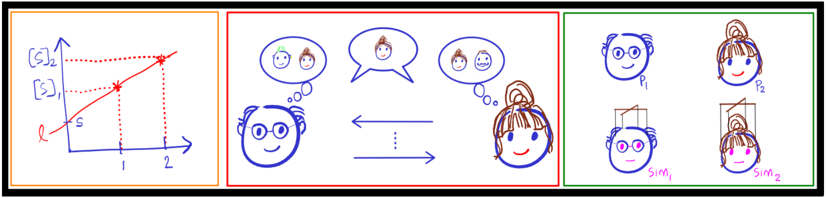


CS783: Theoretical Foundations of Cryptography

Lecture 17 (08/Oct/24)

Instructor: Chethan Kamath

Recall from Last Three Lectures

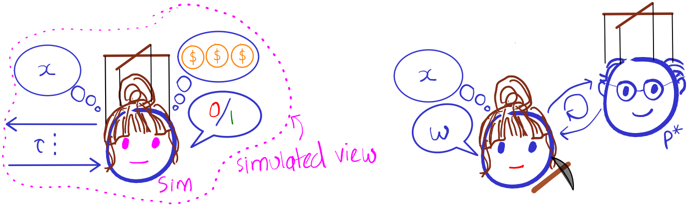
- Interactive proof
- Zero knowledge (ZK) proof
- ZK proof of knowledge

Recall from Last Three Lectures

- Interactive proof
- Zero knowledge (ZK) proof
- ZK proof of knowledge

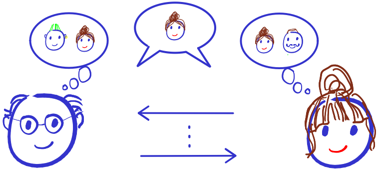
Recall from Last Three Lectures

- Interactive proof
- Zero knowledge (ZK) proof
- ZK proof of knowledge



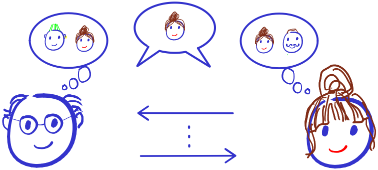
- Simulators and extractors
  - Key tools: out-of-order sampling and rewinding

- Main topic of Module III: private computation of functions



Plan for Today's Lecture...

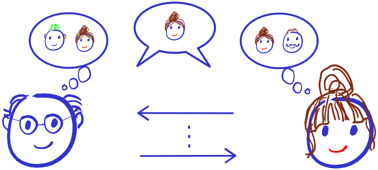
- Main topic of Module III: private computation of functions



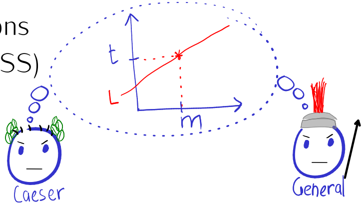
- Define syntax and security for the *two-party* case (2PC)
  - Extends the simulation paradigm

Plan for Today's Lecture...

- Main topic of Module III: private computation of functions



- Define syntax and security for the *two-party* case (2PC)
  - Extends the simulation paradigm
- Perfectly-private 2PC for *linear* functions
  - Key tool: threshold secret sharing (TSS)
  - Shamir's TSS



Plan for Today's Lecture...

General *template*:

1. Identify the task private computation of functions
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

General *template*:

private computation of functions

semi-honest model

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?

"static corruption"

"perfect privacy"

3. Construct a scheme Π
4. Formally prove that Π in secure in model M

General *template*:

1. Identify the task private computation of linear functions
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? "perfect privacy"
3. Construct a scheme Π secret-sharing-based
4. Formally prove that Π in secure in model M

   construct simulator

Private computation of functions is useful!

Apple's PSI

The Apple PSI System

Abhishek Bhowmick, Apple Inc.
Dan Boneh, Stanford University
Steve Myers, Apple Inc.
Kunal Talwar, Apple Inc.
Karl Tarbe, Apple Inc.

July 29, 2021

Mozilla's private aggregation

Mozilla Security Blog

Next steps in privacy-preserving Telemetry with Prio

Steven Englehardt | June 6, 2019

Private computation of functions is useful!

Apple's PSI

The Apple PSI System

Abhishek Bhowmick
Apple Inc.

Dan Boneh
Stanford University

Steve Myers
Apple Inc.

Kunal Talwar
Apple Inc.

Karl Tarbe
Apple Inc.

July 29, 2021

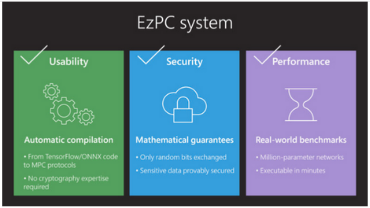
Mozilla's private aggregation

Mozilla Security Blog

Next steps in privacy-preserving Telemetry with Prio

Steven Englehardt | June 6, 2019

Microsoft's EzPZ



MP-SPDZ

data61 / **MP-SPDZ** Public

forked from bristolcrypto/SPDZ-2

Plan for Today's Lecture

1. Private Computation of Functions
2. New Tool: (Threshold) Secret Sharing
3. Computing Any Linear Function with Perfect Privacy

Plan for Today's Lecture

1 Private Computation of Functions

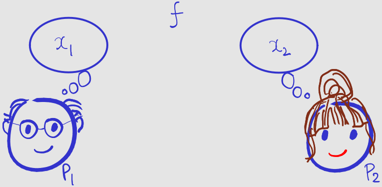
2 New Tool: (Threshold) Secret Sharing

3 Computing Any Linear Function with Perfect Privacy

Syntax of Two-Party Computation (2PC)

Defintion 1 (2PC protocol Π for $f : \mathcal{D}^2 \to \mathcal{R}$)

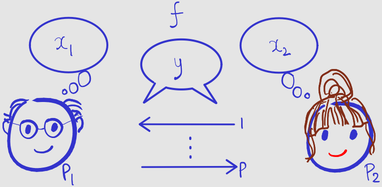
A ρ-round protocol $\Pi = (\mathsf{P}_1, \mathsf{P}_2)$ between two parties P_1 and P_2 with private input $x_1 \in \mathcal{D}$ and $x_2 \in \mathcal{D}$ and common output $y \in \mathcal{R}$



Syntax of Two-Party Computation (2PC)

Defintion 1 (2PC protocol Π for $f : \mathcal{D}^2 \to \mathcal{R}$)

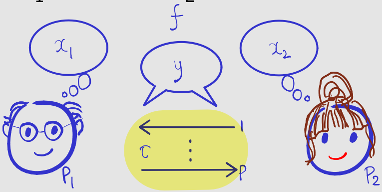
A ρ-round protocol $\Pi = (\mathsf{P}_1, \mathsf{P}_2)$ between two parties P_1 and P_2 with private input $x_1 \in \mathcal{D}$ and $x_2 \in \mathcal{D}$ and common output $y \in \mathcal{R}$



