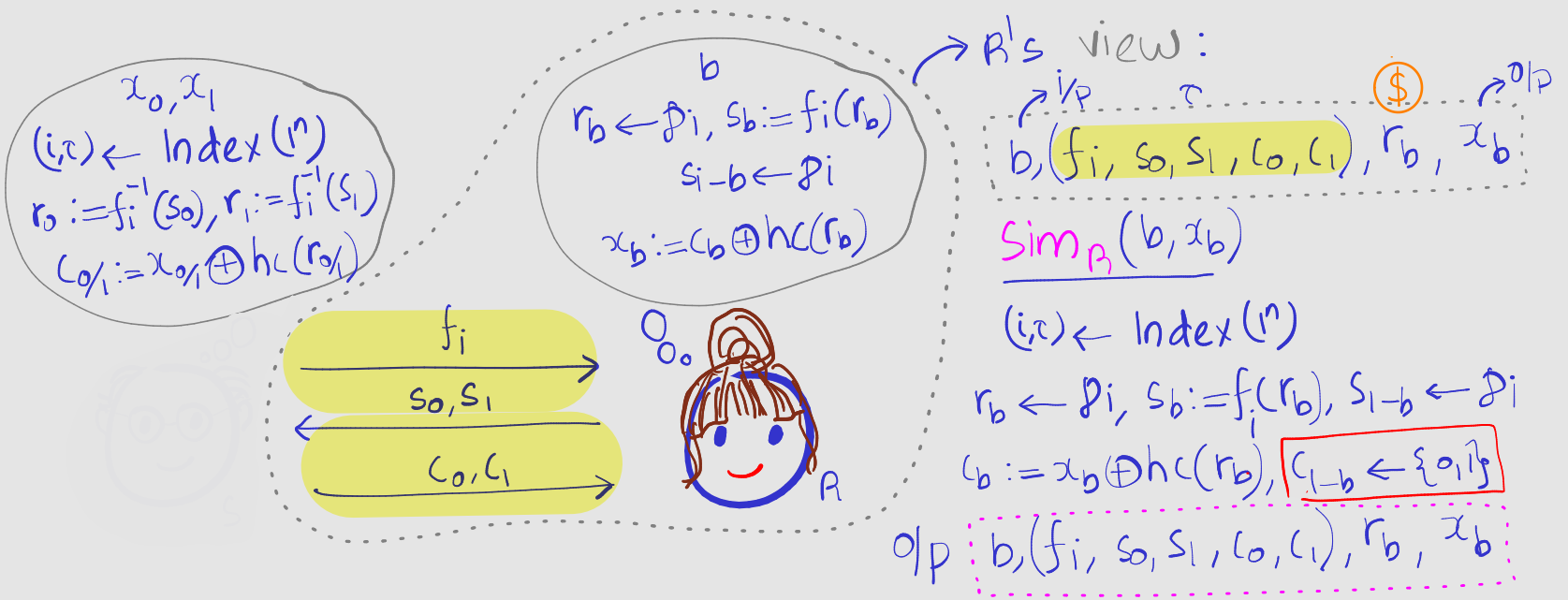
□

TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy).



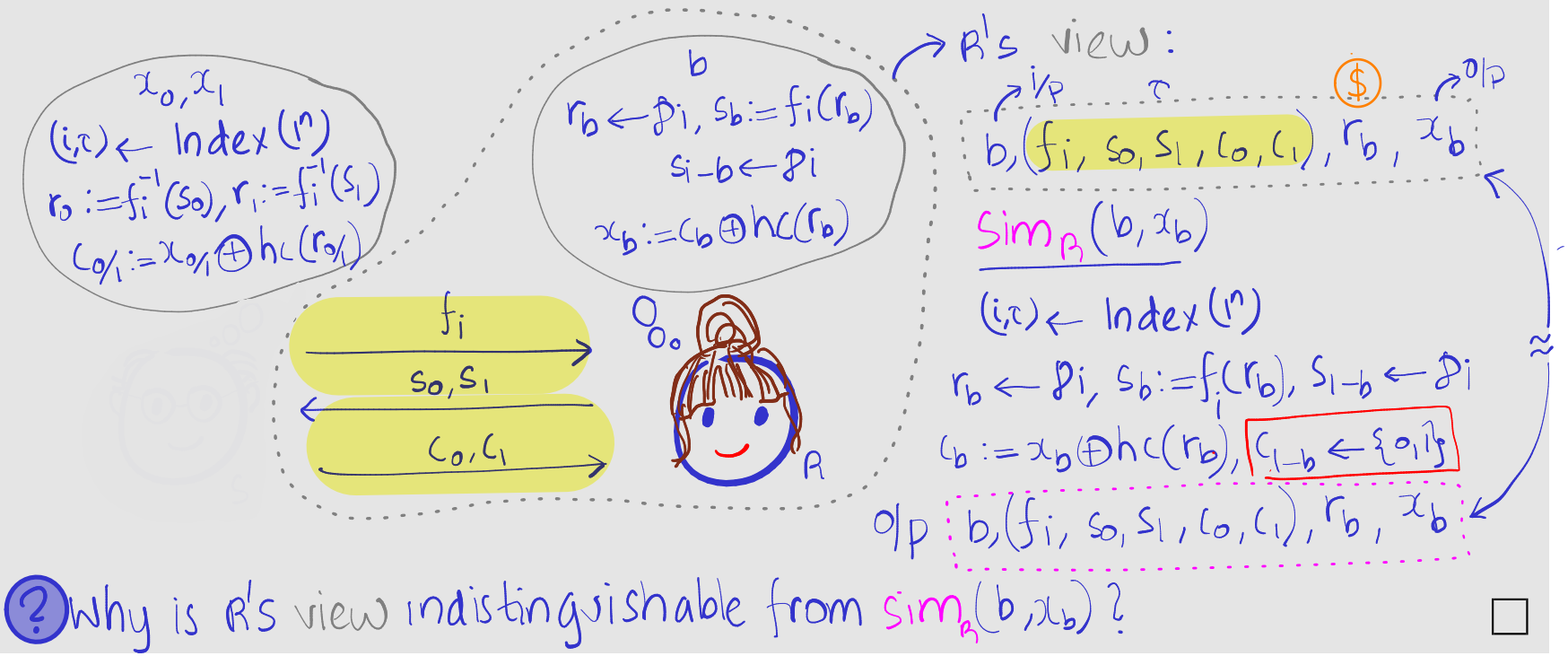
□

TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy). 1) Construct Sim_R 2) Show real view ≈ simulation



? Why is R's view indistinguishable from $Sim_R(b, x_b)$?

