## Two-Party Computation / Secure Function Evaluation

Logistics: - Project milestones due now

- Final project due in last lecture (June 7th)

Poll: topics for final lectures: multiparty computation (information-theoretic)
classical cryptanalysis
lattice-based cryptography + quantum cryptanalysis
} vote on topics (suggest new topics)

(will vote during the break)

Recap of zero-knowledge:

- Zero-knowledge proofs: prove something without revealing anything more other than the fact that statement is true (i.e., contained in the language)
  - Notion formalized by introducing concept of a simulator (verifier's view in the interactive proof can be simulated)
  - Beautiful concept with a very elegant formalization $\Rightarrow$ has become one of the pillars of modern cryptography
- Proof of Knowledge: prove not just membership, but also that prover knows a witness
  - What does it mean to "know" something? An extractor can extract the witness from any successful prover.
  - Can be combined with zero-knowledge: zkPoK
- Schnorr's protocol for knowledge of discrete log:

prover $\quad h = g^x \quad$ verifier

$r \xleftarrow{R} \mathbb{Z}_q \quad \xrightarrow{u = g^r}$

$\xleftarrow{c} \quad c \xleftarrow{R} \mathbb{Z}_q$

$\xrightarrow{z = r + cx} \quad$ check that $g^z \stackrel{?}{=} u \cdot h^c$

- honest-verifier zero-knowledge: run protocol in reverse
- proof of knowledge: rewind prover to extract secrets
- directly gives identification protocol
- can be generalized for more general discrete log relations

- Fiat-Shamir heuristic: honest-verifier public-coin protocol $\Rightarrow$ NIZK in random oracle model (derive randomness from random oracle)
- Schnorr + Fiat-Shamir $\Rightarrow$ signatures from discrete log (basis of DSA / ECDSA)
  - Quite remarkable: zero-knowledge was developed for purely theoretical reasons (i.e. it wo[as] a cool idea) and has now become de facto standard on the web (digital signature schema)

Brief Segway : Zero-Knowledge Problem on HW2 (Reasoning about definitions)

- Protocol that is HVZK but not ZK : add a "bad" branch to zero-knowledge protocol

G is 3-colorable

prover $\qquad$ verifier

$\xleftarrow{\quad b = 0 \quad}$

if $b = 0$: usual ZK protocol

if $b = 1$: send 3-coloring $\longrightarrow$

$\longleftarrow$

HVZK: honest verifier always uses 0, so can simulate using standard ZK simulator

ZK : cannot simulate if malicious verifier sends $b = 1$

Note: - not known that Schnorr's 3-round protocol for discrete log is ZK or not.
- 3-coloring protocol where prover sends over all commitments cannot be simulated (need to simulate view of verifier)

- Protocol that is ZK if and only if factoring is in BPP

language: $\{ N \in \mathbb{Z} \mid N = pq \}$

prover $\quad N \quad$ verifier

$\xrightarrow{\quad p, q \quad}$

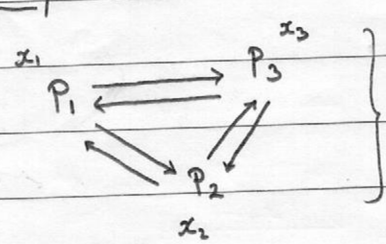
check $N \stackrel{?}{=} pq$

- ZK : simulator is BPP algorithm for factoring
- Factoring $\in$ BPP : factoring algorithm is simulator

Note : protocol where verifier chooses $p, q$, $N = pq$ and prover sends factors do not work (can simulate even if factoring is hard since simulator chooses an $N$ where it knows the factors)

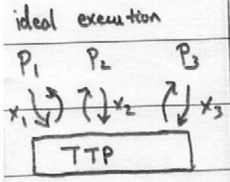
This lecture : look at multiparty computation (a cornerstone in modern cryptography and encompasses almost all of cryptography)

Abstractly : we have collection of parties that want to perform some task on secret inputs




at conclusion of protocol execution, each party should learn $f(x_1, x_2, x_3)$ but nothing more about other parties' inputs

ideal execution




- Yao's millionaire's problem :

$P_1 \; (x_1) \qquad P_2 \; (x_2)$

$\longrightarrow$

$\longleftarrow$

$\mathbb{1}\{x_1 > x_2\} \qquad \mathbb{1}\{x_1 > x_2\}$

parties learn who has the largest value but nothing more about other's value

- Zero-Knowledge

prover $(x, w) \qquad$ verifier $(x)$

$\longrightarrow$

$\longleftarrow$

$\perp \qquad R(x, w)$

verifier does not learn anything more about witness

## Multiparty computation

- Private set intersection (e.g. private contact discovery)

client (list of contacts) $\longrightarrow$ Signal server (list of users)

$\longleftarrow$

client: $\downarrow$ set of contacts that use Signal

server: $\searrow$ $\perp$

- Signal does not learn who is in the client's list of contacts
- client does not learn who is using Signal aside from contacts in its address book

- Danish sugar beet auction
  - Danish sugar beet farmers sell to Danish sugar producer
    ↳ need to negotiate a fair contract (standard mechanism is a sealed-bid auction)
  - Auction relies on secret information $\Rightarrow$ hence the use of multiparty computation

**Informal Theorem** [Yao82, GMW87]. Any functionality that can be computed with a trusted party can be computed <u>without</u> the trusted party.