Syntax of Two-Party Computation (2PC)

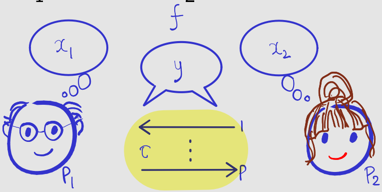
Defintion 1 (2PC protocol Π for $f : \mathcal{D}^2 \to \mathcal{R}$)

A ρ-round protocol $\Pi = (\mathsf{P}_1, \mathsf{P}_2)$ between two parties P_1 and P_2 with private input $x_1 \in \mathcal{D}$ and $x_2 \in \mathcal{D}$ and common output $y \in \mathcal{R}$



Defintion 1 (2PC protocol Π for f : $\mathcal{D}^2 \rightarrow \mathcal{R}$)

A ρ-round protocol $\Pi = (\mathsf{P}_1, \mathsf{P}_2)$ between two parties P_1 and P_2 with private input $x_1 \in \mathcal{D}$ and $x_2 \in \mathcal{D}$ and common output $y \in \mathcal{R}$

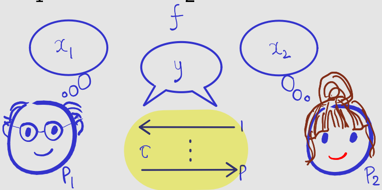


- *Correctness:* $\forall x_1, x_2 \in \mathcal{D}$, $\Pr[\langle P_1(x_1), P_2(x_2)\rangle \rightarrow f(x_1, x_2)] = 1$

Syntax of Two-Party Computation (2PC)

Defintion 1 (2PC protocol Π for $f : \mathcal{D}^2 \to \mathcal{R}$)

A ρ-round protocol $\Pi = (\mathsf{P}_1, \mathsf{P}_2)$ between two parties P_1 and P_2 with private input $x_1 \in \mathcal{D}$ and $x_2 \in \mathcal{D}$ and common output $y \in \mathcal{R}$



- *Correctness:* $\forall x_1, x_2 \in \mathcal{D},\ \Pr[\langle P_1(x_1), P_2(x_2)\rangle \to f(x_1, x_2)] = 1$

- Can be more general:
  - Parties can have common input
  - Each party's output can be different
  - Two parties $\to$ n parties (MPC)

$P_2(x_2)$

$P_i(x_i)$

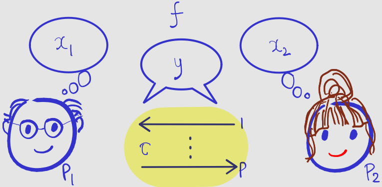
$(x_1)\ P_1$ (x)

$P_n(x_n)$

Syntax of Two-Party Computation (2PC)

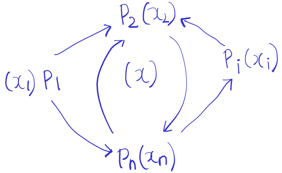
Defintion 1 (2PC protocol Π for f : $\mathcal{D}^2 \to \mathcal{R}$)

A ρ-round protocol $\Pi = (\mathsf{P}_1, \mathsf{P}_2)$ between two parties P_1 and P_2 with private input $x_1 \in \mathcal{D}$ and $x_2 \in \mathcal{D}$ and common output $y \in \mathcal{R}$



- *Correctness:* $\forall x_1, x_2 \in \mathcal{D},\ \Pr[\langle P_1(x_1), P_2(x_2)\rangle \to f(x_1, x_2)] = 1$

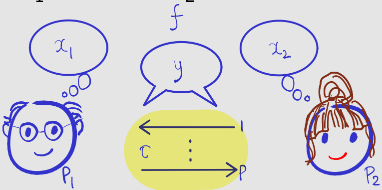
- Can be more general:
  - Parties can have common input
  - Each party's output can be different
  - Two parties $\to$ $\boldsymbol{n}$ parties (MPC)



Syntax of Two-Party Computation (2PC)

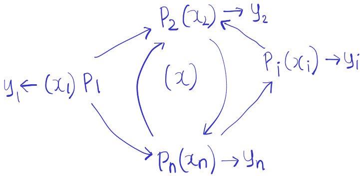
Defintion 1 (2PC protocol Π for $f : \mathcal{D}^2 \to \mathcal{R}$)

A ρ-round protocol $\Pi = (\mathsf{P}_1, \mathsf{P}_2)$ between two parties P_1 and P_2 with private input $x_1 \in \mathcal{D}$ and $x_2 \in \mathcal{D}$ and common output $y \in \mathcal{R}$



- *Correctness:* $\forall x_1, x_2 \in \mathcal{D},\ \Pr[\langle P_1(x_1), P_2(x_2)\rangle \to f(x_1, x_2)] = 1$

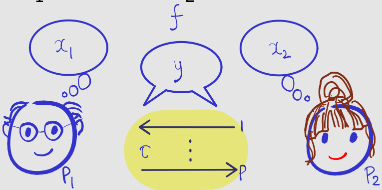
- Can be more general:
  - Parties can have common input
  - Each party's output can be different
  - Two parties $\to$ $\boldsymbol{n}$ parties (MPC)



Syntax of Two-Party Computation (2PC)

Defintion 1 (2PC protocol Π for $f : \mathcal{D}^2 \to \mathcal{R}$)

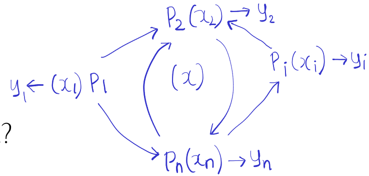
A ρ-round protocol $\Pi = (\mathsf{P}_1, \mathsf{P}_2)$ between two parties P_1 and P_2 with private input $x_1 \in \mathcal{D}$ and $x_2 \in \mathcal{D}$ and common output $y \in \mathcal{R}$



- *Correctness:* $\forall x_1, x_2 \in \mathcal{D},\ \Pr[\langle P_1(x_1), P_2(x_2)\rangle \to f(x_1, x_2)] = 1$

- Can be more general:
  - Parties can have common input
  - Each party's output can be different
  - Two parties $\to$ n parties (MPC)

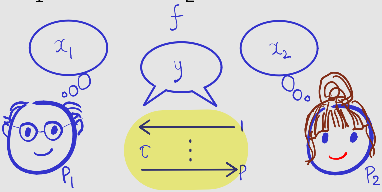
? How to frame ZKP as a 2PC protocol?