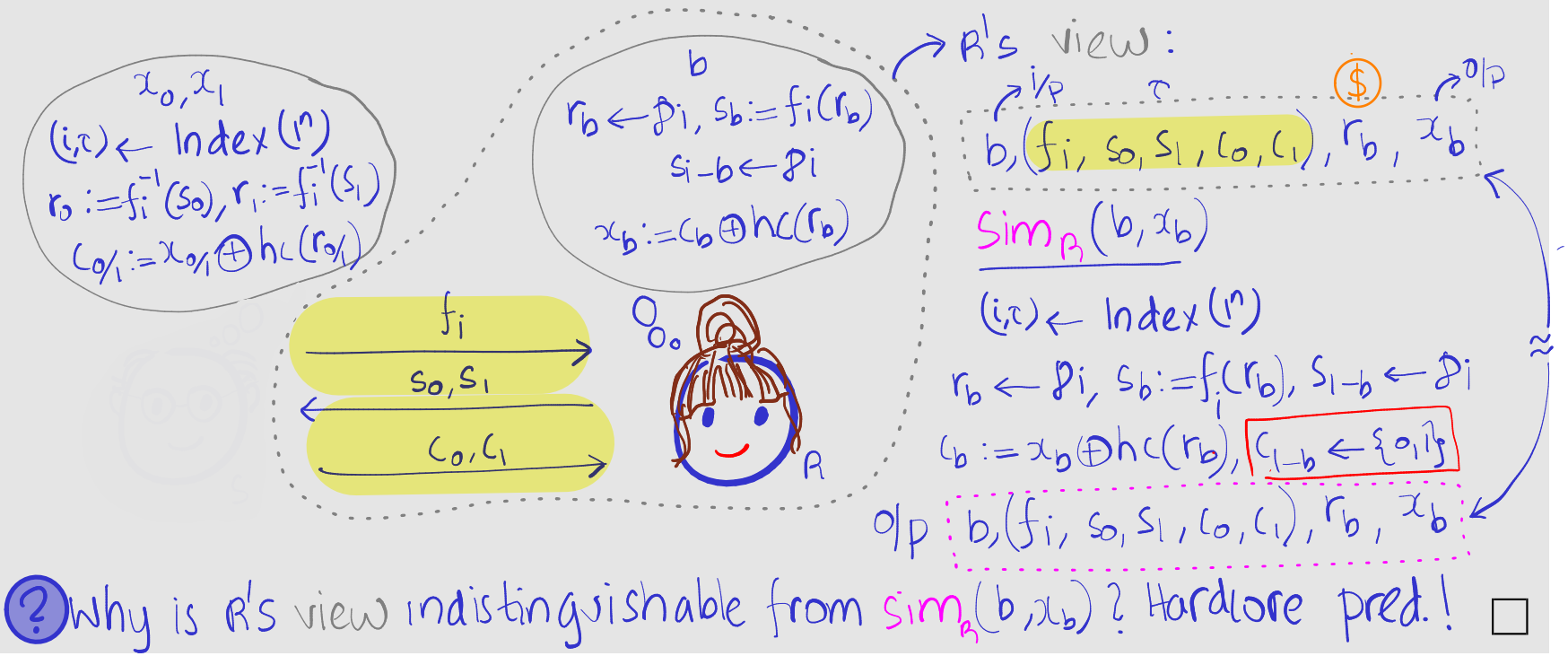
□

TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy). 1) Construct Sim_R 2) Show real view ≈ simulation



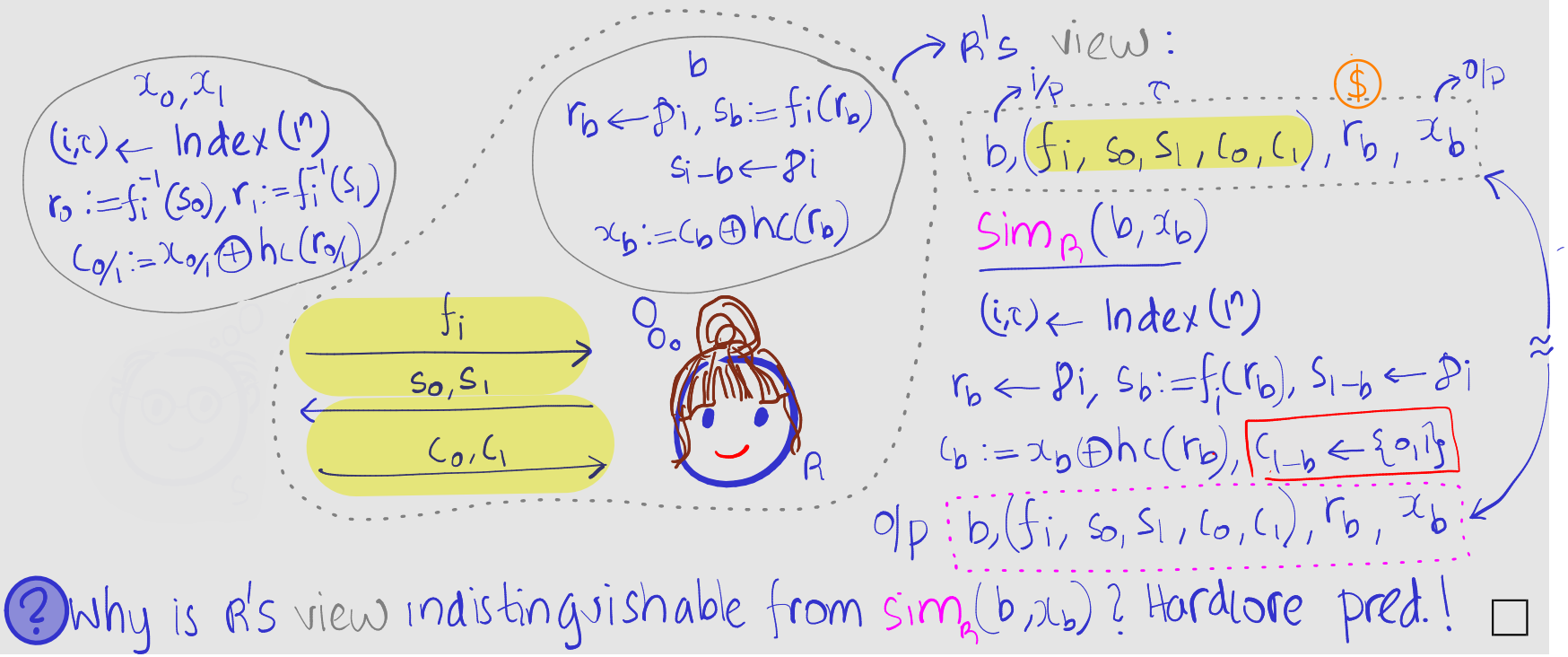
? Why is R's view indistinguishable from $Sim_R(b, x_b)$? Hardcore pred! □

TDP → OT...

Theorem 1

If f_i is a secure TDP then Π is a secure OT.

Proof (of sender privacy).



□

Exercise 3

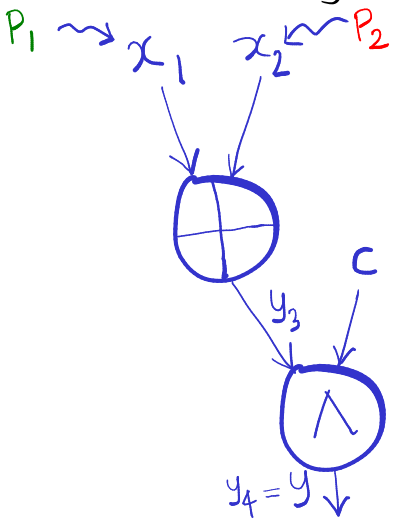
What happens if the sender (resp., receiver) is malicious?

Plan for Today's Lecture

1. Computing $\wedge$ with Perfect Privacy is Impossible
2. New Cryptographic Primitive: Oblivious Transfer (OT)
3. Goldreich-Micali-Wigderson (GMW) Protocol

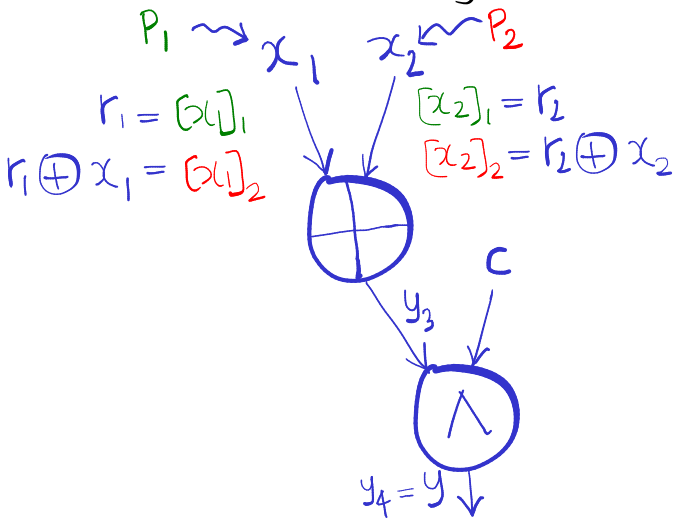
Recall Π (Protocol 1) from Last Lecture

- Every linear function f over $\mathbb{F}_2 = (\mathbb{Z}_2, \oplus, \wedge)$ can be represented by a Boolean circuit consisting of $\oplus$ and $\wedge_c$



Recall Π (Protocol 1) from Last Lecture

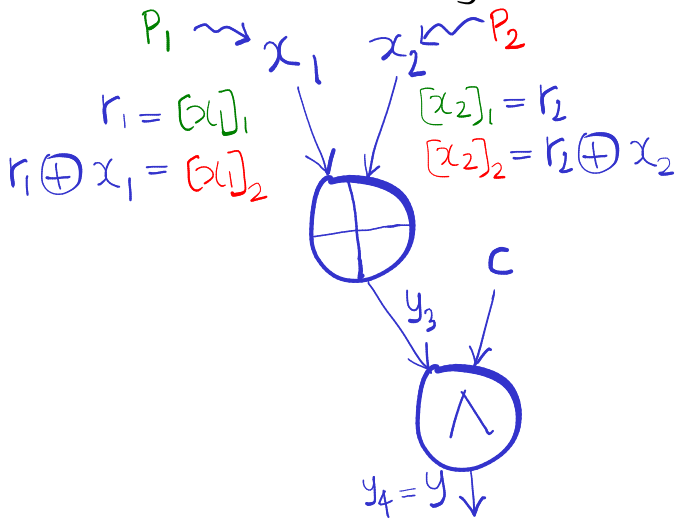
- Every linear function f over $\mathbb{F}_2 = (\mathbb{Z}_2, \oplus, \wedge)$ can be represented by a Boolean circuit consisting of $\oplus$ and $\wedge_c$



