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 topic)

Recap of zero-knowledge:
- Zero-knowledge proof: 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 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:

$$\begin{align*}
\text{prover} & : h = g^x \\
\text{verifier} & : r \in \mathbb{Z}_p^* \quad u = g^r \\
\text{prover} & : c \in \mathbb{Z}_q \\
& : z = y + cx
\end{align*}$$

check that $g^z = u \cdot h^c$ - can be generalized for more general discrete log relations

Fiat-Shamir heuristic: honest-verifier public-coin protocol $\Rightarrow$ NIZK in random oracle model
(desire 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 was a cool idea) and has now become de facto standard on the web (digital signature scheme)
Brief Segway: Zero-Knowledge Prover on HW2. (Reasoning about definitions)

- Protocol that is HVZK but not ZK: add a "bad" branch to zero-knowledge protocol
  
  \[ G \text{ is 3-valued verifier} \]
  
  \[ b = 0 \]HVZK: honest verifier always uses 0, so can simulate using
  
  Standard ZK simulator
  
  \[ b = 0; \text{ usual ZK protocol} \]
  
  \[ b = 1; \text{ send breaking } \]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-round protocol where prover sends over all commitments cannot be simulated (used to simulate zero of Pedersen)

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

\[ \text{language: } \{ N \in \mathbb{Z} | N = p^2 \} \]

- ZK: simulator is BPP algorithm for factoring

\[ \text{prover } N \quad \text{ verifier } \]

\[ \rightarrow \text{ check } N = p^2 \]

Note: protocol where verifier chooses \( p, N = p^2 \) and prover sends factors do not work

(can simulate even if factoring is hard since simulator chooses on \( p \) 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

\[ x_1, x_2, x_3 \]

\[ \text{at conclusion of protocol execution, each party should learn } f(x_1, x_2, x_3) \]

but nothing more about other parties' inputs

\[ P_1 (x_1) \quad P_2 (x_2) \quad \text{parties learn who has the largest value} \]

\[ \text{but nothing more about other's value} \]

\[ 1 \{ x > x_1 \} \quad 1 \{ x > x_2 \} \]

- Yao's millionaire's problem:

- Zero-Knowledge
Multiparty computation

- Private set intersection (e.g., private contact discovery)
  
  \[ \text{client} \quad \text{Signal} \quad \text{server} \quad \text{list of contacts} \quad \text{list of users} \]

  \[ \text{client does not learn who is in the client's list of contacts} \]

  \[ \text{Signal does not learn who is in its address book} \]

  \[ \text{client does not learn who is using} \]

  \[ \text{Signal aside from contacts in its address book} \]

- Danish sugar beet auction

  - Danish sugar beet farmers sell to Danish sugar producer

  \[ \Rightarrow \text{need to negotiate a fair contract (standard mechanism is a sealed-bid auction)} \]

  - Auction relies on secret information \[ \Rightarrow \text{hence the use of multiparty computation} \]

Informal Theorem [Yao 82, GMW 87]. Any functionality that can be computed with a trusted party can be computed without the trusted party.

\[ \Rightarrow \text{Remarkable theorem: cryptography removes the need for trust assumptions!} \]

Obvious Transfer: a primitive that is complete for multiparty computation

- Notably: under black-box separations, one-way functions (symmetric cryptography) and public-key encryption do not suffice for general multiparty computation.

OT: 2-party protocol between sender and receiver

\[ \begin{array}{c}
\text{ideal functionality:} \\
\text{Sender learns nothing about b} \\
\text{Receiver learns nothing about m_1 \oplus b} \\
\text{Remarkable that this is sufficient for all of MPC!} \\
\end{array} \]

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.
Receiver Privacy: sender does not learn $b$:
for all pairs of messages $(m_0, m_1)$:

\[ \text{View}_S((m_0, m_1), b) \approx \text{View}_S((m_0, m_1), 1) \]

Sender Privacy: receiver learns nothing about $m_b$ (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 $\text{Sim}$:

\[ \text{View}_R((m_0, m_1), b) \approx \text{Sim}(b, m_b) \]

OT from DDH (Naor-Pinkas protocol):

\begin{align*}
\text{Sender (m_0, m_1)} & \quad \text{receiver (b \in \{0, 1\})} \\
& \quad \gamma, \tau, t, x \in \mathbb{Z}_p^* \\
& \quad z = g^\gamma, \quad z_b = g^{\gamma t} \\
& \quad y = g^\gamma, \quad x_{1b} = g^x \\
& \quad (x, y, z, z_b) \quad \xleftarrow{} \end{align*}

1. check that $z_b \neq 1$
2. choose $u_0, u_1, v_0, v_1 \in \mathbb{Z}_p^*$

and compute:

\[ w_b = u_0 - v_b \]
\[ c_b = z_b^{u_b} \cdot v_0 \cdot m_b \]

\[ (w_0, c_0), (w_1, c_1) \rightarrow \text{computes } \frac{c_b}{w_b} \]

- Recall ElGamal (encryption from DDH)
  \[ \text{pk}: g, g^s = h \quad \text{Encrypt}(\text{pk}, m): g^t, h^{tu_b + v_b} \cdot m_b \]  
  \[ \text{sk}: s \quad \text{Security follows from DDH}: (g, g^s, g^t, g^{tu_b + v_b}) \]

How to view Naor-Pinkas:
- In Naor-Pinkas, receiver's requested message is encrypted using ElGamal:
  \[ \text{pk}: g, g^s = h \quad \text{Encrypt}(\text{pk}, m): g^t, h^{tu_b + v_b} \cdot m_b \]  
  \[ \text{sk}: s \]

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

\[ L \rightarrow \text{for instance: also used in H12, extra credit} \]
Noor-Pinkas Protocol

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

Sender Privacy: Information theoretic:

\[(g, g^s, g^t, g^c) \quad \text{for } c \neq ct\]

\[\Rightarrow (g, g^s, g^{tu+sv}, g^{cu+sv})\]

independent and uniformly random since \(u, v\) are encryption is a one-time pad.

Blinding of message

Simulator is trivial to construct: for \(c, u, v\), output random group element,

Simulate other components as in the real protocol.

OT requires algebraic assumptions to build. \((DH, factorization, LWE, etc.)\)

\(\Rightarrow\) must perform public-key operations \(\Rightarrow\) can be expensive in practice, if need to do many OTs

OT extensions [Ishai Kilian Naor Pinkas]: “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 (\(\sim\) few hundred ECC operations + AES for the rest)

  \(\Rightarrow\) but need to communicate proportionally to size of data/messages

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