Syntax of Two-Party Computation (2PC)

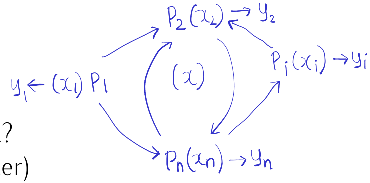
Defintion 1 (2PC protocol Π for f : $\mathcal{D}^2 \to \mathcal{R}$)

A ρ-round protocol $\Pi = (\mathsf{P}_1, \mathsf{P}_2)$ between two parties P_1 and P_2 with private input $x_1 \in \mathcal{D}$ and $x_2 \in \mathcal{D}$ and common output $y \in \mathcal{R}$



- *Correctness:* $\forall x_1, x_2 \in \mathcal{D}$, $\Pr[\langle P_1(x_1), P_2(x_2)\rangle \to f(x_1, x_2)] = 1$

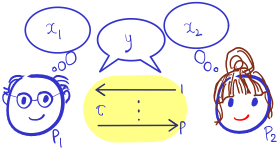
- Can be more general:
  - Parties can have common input
  - Each party's output can be different
  - Two parties $\to$ n parties (MPC)



? How to frame ZKP as a 2PC protocol?
  - $P_1 :=$ P (prover) and $P_2 :=$ V (verifier)
  - $f := \mathcal{R}$, the NP relation $\Rightarrow$ common input$=x$, $x_1 = w$, $x_2 = \perp$

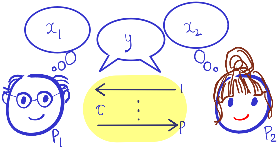
Let's Define Security...

? What are the requirements intuitively from P_1's perspective?



What are the requirements intuitively from P_1's perspective?

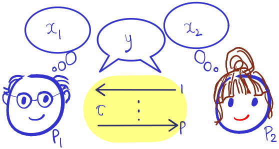
- P_2 should not learn anything about P_1's input x_1...



Let's Define Security...

What are the requirements intuitively from P_1's perspective?

- P_2 should not learn anything about P_1's input x_1...
- ... other than what she learns from output y



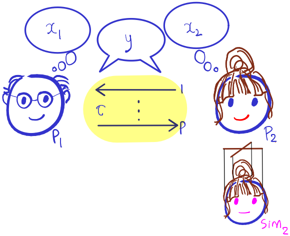
- How to formalise this?

Let's Define Security...

❓ What are the requirements intuitively from P_1's perspective?

- P_2 should not learn anything about P_1's input x_1...
- ... other than what she learns from output y

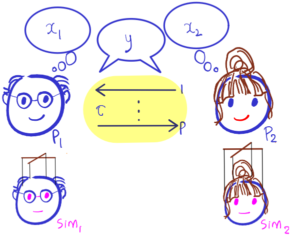
- How to formalise this? As in ZK
  - There exists a *simulator* for Sim_2's view



Let's Define Security...

? What are the requirements intuitively from P_1's perspective?

- P_2 should not learn anything about P_1's input x_1...
- ... other than what she learns from output y

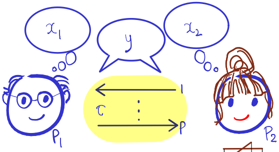


- How to formalise this? As in ZK
  - There exists a *simulator* for Sim_2's view
- Same from P_2 perspective

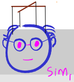
Let's Define Security...

What are the requirements intuitively from P_1's perspective?

- P_2 should not learn anything about P_1's input x_1...
- ... other than what she learns from output y



- How to formalise this? As in ZK
  - There exists a *simulator* for Sim_2's view
- Same from P_2 perspective



Defintion 2 (Semi-honest perfect security for 2PC)

Π *computes* f *with perfect privacy if there exists 1) a PPT simulator* Sim_2 *such that for all distinguishers* D *and for all* $x_1, x_2 \in \mathcal{D}$*, the following is zero*

$$\Pr[\mathsf{D}(\mathsf{View}_{\mathsf{P}_2}(\langle \mathsf{P}_1(x_1), \mathsf{P}_2(x_2)\rangle = 1] - \Pr[\mathsf{D}(\mathsf{Sim}_2)(x_2, y)) = 1]$$

and 2) a PPT simulator Sim_1 *such that....*

Let's Define Security...

Defintion 2 (Semi-honest perfect security for 2PC)

Π *computes* f *with perfect privacy if there exists 1) a PPT simulator* Sim_2 *such that for all distinguishers* D *and for all* $x_1, x_2 \in \mathcal{D}$*, the following is zero*

$$\Pr[\mathsf{D}(\mathsf{View}_{\mathsf{P}_2}(\langle \mathsf{P}_1(x_1), \mathsf{P}_2(x_2)\rangle = 1] - \Pr[\mathsf{D}(\mathsf{Sim}_2)(x_2, y)) = 1]$$

and 2) a PPT simulator Sim_1 *such that....*

- Extending the definition:
  - Semi-honest $\rightarrow$ malicious
    - Honest P_2/P_1 $\rightarrow$ any malicious P_2^*/P_1^* (just as in ZK)

Let's Define Security...

Defintion 2 (Semi-honest perfect security for 2PC)

Π *computes* f *with perfect privacy if there exists 1) a PPT simulator* Sim_2 *such that for all distinguishers* D *and for all* $x_1, x_2 \in \mathcal{D}$*, the following is zero*

$$\Pr[\mathsf{D}(\mathsf{View}_{\mathsf{P}_2}(\langle \mathsf{P}_1(x_1), \mathsf{P}_2(x_2)\rangle = 1] - \Pr[\mathsf{D}(\mathsf{Sim}_2)(x_2, y)) = 1]$$

and 2) a PPT simulator Sim_1 *such that....*

- Extending the definition:
  - Semi-honest $\rightarrow$ malicious
    - Honest $P_2/P_1 \rightarrow$ any malicious P_2^*/P_1^* (just as in ZK)
  - Two parties $\rightarrow$ n parties (MPC)

Let's Define Security...

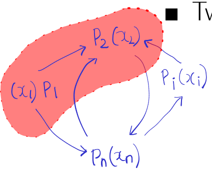
Defintion 2 (Semi-honest perfect security for 2PC)

Π computes f with perfect privacy if there exists 1) a PPT simulator Sim_2 such that for all distinguishers D and for all $x_1, x_2 \in \mathcal{D}$, the following is zero

$$\Pr[\mathsf{D}(\mathsf{View}_{\mathsf{P}_2}(\langle \mathsf{P}_1(x_1), \mathsf{P}_2(x_2)\rangle = 1] - \Pr[\mathsf{D}(\mathsf{Sim}_2)(x_2, y)) = 1]$$

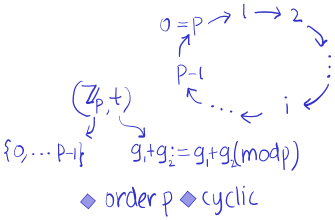
and 2) a PPT simulator Sim_1 such that....