- Summary of Π (simplified for 2PC over $\mathbb{F}_2$):
  - Each party secret-shares its input bits with the other party

Recall Π (Protocol 1) from Last Lecture

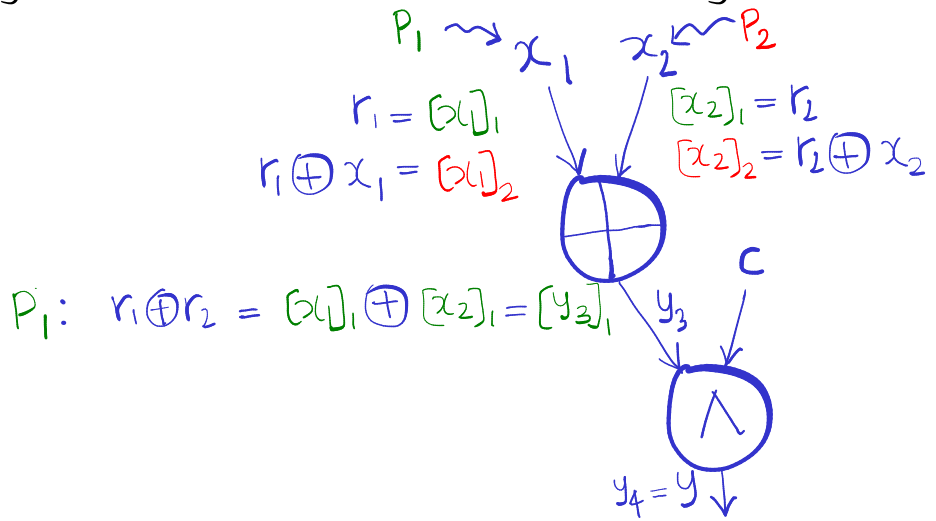
- Every linear function f over $\mathbb{F}_2 = (\mathbb{Z}_2, \oplus, \wedge)$ can be represented by a Boolean circuit consisting of $\oplus$ and $\wedge_c$



- Summary of Π (simplified for 2PC over $\mathbb{F}_2$):
  - Each party secret-shares its input bits with the other party
  - Invariant: each party P_i will have secret share $[y_w]_i$ of value y_w of wire w

Recall Π (Protocol 1) from Last Lecture

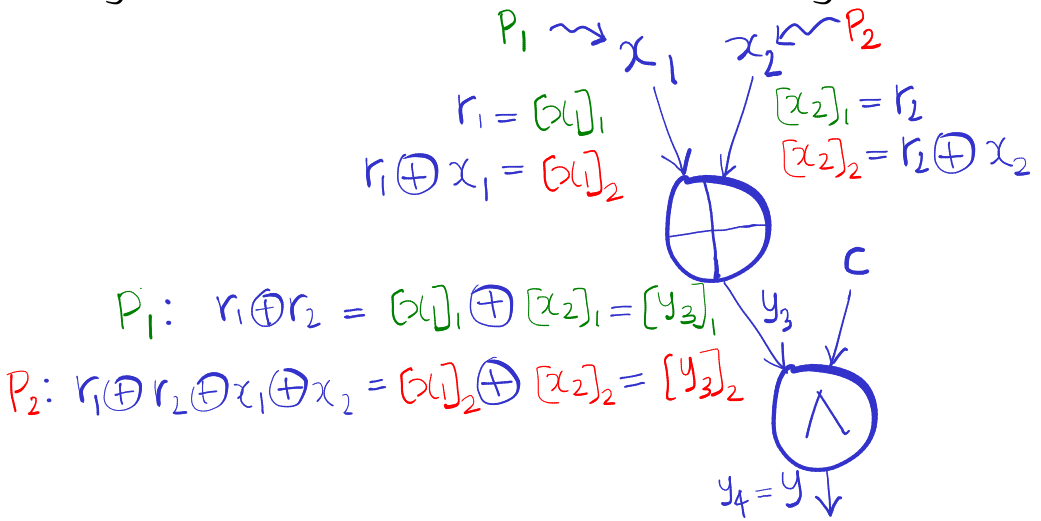
- Every linear function f over $\mathbb{F}_2 = (\mathbb{Z}_2, \oplus, \wedge)$ can be represented by a Boolean circuit consisting of $\oplus$ and $\wedge_c$



- Summary of Π (simplified for 2PC over $\mathbb{F}_2$):
  - Each party secret-shares its input bits with the other party
  - Invariant: each party P_i will have secret share $[y_w]_i$ of value y_w of wire w
  - Both parties compute locally "over shares"
    - $\oplus$ gate: XOR shares of i/p wires to obtain share of o/p wire
    - $\wedge_c$ gate: AND share of i/p wire with c to get obtain of o/p of $\wedge_c$

Recall Π (Protocol 1) from Last Lecture

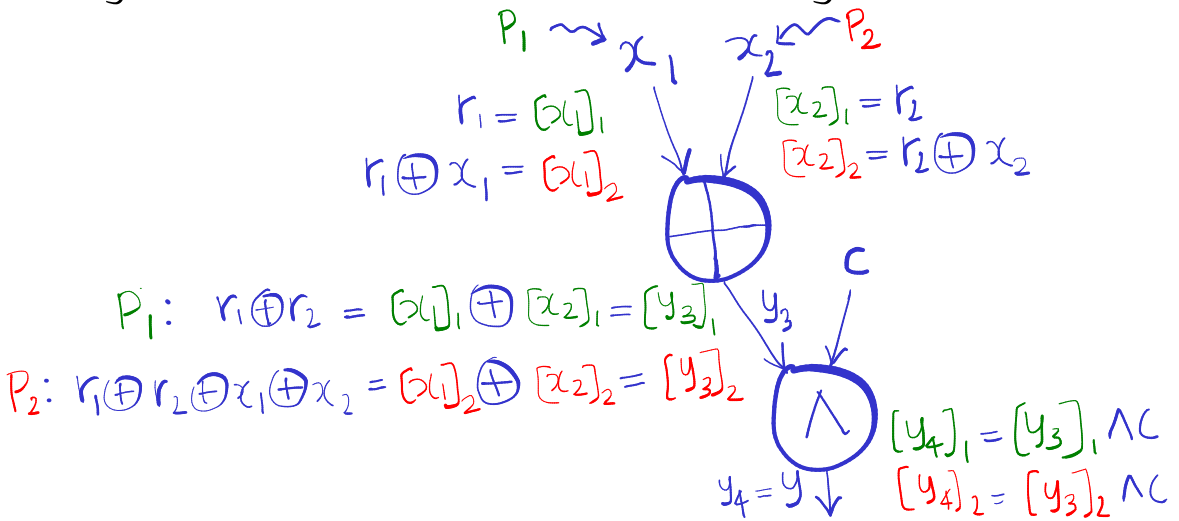
- Every linear function f over $\mathbb{F}_2 = (\mathbb{Z}_2, \oplus, \wedge)$ can be represented by a Boolean circuit consisting of $\oplus$ and $\wedge_c$



- Summary of Π (simplified for 2PC over $\mathbb{F}_2$):
  - Each party secret-shares its input bits with the other party
  - Invariant: each party P_i will have secret share $[y_w]_i$ of value y_w of wire w
  - Both parties compute locally "over shares"
    - $\oplus$ gate: XOR shares of i/p wires to obtain share of o/p wire
    - $\wedge_c$ gate: AND share of i/p wire with c to get obtain of o/p of $\wedge_c$