↳ Remarkable theorem: cryptography removes the <u>need</u> for trust assumptions!

Oblivious Transfer: a primitive that is <u>complete</u> for multiparty computation
- Notably: under black-box separations, one-way functions (symmetric cryptography) and public-key encryption do <u>not</u> suffice for <u>general</u> multiparty computation.

OT: 2-party protocol between sender and receiver

ideal functionality:

TTP

sender $\xrightarrow{m_0, m_1}$ TTP; receiver $\xrightarrow{b \in \{0,1\}}$ TTP; TTP $\xrightarrow{m_b}$ receiver

- sender learns nothing about $b$
- receiver learns nothing about $m_{1-b}$
- remarkable that this is sufficient for <u>all</u> of MPC!

More formally, let $\text{View}_S((m_0, m_1), b)$ denote the view of the sender in the OT protocol on inputs $(m_0, m_1)$ and $b$. Define $\text{View}_R((m_0, m_1), b)$ accordingly.

<u>Receiver Privacy</u>: sender does not learn $b$:

for all pairs of messages $(m_0, m_1)$:

$$\text{View}_S((m_0, m_1), 0) \overset{c}{\approx} \text{View}_S((m_0, m_1), 1)$$

indistinguishability-based notion of security

<u>Sender Privacy</u>: receiver <u>learns nothing</u> about $m_{1-b}$ (other than what could be inferred from $m_b$)

for all pairs of messages $(m_0, m_1)$ and all efficient receivers $R^*$, there exists an efficient simulator Sim:

$$\text{View}_R((m_0, m_1), b) \overset{c}{\approx} \text{Sim}(b, m_b)$$

<u>OT from DDH (Naor-Pinkas protocol)</u>:

<u>sender $(m_0, m_1)$</u> | <u>receiver $(b \in \{0,1\})$</u>

receiver: $r, s, t \xleftarrow{R} \mathbb{Z}_q$

$x = g^s \qquad z_b = g^{st}$

$y = g^t \qquad z_{1-b} = g^r$

receiver → sender: $(x, y, z_0, z_1)$

sender:

1. check that $z_0 \neq z_1$
2. choose $u_0, u_1, v_0, v_1 \xleftarrow{R} \mathbb{Z}_q$ and compute

   $w_b \leftarrow y^{u_b} g^{v_b}$

   $c_b \leftarrow z_b^{u_b} x^{v_b} \cdot m_b$

sender → receiver: $(w_0, c_0), (w_1, c_1)$

receiver: computes $\dfrac{c_b}{w_b^s}$

- Recall ElGamal (encryption from DDH)

  pk: $g, g^s = h$ $\qquad$ Encrypt(pk, m): $g^t, h^t \cdot m$

  sk: $s$

  security follows from DDH: $(g, g^s, g^t, g^{st})$ — $g^{st}$ used to blind message

How to view Naor-Pinkas

- In Naor-Pinkas, receiver's requested message is encrypted using ElGamal:

  pk: $g, g^s = h$ $\qquad$ Encrypt$(pk, m_b)$: $g^{tu_b + v_b}, \quad h^{tu_b + v_b} \cdot m_b$

  sk: $s$

  - Receiver chooses ElGamal public key and randomness used for encryption
  - Sender re-randomizes the DDH tuple in order to hide the message (pairwise-independent hash in the exponent)
  - Common technique when working with DDH (random self-reduction)

    ↳ for instance: also used in HW2, extra credit

Naor-Pinkas Protocol

Receiver Privacy : Immediate from DDH : sender's view consists of random group elements

Sender Privacy : Information-theoretic :

$$(g, g^s, g^t, g^c) \quad \text{for } c \neq st$$

$$\Rightarrow \quad (g, g^s, \underbrace{g^{tu+v}, g^{cu+sv}}_{})$$

independent and uniformly random since $u, v$ are uniform and independent $\Rightarrow$ encryption is a one-time pad blinding of message

Simulator is trivial to construct: for $c_{1-b}$, output random group element, simulate other components as in the real protocol

OT requires algebraic assumptions to build (DDH, factoring, LWE, etc.)

↳ must perform public-key operations ⇝ can be expensive in practice if need to do many OTs

OT extensions [Ishai Kilian Nissim Petrank] : "Bootstrapping OT" : perform a large number of OTs at the cost of $\approx \lambda$ base OTs and extend using symmetric primitives

- However: communication still scales linearly
- Take-away : OTs are fairly cheap computationally (~ few hundred ECC operations + AES for the rest), but need to communicate proportionally to size of data / messages

  ↳ Can reduce communication via PIR, but costly in computation (or need multiple servers)