- Extending the definition:
  - Semi-honest $\rightarrow$ malicious
    - Honest P_2/P_1 $\rightarrow$ any malicious P_2^*/P_1^* (just as in ZK)
  - Two parties $\rightarrow$ n parties (MPC): t-privacy (for $t \leq n$)
    - Any fixed $(t-1)$-sized subset of parties $\mathcal{P}^* \subset [n]$ can be corrupt
    - There exists $\mathsf{Sim}_{\mathcal{P}^*}$ that simulates views of all parties in $\mathcal{P}^*$ given their inputs $\{x_i\}_{i \in \mathcal{P}^*}$ and the output y

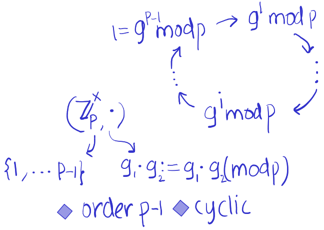


Plan for Today's Lecture

Addition modulo prime p

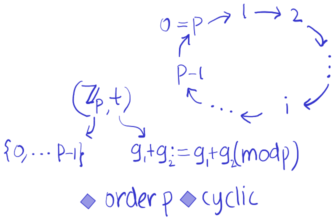


Multiplication modulo prime p

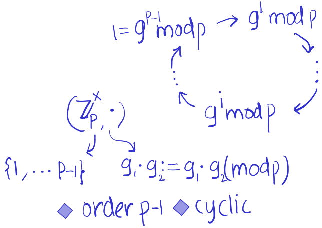


- Recall groups $(\mathbb{Z}_p, +)$ and $(\mathbb{Z}_p^\times, \cdot)$ from Lecture 08

Addition modulo prime p



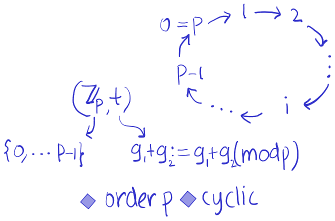
Multiplication modulo prime p



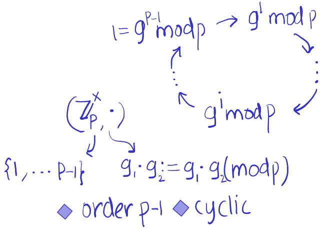
- Recall groups $(\mathbb{Z}_p, +)$ and $(\mathbb{Z}_p^\times, \cdot)$ from Lecture 08
  - $+$ and $\cdot$ are "compatible" with each other:
    - For any $a, b, c \in \mathbb{Z}_p$, we have $a \cdot (b + c) = a \cdot b + a \cdot c$

Finite Fields

Addition modulo prime p



Multiplication modulo prime p



- Recall groups $(\mathbb{Z}_p, +)$ and $(\mathbb{Z}_p^\times, \cdot)$ from Lecture 08
  - $+$ and $\cdot$ are "compatible" with each other:
    - For any $a, b, c \in \mathbb{Z}_p$, we have $a \cdot (b + c) = a \cdot b + a \cdot c$
- Can be combined to get a "field" $\mathbb{F}_p = (\mathbb{Z}_p, +, \cdot)$

Finite Fields

Addition modulo prime p

$0 = p \to 1 \to 2 \to \cdots \to i \to \cdots \to p-1 \to 0 = p$

$(\mathbb{Z}_p, +)$

$\{0, \dots p-1\}$ $\quad$ $g_1 + g_2 := g_1 + g_2 \pmod p$

◆ order p ◆ cyclic

Multiplication modulo prime p

$1 = g^{p-1} \bmod p \to g^1 \bmod p \to \cdots \to g^i \bmod p \to \cdots \to 1 = g^{p-1} \bmod p$

$(\mathbb{Z}_p^\times, \cdot)$

$\{1, \dots p-1\}$ $\quad$ $g_1 \cdot g_2 := g_1 \cdot g_2 \pmod p$

◆ order $p-1$ ◆ cyclic

- Recall groups $(\mathbb{Z}_p, +)$ and $(\mathbb{Z}_p^\times, \cdot)$ from Lecture 08
  - $+$ and $\cdot$ are "compatible" with each other:
    - For any $a, b, c \in \mathbb{Z}_p$, we have $a \cdot (b + c) = a \cdot b + a \cdot c$
- Can be combined to get a "field" $\mathbb{F}_p = (\mathbb{Z}_p, +, \cdot)$
- Finite counterpart of real numbers $\mathbb{R}$
  - $(\mathbb{R}, +)$ and $(\mathbb{R} \setminus \{0\}, \cdot)$ are groups
  - $+$ and $\cdot$ are distributive

Addition modulo prime p

$0 = p \to 1 \to 2 \to \cdots \to i \to \cdots \to p-1 \to 0$

$(\mathbb{Z}_p, +)$: $\{0, \dots, p-1\}$, $g_1 + g_2 := g_1 + g_2 \pmod p$

◆ order p ◆ cyclic

Multiplication modulo prime p

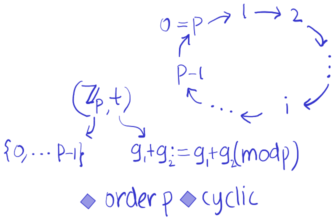
$1 = g^{p-1} \bmod p \to g^1 \bmod p \to \cdots \to g^i \bmod p \to \cdots \to 1$

$(\mathbb{Z}_p^\times, \cdot)$: $\{1, \dots, p-1\}$, $g_1 \cdot g_2 := g_1 \cdot g_2 \pmod p$

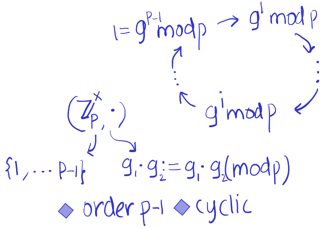
◆ order $p-1$ ◆ cyclic

- Recall groups $(\mathbb{Z}_p, +)$ and $(\mathbb{Z}_p^\times, \cdot)$ from Lecture 08
  - $+$ and $\cdot$ are "compatible" with each other:
    - For any $a, b, c \in \mathbb{Z}_p$, we have $a \cdot (b + c) = a \cdot b + a \cdot c$
- Can be combined to get a "field" $\mathbb{F}_p = (\mathbb{Z}_p, +, \cdot)$
- Finite counterpart of real numbers $\mathbb{R}$
  - $(\mathbb{R}, +)$ and $(\mathbb{R} \setminus \{0\}, \cdot)$ are groups
  - $+$ and $\cdot$ are distributive
- $\mathbb{F}_2 = (\mathbb{Z}_2, +, \cdot)$ corresponds to ?