Recall Π (Protocol 1) from Last Lecture

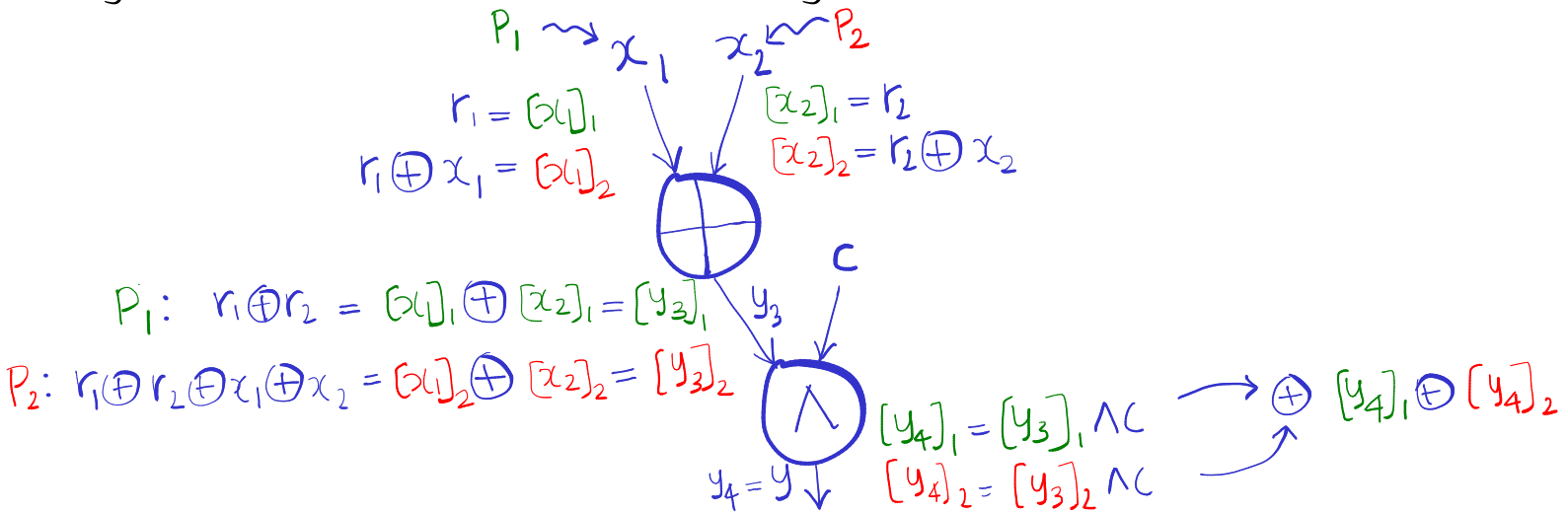
- Every linear function f over $\mathbb{F}_2 = (\mathbb{Z}_2, \oplus, \wedge)$ can be represented by a Boolean circuit consisting of $\oplus$ and $\wedge_c$



- Summary of Π (simplified for 2PC over $\mathbb{F}_2$):
  - Each party secret-shares its input bits with the other party
  - Invariant: each party P_i will have secret share $[y_w]_i$ of value y_w of wire w
  - Both parties compute locally "over shares"
    - $\oplus$ gate: XOR shares of i/p wires to obtain share of o/p wire
    - $\wedge_c$ gate: AND share of i/p wire with c to get obtain of o/p of $\wedge_c$

Recall Π (Protocol 1) from Last Lecture

- Every linear function f over $\mathbb{F}_2 = (\mathbb{Z}_2, \oplus, \wedge)$ can be represented by a Boolean circuit consisting of $\oplus$ and $\wedge_c$



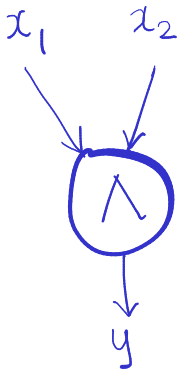
- Summary of Π (simplified for 2PC over $\mathbb{F}_2$):
  - Each party secret-shares its input bits with the other party
  - Invariant: each party P_i will have secret share $[y_w]_i$ of value y_w of wire w
  - Both parties compute locally "over shares"
    - $\oplus$ gate: XOR shares of i/p wires to obtain share of o/p wire
    - $\wedge_c$ gate: AND share of i/p wire with c to get obtain of o/p of $\wedge_c$
  - P_1 and P_2 use their shares of output wire to reconstruct output

Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant

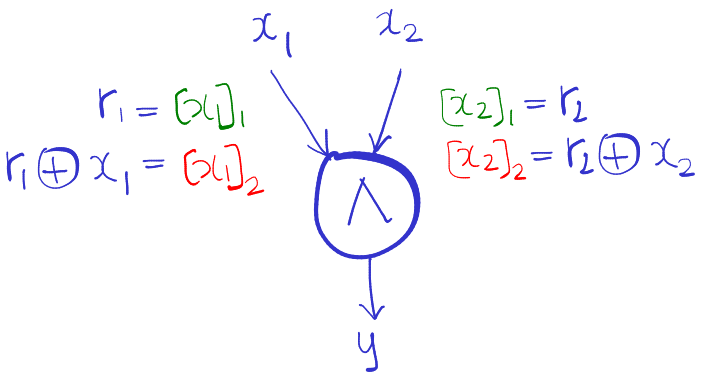
Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?



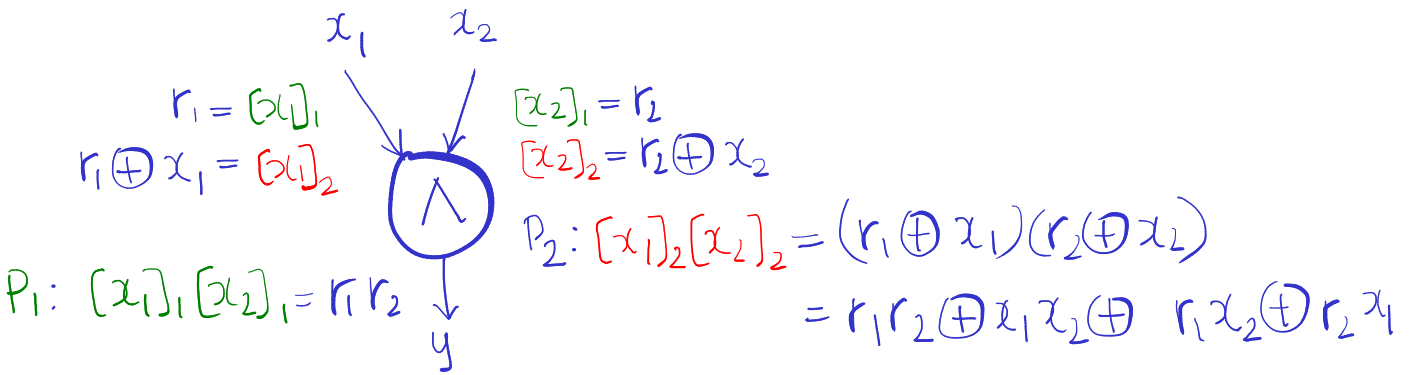
Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?



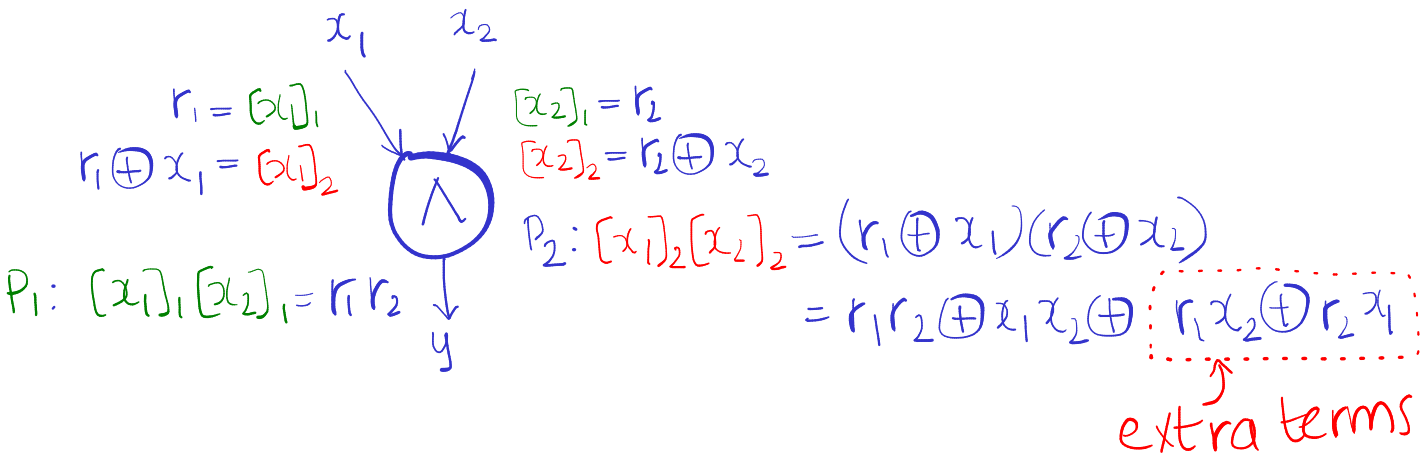
Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?



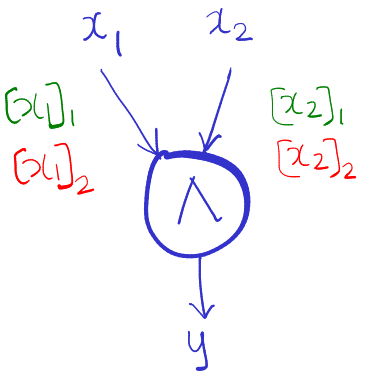
Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?



Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?

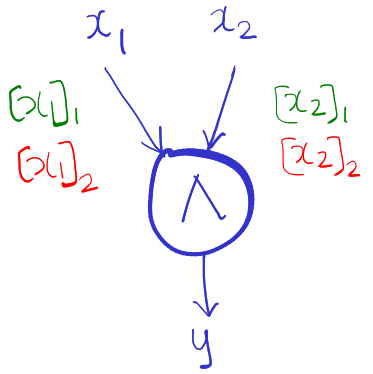


1-out-of-4

? Can you reduce secret-shared $\wedge$ to an OT?

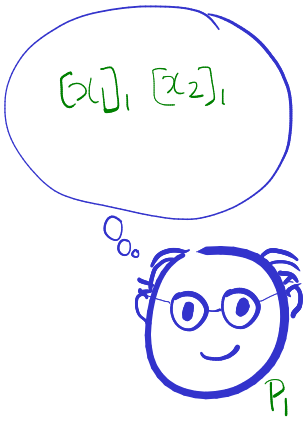
Let's use OT to Compute $\wedge$ with Computational Privacy

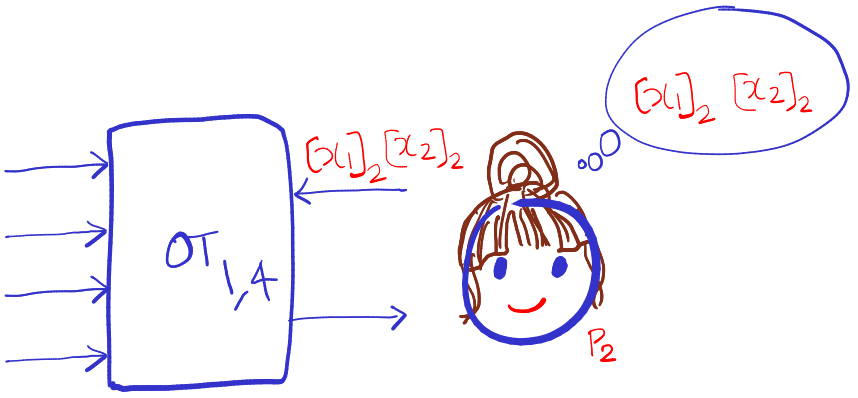
- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?



1-out-of-4 $\hat{\wedge}$ OT

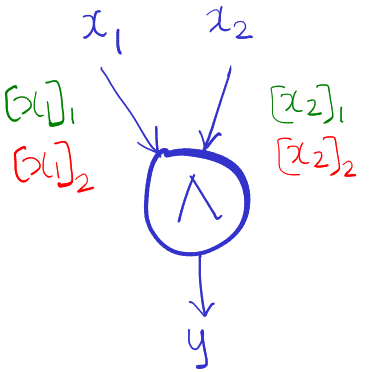
? Can you reduce secret-shared $\wedge$ to an $\hat{\wedge}$OT?





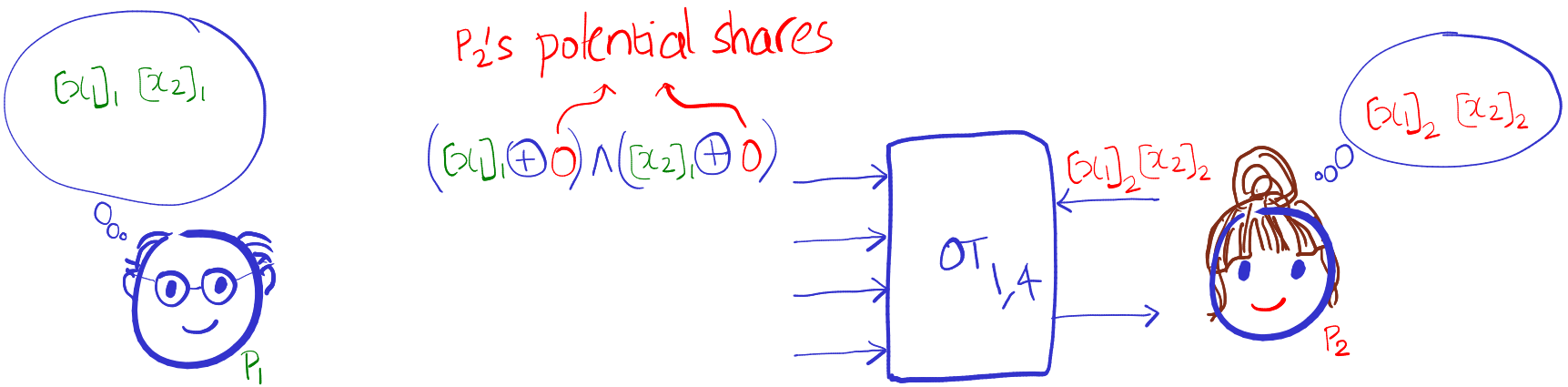
Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?



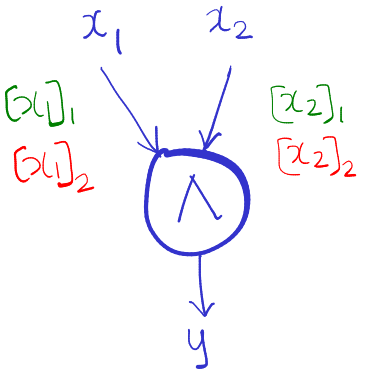
1-out-of-4 $\hat{\wedge}$ OT

? Can you reduce secret-shared $\wedge$ to an $\hat{\wedge}$OT?



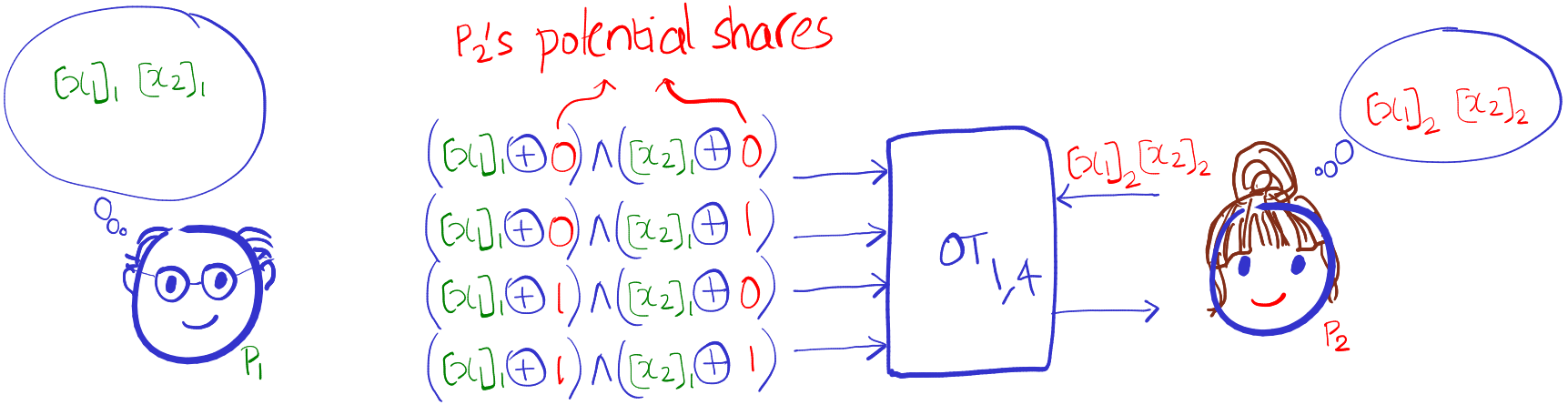
Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?