Addition modulo prime p

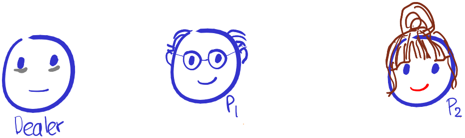


Multiplication modulo prime p



- Recall groups $(\mathbb{Z}_p, +)$ and $(\mathbb{Z}_p^\times, \cdot)$ from Lecture 08
  - $+$ and $\cdot$ are "compatible" with each other:
    - For any $a, b, c \in \mathbb{Z}_p$, we have $a \cdot (b + c) = a \cdot b + a \cdot c$
- Can be combined to get a "field" $\mathbb{F}_p = (\mathbb{Z}_p, +, \cdot)$
- Finite counterpart of real numbers $\mathbb{R}$
  - $(\mathbb{R}, +)$ and $(\mathbb{R} \setminus \{0\}, \cdot)$ are groups
  - $+$ and $\cdot$ are distributive
- $\mathbb{F}_2 = (\mathbb{Z}_2, +, \cdot)$ corresponds to Boolean algebra $(\{F, T\}, \oplus, \wedge)$

(t, n)-Threshold Secret Sharing (TSS)

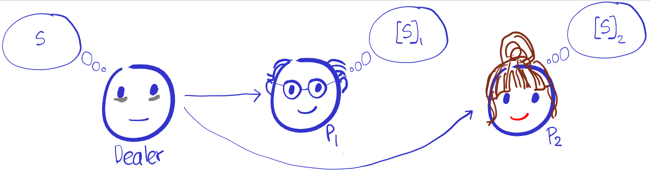


- Goal: share $s \in \mathbb{F}_p$ among n shares $[s]_1, \ldots, [s]_n$ such that:

(t, n)-Threshold Secret Sharing (TSS)

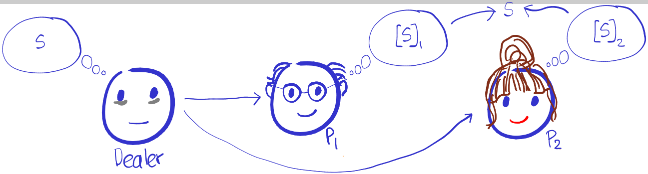
- Goal: share $s \in \mathbb{F}_p$ among n shares $[s]_1, \ldots, [s]_n$ such that:

(t, n)-Threshold Secret Sharing (TSS)



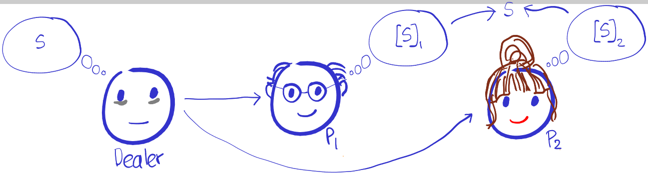
- Goal: share $s \in \mathbb{F}_p$ among n shares $[s]_1, \ldots, [s]_n$ such that:

(t, n)-Threshold Secret Sharing (TSS)



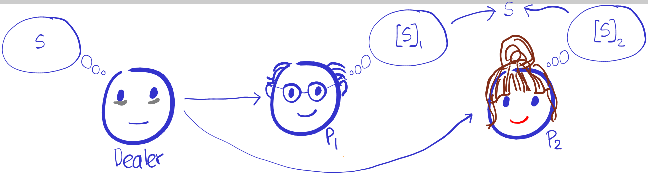
- Goal: share $s \in \mathbb{F}_p$ among n shares $[s]_1, \ldots, [s]_n$ such that:
  1. s can efficiently "reconstructed" given any subset of $\geq t$ shares
  2. s is perfectly hidden given any subset of $< t$ shares

(t, n)-Threshold Secret Sharing (TSS)



- Goal: share $s \in \mathbb{F}_p$ among n shares $[s]_1, \ldots, [s]_n$ such that:
  1. s can efficiently "reconstructed" given any subset of $\geq t$ shares
  2. s is perfectly hidden given any subset of $< t$ shares
- Let's construct TSS for $\mathbb{F}_2$. Can you come up with
  1. a $(1, 2)$-TSS?

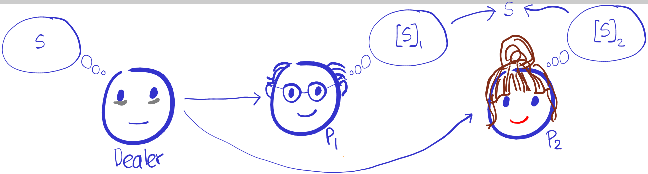
(t, n)-Threshold Secret Sharing (TSS)



- Goal: share $s \in \mathbb{F}_p$ among n shares $[s]_1, \ldots, [s]_n$ such that:
  1. s can efficiently "reconstructed" given any subset of $\geq t$ shares
  2. s is perfectly hidden given any subset of $< t$ shares

? Let's construct TSS for $\mathbb{F}_2$. Can you come up with
  1. a $(1, 2)$-TSS? Trivial: set $[s]_1 = [s]_2 := s$
  2. a $(2, 2)$-TSS using $\oplus$?

(t, n)-Threshold Secret Sharing (TSS)

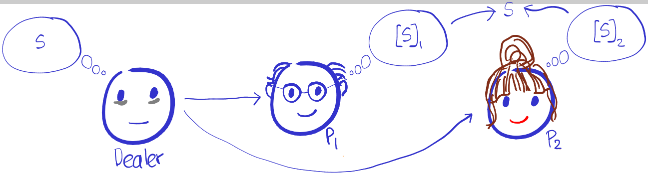


- Goal: share $s \in \mathbb{F}_p$ among n shares $[s]_1, \ldots, [s]_n$ such that:
  1. s can efficiently "reconstructed" given any subset of $\geq t$ shares
  2. s is perfectly hidden given any subset of $< t$ shares

? Let's construct TSS for $\mathbb{F}_2$. Can you come up with

1. a $(1, 2)$-TSS? Trivial: set $[s]_1 = [s]_2 := s$
2. a $(2, 2)$-TSS using $\oplus$?
   - Set $[s]_1 := k$ and $[s]_2 := k \oplus s$ for $k \leftarrow \{0, 1\}$
3. a $(2, n)$-TSS using $(2, 2)$-TSS?

(t, n)-Threshold Secret Sharing (TSS)



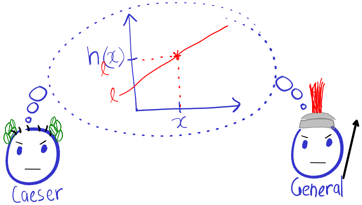
- Goal: share $s \in \mathbb{F}_p$ among n shares $[s]_1, \ldots, [s]_n$ such that:
  1. s can efficiently "reconstructed" given any subset of $\geq t$ shares
  2. s is perfectly hidden given any subset of $< t$ shares

? Let's construct TSS for $\mathbb{F}_2$. Can you come up with

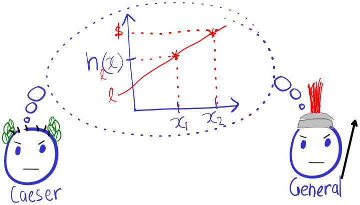
1. a $(1, 2)$-TSS? Trivial: set $[s]_1 = [s]_2 := s$
2. a $(2, 2)$-TSS using $\oplus$?
   - Set $[s]_1 := k$ and $[s]_2 := k \oplus s$ for $k \leftarrow \{0, 1\}$
3. a $(2, n)$-TSS using $(2, 2)$-TSS?