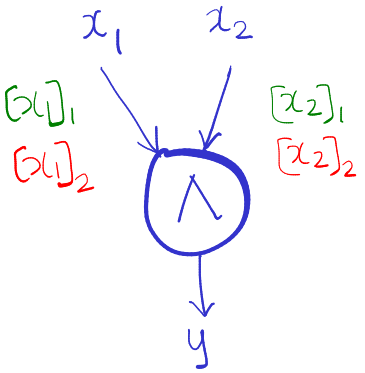
1-out-of-4 $\hat{\wedge}$ OT

? Can you reduce secret-shared $\wedge$ to an $\hat{\wedge}$OT?



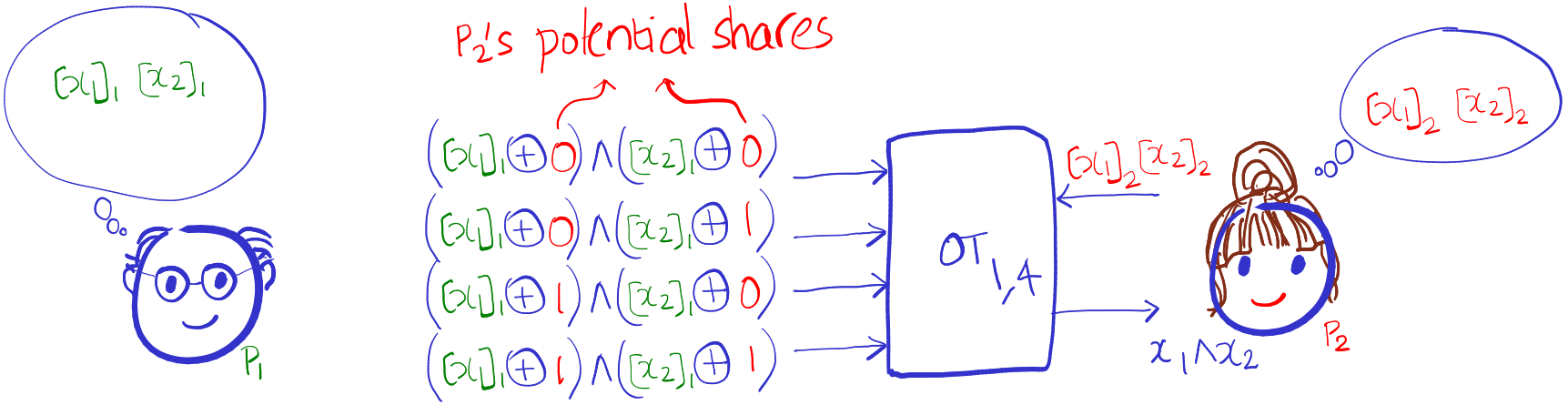
Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?



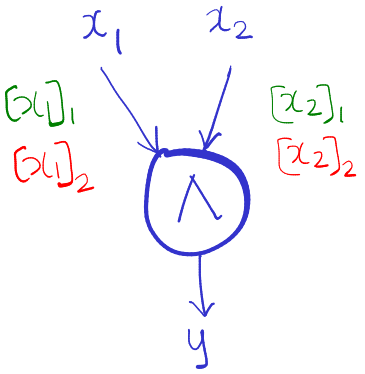
1-out-of-4 $\hat{\wedge}$ OT

? Can you reduce secret-shared $\wedge$ to an $\hat{\wedge}$OT?



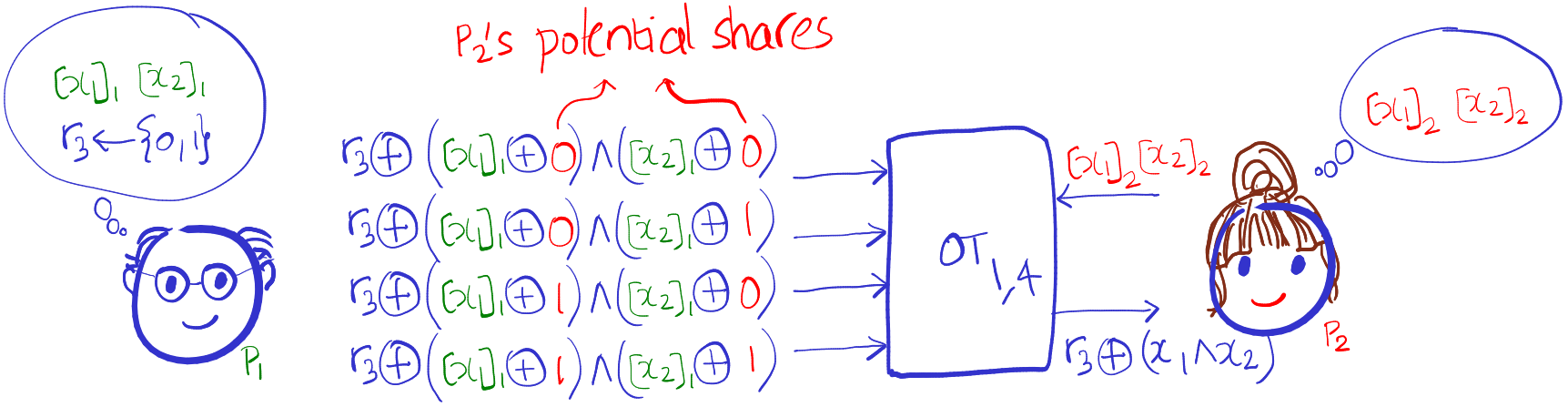
Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?



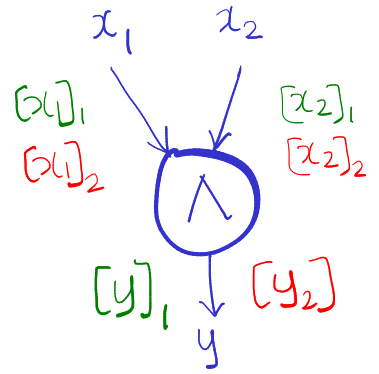
1-out-of-4 $\hat{\wedge}$ OT

? Can you reduce secret-shared $\wedge$ to an $\hat{\wedge}$OT?



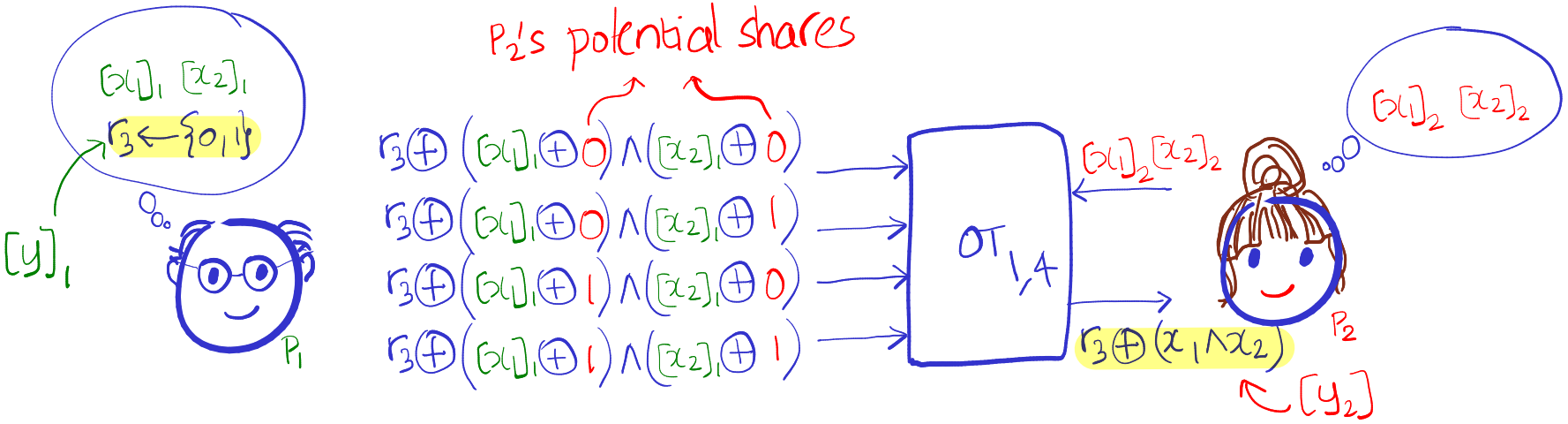
Let's use OT to Compute $\wedge$ with Computational Privacy

- Hurdle with extending Π to arbitrary functions?
  - Need to deal with $\wedge$ to maintain invariant
- What happens when shares of input wires of $\wedge$ locally ANDed?

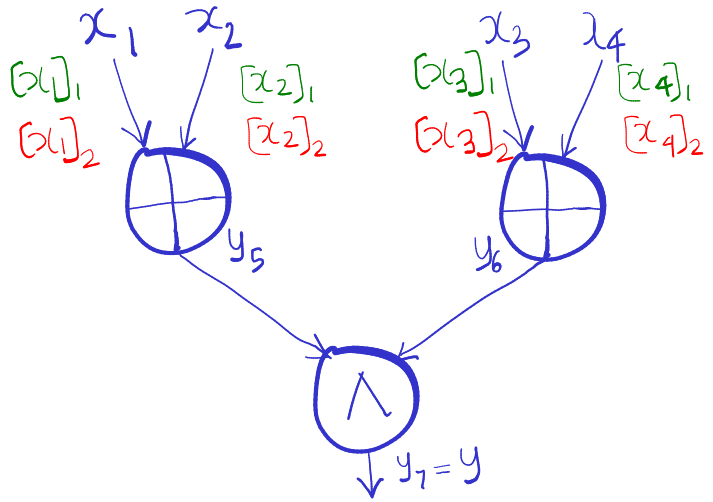


1-out-of 4

? Can you reduce secret-shared $\wedge$ to an OT?



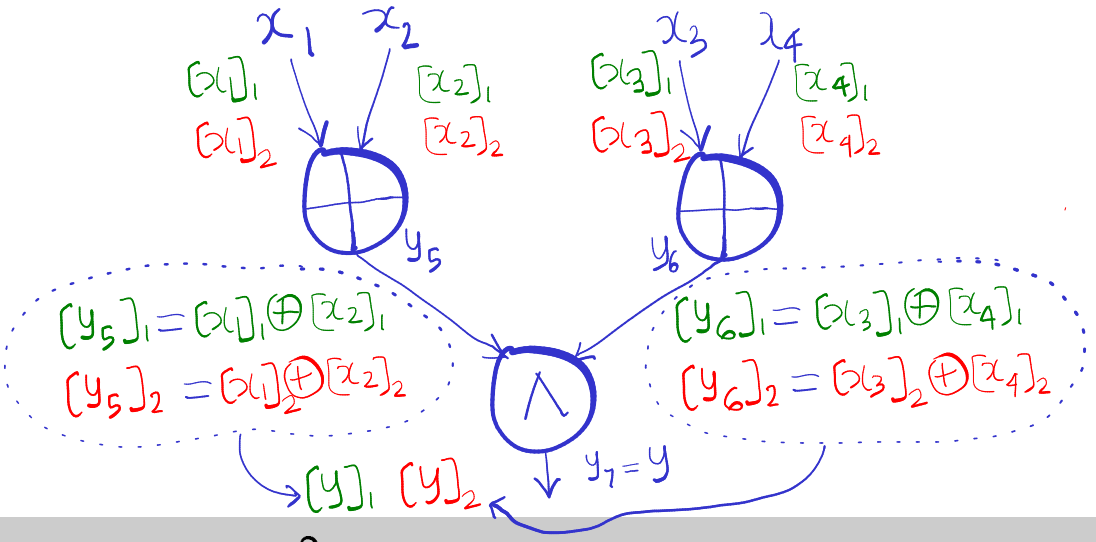
GMW Protocol...



Protocol 2 (for $f : \{0,1\}^{2n} \to \{0,1\}$ represented by circuit C)

1. *Each party secret-shares its input bits with the other party*

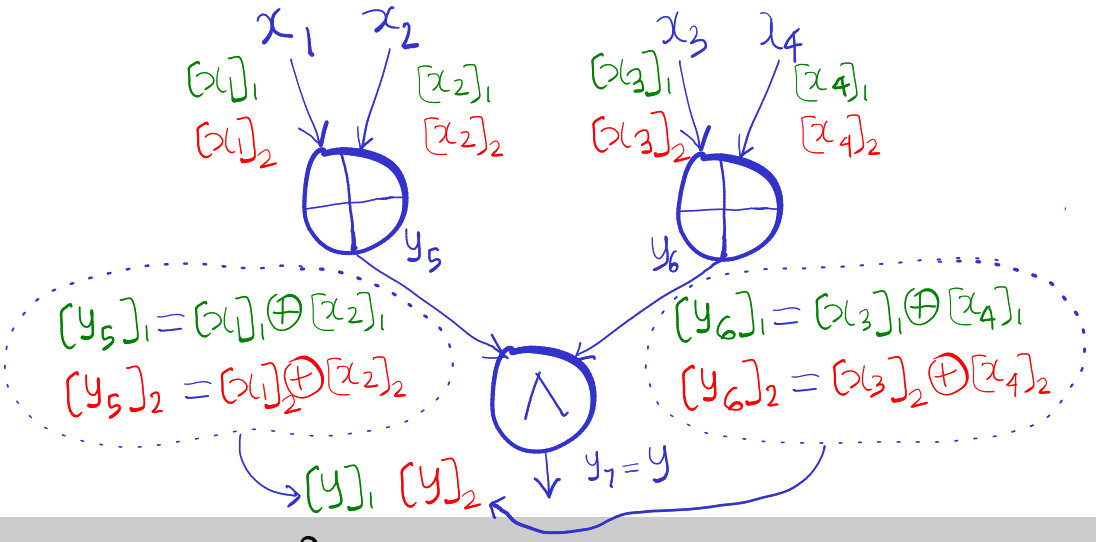
GMW Protocol...



Protocol 2 (for $f : \{0,1\}^{2n} \to \{0,1\}$ represented by circuit C)

1. *Each party secret-shares its input bits with the other party*
2. *Emulate circuit: for each gate $g \in C$ in topological order*
   - *if $g = \oplus$: P_1 locally XORs its shares of g's input wires to obtain its share of g's output wire; P_2 does the same*
   - *if $g = \wedge$: P_1 (sender) and P_2 (receiver) use 1-out-of-4 OT protocol to generate their shares of g's output wire*

GMW Protocol...



Protocol 2 (for $f : \{0,1\}^{2n} \to \{0,1\}$ represented by circuit C)

1. *Each party secret-shares its input bits with the other party*
2. *Emulate circuit: for each gate $g \in C$ in topological order*
   - *if $g = \oplus$: P_1 locally XORs its shares of g's input wires to obtain its share of g's output wire; P_2 does the same*
   - *if $g = \wedge$: P_1 (sender) and P_2 (receiver) use 1-out-of-4 OT protocol to generate their shares of g's output wire*
3. *P_1 and P_2 use their shares of output wire to reconstruct output*

GMW Protocol...

Theorem 2

OT secure $\Rightarrow$ *GMW protocol computes* f *with computational privacy.*

GMW Protocol...

Theorem 2

OT secure $\Rightarrow$ *GMW protocol computes* f *with computational privacy.*

Exercise 4

- *Prove* P_1*'s privacy: OT simulator* $\rightarrow$ *simulator* Sim_{P_2} *for* P_2.
- *Prove* P_2*'s privacy: simulator* Sim_{P_1} *for* P_1.

To Recap Today's Lecture

- Perfectly-private 2PC for *general* functions is impossible!
  - Counter-example: $\wedge$

To Recap Today's Lecture

- Perfectly-private 2PC for *general* functions is impossible!
  - Counter-example: $\wedge$
- What did we do?

To Recap Today's Lecture

- Perfectly-private 2PC for *general* functions is impossible!
  - Counter-example: $\wedge$
- What did we do? Relax to *computational* privacy
  - New cryptographic primitive: oblivious transfer (OT)
    - Trapdoor permutation (TDP) $\rightarrow$ OT

To Recap Today's Lecture

- Perfectly-private 2PC for *general* functions is impossible!
  - Counter-example: $\wedge$
- What did we do? Relax to *computational* privacy
  - New cryptographic primitive: oblivious transfer (OT)
    - Trapdoor permutation (TDP) $\rightarrow$ OT
- GMW protocol: OT $\rightarrow$ computationally-private 2PC for general functions over $\mathbb{F}_2$