Exercise 1

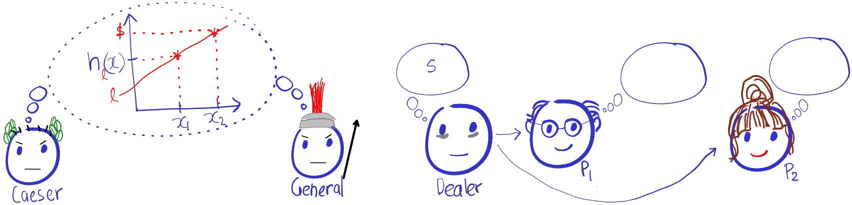
What happens when you extend 3 to construct (t, n)-TSS for arbitrary n and $t \leq n$?



- Recall construction of pairwise-independent hash function $h_\ell(x) := \ell(x)$
  - Key is a "random line" ℓ over $\mathbb{F}_p$ and hash is evaluation on ℓ



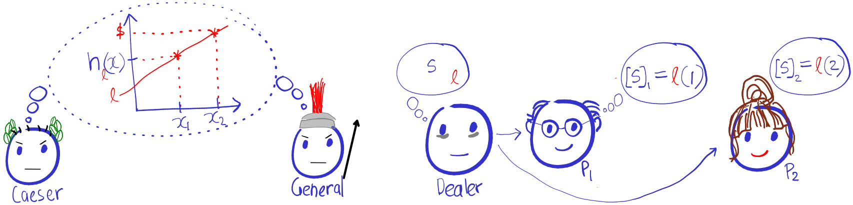
- Recall construction of pairwise-independent hash function $h_\ell(x) := \ell(x)$
  - Key is a "random line" ℓ over $\mathbb{F}_p$ and hash is evaluation on ℓ
  - Output is uniformly random over $\mathbb{F}_p$ as long as ℓ is evaluated at ≤ 2 points



- Recall construction of pairwise-independent hash function $h_\ell(x) := \ell(x)$
  - Key is a "random line" ℓ over $\mathbb{F}_p$ and hash is evaluation on ℓ
  - Output is uniformly random over $\mathbb{F}_p$ as long as ℓ is evaluated at ≤ 2 points

? Can you construct a $(2, n)$-TSS using ideas above?
  - Sharing $s \in \mathbb{F}_p$:

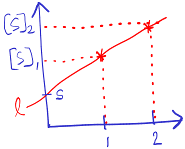
Shamir's (t, n)-TSS...



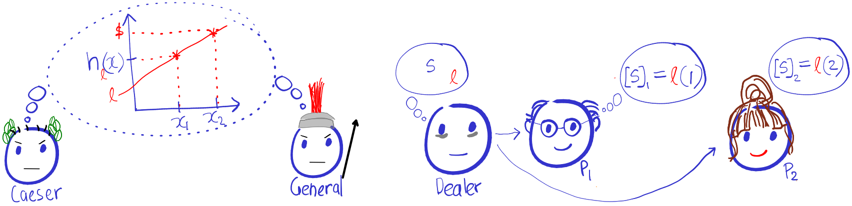
- Recall construction of pairwise-independent hash function $h_\ell(x) := \ell(x)$
  - Key is a "random line" ℓ over $\mathbb{F}_p$ and hash is evaluation on ℓ
  - Output is uniformly random over $\mathbb{F}_p$ as long as ℓ is evaluated at ≤ 2 points

? Can you construct a $(2, n)$-TSS using ideas above?

- Sharing $s \in \mathbb{F}_p$:
  1. Sample a random line ℓ over $\mathbb{F}_p$ with $\ell(0) := s$
  2. Share $[s]_i$ of party P_i is $[s]_i := \ell(i) \in \mathbb{F}_p$
- Reconstruction from $[s]_i$ and $[s]_j$ $(i \neq j)$



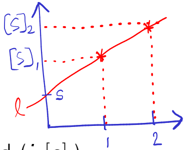
Shamir's (t, n)-TSS...



- Recall construction of pairwise-independent hash function $h_\ell(x) := \ell(x)$
  - Key is a "random line" ℓ over $\mathbb{F}_p$ and hash is evaluation on ℓ
  - Output is uniformly random over $\mathbb{F}_p$ as long as ℓ is evaluated at ≤ 2 points

Can you construct a $(2, n)$-TSS using ideas above?

- Sharing $s \in \mathbb{F}_p$:
  1. Sample a random line ℓ over $\mathbb{F}_p$ with $\ell(0) := s$
  2. Share $[s]_i$ of party P_i is $[s]_i := \ell(i) \in \mathbb{F}_p$
- Reconstruction from $[s]_i$ and $[s]_j$ $(i \neq j)$
  1. Reconstruct ℓ by drawing line through $(i, [s]_i)$ and $(j, [s]_j)$
  2. Output $\ell(0)$



Shamir's (t, n)-TSS...

Why does reconstruction work?

Shamir's (t, n)-TSS...

- Why does reconstruction work? Two points uniquely determine a line (even in $\mathbb{F}_p$)
- Why is it perfectly hiding given only one share?

Shamir's (t, n)-TSS...

- Why does reconstruction work? Two points uniquely determine a line (even in $\mathbb{F}_p$)
- Why is it perfectly hiding given only one share? One point doesn't determine a line (even in $\mathbb{F}_p$) $\Rightarrow$ Evaluation at 0 random

Shamir's (t, n)-TSS...

- Why does reconstruction work? Two points uniquely determine a line (even in $\mathbb{F}_p$)
- Why is it perfectly hiding given only one share? One point doesn't determine a line (even in $\mathbb{F}_p$) $\Rightarrow$ Evaluation at 0 random

+ The construction is "linear":
  - $([s_1]_1, [s_1]_2)$ shares of s_1 and $([s_2]_1, [s_2]_2)$ shares of $s_2 \Rightarrow$ $([s_1]_1 + [s_2]_1 \bmod p, [s_1]_2 + [s_2]_2 \bmod p)$ shares of $s_1 + s_2 \bmod p$
  - $([s]_1, [s]_2)$ shares of $s \Rightarrow$ for $c \in \mathbb{F}_p$, $(c \cdot [s]_1 \bmod p, c \cdot [s]_2 \bmod p)$ shares of $s \cdot c \bmod p$

Shamir's (t, n)-TSS...

- Why does reconstruction work? Two points uniquely determine a line (even in $\mathbb{F}_p$)
- Why is it perfectly hiding given only one share? One point doesn't determine a line (even in $\mathbb{F}_p$) $\Rightarrow$ Evaluation at 0 random

+ The construction is "linear":
  - $([s_1]_1, [s_1]_2)$ shares of s_1 and $([s_2]_1, [s_2]_2)$ shares of s_2 $\Rightarrow$ $([s_1]_1 + [s_2]_1 \bmod p, [s_1]_2 + [s_2]_2 \bmod p)$ shares of $s_1 + s_2 \bmod p$
  - $([s]_1, [s]_2)$ shares of s $\Rightarrow$ for $c \in \mathbb{F}_p$, $(c \cdot [s]_1 \bmod p, c \cdot [s]_2 \bmod p)$ shares of $s \cdot c \bmod p$

- All of the above ideas extend to arbitrary n and $t \leq n$

Exercise 2 (Hint: use a degree-t polynomial instead of line)

Formally describe and prove Shamir's (t, n)-TSS

Plan for Today's Lecture

How to Privately Compute Linear Functions?...

Defintion 3

A function $\mathsf{f} : \mathbb{F}_p^n \to \mathbb{F}_p$ *is linear if* $\forall \bar{a}, \bar{b} \in \mathbb{F}_p^n : \mathsf{f}(\bar{a} + \bar{b}) = \mathsf{f}(\bar{a}) + \mathsf{f}(\bar{b})$

$\Rightarrow \forall c \in \mathbb{F}_p, \bar{a} \in \mathbb{F}_p^n : \mathsf{f}(c \cdot \bar{a}) = c \cdot \mathsf{f}(\bar{a})$

How to Privately Compute Linear Functions?...

Defintion 3

A function $\mathsf{f} : \mathbb{F}_p^n \to \mathbb{F}_p$ *is linear if* $\forall \bar{a}, \bar{b} \in \mathbb{F}_p^n : \mathsf{f}(\bar{a} + \bar{b}) = \mathsf{f}(\bar{a}) + \mathsf{f}(\bar{b})$

$$\Rightarrow \forall c \in \mathbb{F}_p, \bar{a} \in \mathbb{F}_p^n : \mathsf{f}(c \cdot \bar{a}) = c \cdot \mathsf{f}(\bar{a})$$

Exercise 3

Show that any linear function f *can be computed using a circuit* C *with addition gates* $\oplus$ *and multiply-by-constant gates* $\odot_c$

over $\mathbb{F}_p$

How to Privately Compute Linear Functions?

Defintion 3

A function $\mathsf{f} : \mathbb{F}_p^n \to \mathbb{F}_p$ *is linear if* $\forall \bar{a}, \bar{b} \in \mathbb{F}_p^n : \mathsf{f}(\bar{a} + \bar{b}) = \mathsf{f}(\bar{a}) + \mathsf{f}(\bar{b})$

$$\Rightarrow \forall c \in \mathbb{F}_p, \bar{a} \in \mathbb{F}_p^n : \mathsf{f}(c \cdot \bar{a}) = c \cdot \mathsf{f}(\bar{a})$$

Exercise 3

Show that any linear function f *can be computed using a circuit* C *with addition gates* $\oplus$ *and multiply-by-constant gates* $\odot_c$

over $\mathbb{F}_p$

Idea: each party P_i secret-shares its input x_i with all other parties and everyone computes locally “over shares”

How to Privately Compute Linear Functions?

Defintion 3

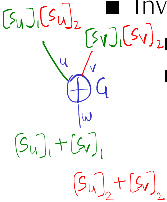
A function $\mathsf{f} : \mathbb{F}_p^n \to \mathbb{F}_p$ *is linear if* $\forall \bar{a}, \bar{b} \in \mathbb{F}_p^n : \mathsf{f}(\bar{a} + \bar{b}) = \mathsf{f}(\bar{a}) + \mathsf{f}(\bar{b})$

$$\Rightarrow \forall c \in \mathbb{F}_p, \bar{a} \in \mathbb{F}_p^n : \mathsf{f}(c \cdot \bar{a}) = c \cdot \mathsf{f}(\bar{a})$$

Exercise 3

Show that any linear function f *can be computed using a circuit* C *with addition gates* $\oplus$ *and multiply-by-constant gates* $\odot_c$

- Idea: each party P_i secret-shares its input x_i with all other parties and everyone computes locally "over shares"
- Invariant: every party P_i will have secret share $[s_w]_i$ of wire w
  - To generate shares of output of $\oplus$, *add* shares of input wires
  - To generate shares of output of $\odot_c$, *multiply* share with c



How to Privately Compute Linear Functions?

Defintion 3

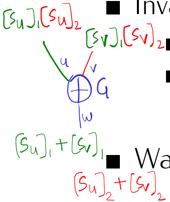
A function $\mathsf{f} : \mathbb{F}_p^n \to \mathbb{F}_p$ *is linear if* $\forall \bar{a}, \bar{b} \in \mathbb{F}_p^n : \mathsf{f}(\bar{a} + \bar{b}) = \mathsf{f}(\bar{a}) + \mathsf{f}(\bar{b})$

$$\Rightarrow \forall c \in \mathbb{F}_p, \bar{a} \in \mathbb{F}_p^n : \mathsf{f}(c \cdot \bar{a}) = c \cdot \mathsf{f}(\bar{a})$$

Exercise 3

Show that any linear function f *can be computed using a circuit* C *with addition gates* $\oplus$ *and multiply-by-constant gates* $\odot_c$

- Idea: each party P_i secret-shares its input x_i with all other parties and everyone computes locally "over shares"
- Invariant: every party P_i will have secret share $[s_w]_i$ of wire w
  - To generate shares of output of $\oplus$, *add* shares of input wires
    - To generate shares of output of $\odot_c$, *multiply* share with c



- Warm-up: let's privately compute $\oplus$ over $\mathbb{F}_2$

How to Privately Compute Linear Functions?...

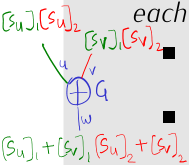
Protocol 1 (Protocol Π for linear f : $\mathbb{F}_p^n \to \mathbb{F}_p$)

1. *Secret-share input:*
   1. *Each P_i chooses random degree-t polynomial q_i with $q_i(0) = x_i$*
   2. *Each P_i sends share $[x_i]_j := q_i(j)$ to P_j (for all $j \neq i$)*

How to Privately Compute Linear Functions?...