To Recap Today's Lecture

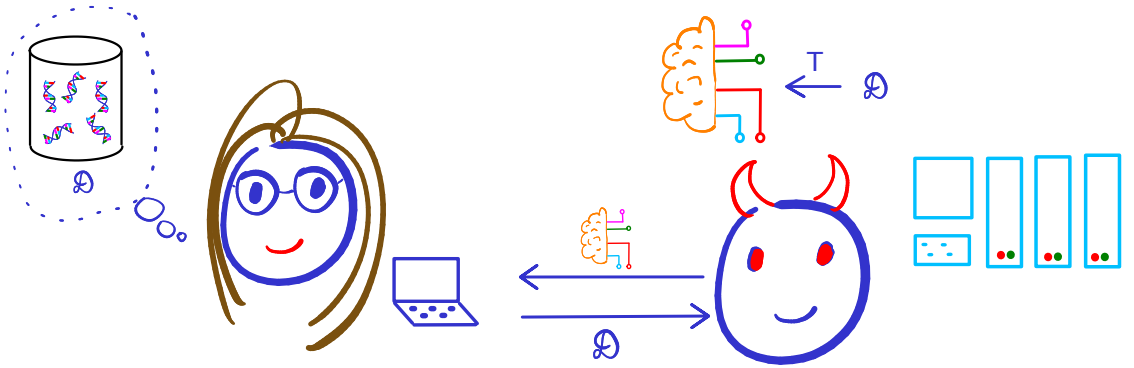
- Perfectly-private 2PC for *general* functions is impossible!
  - Counter-example: $\wedge$
- What did we do? Relax to *computational* privacy
  - New cryptographic primitive: oblivious transfer (OT)
    - Trapdoor permutation (TDP) $\rightarrow$ OT
- GMW protocol: OT $\rightarrow$ computationally-private 2PC for general functions over $\mathbb{F}_2$
  - Alternatively, Yao's garbling from OT and SKE
  - Can be extended to computationally-private MPC for general functions over $\mathbb{F}_p$

To Recap Today's Lecture

- Perfectly-private 2PC for *general* functions is impossible!
  - Counter-example: $\wedge$
- What did we do? Relax to *computational* privacy
  - New cryptographic primitive: oblivious transfer (OT)
    - Trapdoor permutation (TDP) $\rightarrow$ OT
- GMW protocol: OT $\rightarrow$ computationally-private 2PC for general functions over $\mathbb{F}_2$
  - Alternatively, Yao's garbling from OT and SKE
  - Can be extended to computationally-private MPC for general functions over $\mathbb{F}_p$
- For privacy against malicious parties, use zero knowledge proof (ZKP)
  - Semi-honest-secure MPC + ZKP $\rightarrow$ malicious-secure MPC

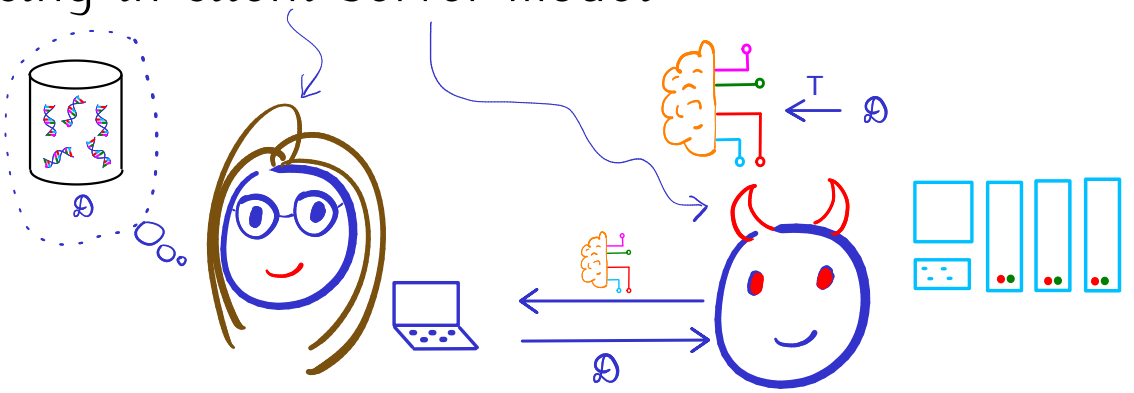
Next Two Lectures

- Outsourcing in client–server model



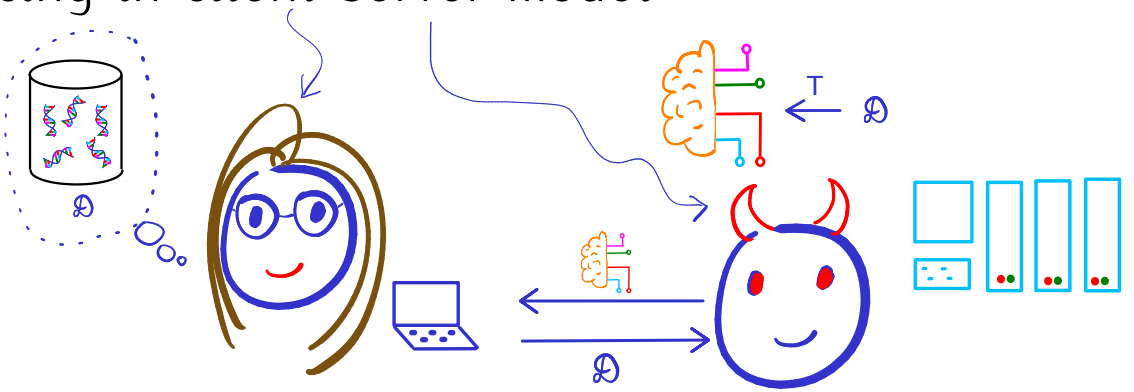
Next Two Lectures

- Outsourcing in client–server model



Next Two Lectures

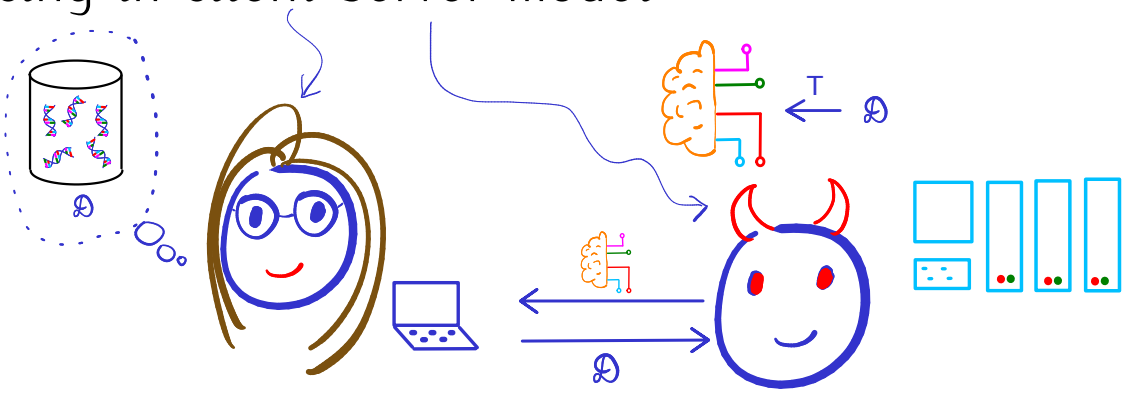
- Outsourcing in client–server model



- Privacy-preserving outsourcing
  - Using fully homomorphic encryption (FHE)
  - LWE$\rightarrow$ FHE (GSW construction)

Next Two Lectures

- Outsourcing in client-server model



- Privacy-preserving outsourcing
  - Using fully homomorphic encryption (FHE)
  - LWE$\rightarrow$ FHE (GSW construction)

- Verifiable outsourcing
  - Using succinct non-interactive argument (SNARG)
  - SNARG for repeated squaring problem in ROM
    - Pietrzak's protocol
  - SNARG for **NP** in ROM (if time permits)
    - Kilian's protocol

References

1. Most of this lecture is based on (slides of) Lecture 18 from Vinod Vaikuntanathan's MIT6875. For a more formal description of the protocols, see [Gol04, §7.3].
2. Claim 1 is folklore. The proof presented here is due to Rotem Oshman (and taken from the slides above)
3. Oblivious transfer was introduced by Rabin [Rab81] (although Wiesner introduced a similar primitive in [Wie83]). Its construction from TDP is from [EGL82].
4. The GMW protocol is from [GMW87]


Shimon Even, Oded Goldreich, and Abraham Lempel.

A randomized protocol for signing contracts.

In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 205–210. Plenum Press, New York, USA, 1982.


Oded Goldreich, Silvio Micali, and Avi Wigderson.

How to play any mental game or A completeness theorem for protocols with honest majority.

In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.


Oded Goldreich.

The Foundations of Cryptography - Volume 2: Basic Applications.

Cambridge University Press, 2004.


Michael Rabin.

How to exchange secrets with oblivious transfer.

Technical report, 1981.


Stephen Wiesner.

Conjugate coding.

SIGACT News, 15(1):78–88, January 1983.