Protocol 1 (Protocol Π for linear f : $\mathbb{F}_p^n \to \mathbb{F}_p$)

1. *Secret-share input:*
   1. *Each P_i chooses random degree-t polynomial q_i with $q_i(0) = x_i$*
   2. *Each P_i sends share $[x_i]_j := q_i(j)$ to P_j (for all $j \neq i$)*
2. *Emulate circuit: for each gate G_k with in topological order, each P_i does the following*



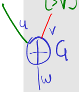
   - *If $G_k = \oplus$: define share of G_k's output wire to be sum of P_i's shares of G_k's input wires*
   - *If $G_k = \odot_c$: define share of G_k's output wire to be c times share of G_k's input wire*

Protocol 1 (Protocol Π for linear f : $\mathbb{F}_p^n \to \mathbb{F}_p$)

1. *Secret-share input:*
   1. *Each P_i chooses random degree-t polynomial q_i with $q_i(0) = x_i$*
   2. *Each P_i sends share $[x_i]_j := q_i(j)$ to P_j (for all $j \neq i$)*
2. *Emulate circuit: for each gate G_k with in topological order, each P_i does the following*

   $[s_u]_1, [s_u]_2$

   $[s_v]_1, [s_v]_2$

   - *If $G_k = \oplus$: define share of G_k's output wire to be sum of P_i's shares of G_k's input wires*
   - *If $G_k = \odot_c$: define share of G_k's output wire to be c times share of G_k's input wire*

   $[s_u]_1 + [s_v]_1, [s_u]_2 + [s_v]_2$
3. *Reconstruct output: each party P_i*
   1. *Broadcasts its share of output wire to parties P_j, $j \neq i$*
   2. *Reconstructs q_i from all shares of output wire*
   3. *Outputs $q_i(0)$*

Π Computes f with Perfect Secrecy

Theorem 1

Assuming $(\boldsymbol{t}, \boldsymbol{n})$-linear TSS, Π computes f *with $\boldsymbol{t}$-privacy*

Proof (Sketch).

- Idea: $< \boldsymbol{t}$ parties $\mathcal{P}^*$ corrupt $\Rightarrow$ inputs of $[\boldsymbol{n}] \setminus \mathcal{P}^*$ perfectly hidden by security of TSS

Π Computes f with Perfect Secrecy

Theorem 1

Assuming (t, n)-linear TSS, Π computes f *with t-privacy*

Proof (Sketch).

- Idea: $< t$ parties $\mathcal{P}^*$ corrupt $\Rightarrow$ inputs of $[n] \setminus \mathcal{P}^*$ perfectly hidden by security of TSS
- Simulator $\mathsf{Sim}(\mathcal{P}^*, \{x_i\}_{i \in \mathcal{P}^*}, y)$
  - Share input:
    1. For every $i \in \mathcal{P}^*$, sample random q_i with $q_i(0) = x_i$ (as in Π)
    2. For every $i \notin \mathcal{P}^*$, sample random q_i with $q_i(0) = 0$

Π Computes f with Perfect Secrecy

Theorem 1

Assuming (t, n)-linear TSS, Π computes f with t-privacy

Proof (Sketch).

- Idea: $< t$ parties $\mathcal{P}^*$ corrupt $\Rightarrow$ inputs of $[n] \setminus \mathcal{P}^*$ perfectly hidden by security of TSS
- Simulator $\mathsf{Sim}(\mathcal{P}^*, \{x_i\}_{i \in \mathcal{P}^*}, y)$
  - Share input:
    1. For every $i \in \mathcal{P}^*$, sample random q_i with $q_i(0) = x_i$ (as in Π)
    2. For every $i \notin \mathcal{P}^*$, sample random q_i with $q_i(0) = 0$
  - Emulate circuit: for each non-output gate G_k in top. order
    - Generate shares of G_k's output wire as in Π

Π Computes f with Perfect Secrecy

Theorem 1

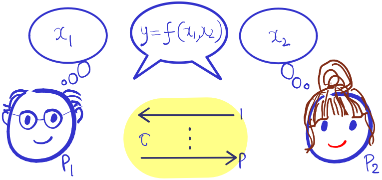
Assuming (t, n)-linear TSS, Π computes f *with t-privacy*

Proof (Sketch).

- Idea: $< t$ parties $\mathcal{P}^*$ corrupt $\Rightarrow$ inputs of $[n] \setminus \mathcal{P}^*$ perfectly hidden by security of TSS
- Simulator $\mathsf{Sim}(\mathcal{P}^*, \{x_i\}_{i \in \mathcal{P}^*}, y)$
  - Share input:
    1. For every $i \in \mathcal{P}^*$, sample random q_i with $q_i(0) = x_i$ (as in Π)
    2. For every $i \notin \mathcal{P}^*$, sample random q_i with $q_i(0) = 0$
  - Emulate circuit: for each non-output gate G_k in top. order
    - Generate shares of G_k's output wire as in Π
  - Program output:
    - Set polynomial q_o of output gate consistently with its input wires and with $q_o(0) = y$ □

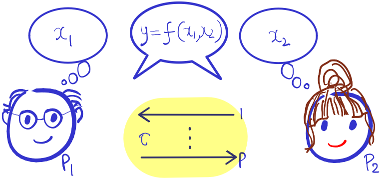
To Recap Today's Lecture

- Task 6: Private computation of functions



To Recap Today's Lecture

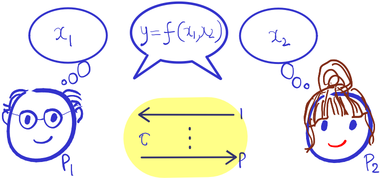
- Task 6: Private computation of functions



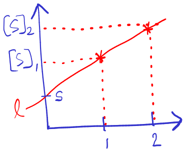
- Defined syntax and security for the *two-party* case (2PC)
  - Extends the simulation paradigm

To Recap Today's Lecture

- Task 6: Private computation of functions



- Defined syntax and security for the *two-party* case (2PC)
  - Extends the simulation paradigm
- Perfectly-private MPC for *linear* functions
  - Key tool: threshold secret sharing (TSS)
  - Shamir's TSS
  - Key idea: "compute over shares"



Next Lecture

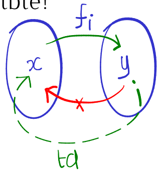
- Continue with Task 6

  ⚠ Perfectly-private 2PC for *general* functions is impossible!

  - Counter-example: $\wedge$

Next Lecture

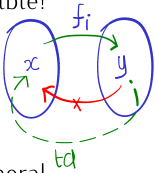
- Continue with Task 6
  - ⚠ Perfectly-private 2PC for *general* functions is impossible!
    - Counter-example: $\wedge$
  - What do we do? Relax to computational privacy
  - New tool: oblivious transfer
    - Oblivious transfer from trapdoor permutations



Next Lecture

- Continue with Task 6

  ⚠ Perfectly-private 2PC for *general* functions is impossible!
    - Counter-example: $\wedge$
  - What do we do? Relax to computational privacy
  - New tool: oblivious transfer
    - Oblivious transfer from trapdoor permutations
  - GMW protocol: computationally-private MPC for general functions



References

1. MPC was first studied in [GMW87], building on [GMR89]
2. Shamir's TSS is from [Sha79]
3. The perfectly-secure MPC protocol for linear functions described here is taken from [AL17, §4.2]

Gilad Asharov and Yehuda Lindell.
A full proof of the BGW protocol for perfectly secure multiparty computation.
Journal of Cryptology, 30(1):58–151, January 2017.

Shafi Goldwasser, Silvio Micali, and Charles Rackoff.
The knowledge complexity of interactive proof systems.
SIAM J. Comput., 18(1):186–208, 1989.

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.

Adi Shamir.
How to share a secret.
Commun. ACM, 22(11):612–613, 1979.