FIT5124 Advanced Topics in Security

Lecture 6: Secure Computation Protocols II – Private Computation

Ron Steinfeld
Clayton School of IT
Monash University

April 2015

Secure Computation Protocols II

Secure Computation Protocols: How to achieve more complex security requirements beyond basic confidentiality or integrity?
We will look at two topics:

- Privacy in authentication and protocol integrity (prev. lecture): Zero-Knowledge protocols and applications to, e.g.
  - **Non-Transferability** of authentication: How to prove my identity without leaving a verifiable trace?
  - **Anonymity** in authentication: How to prove I belong to a group without revealing my identity?
  - **Catching Misbehaviour in General Protocols:** How to detect that a user doesn't follow a protocol?
- **Privacy in computation** (this lecture and next): general secure computation without a trusted party:, e.g.
  - Private data retrieval
  - Private data mining
  - Private e-voting...

Plan for this lecture

General Secure Computation and Applications:

- Example Motivation: Private data retrieval
- First example of a Private Computation protocol: Diffie-Hellman Based Oblivious Transfer (OT)
    - Completeness
    - 'Honest but curious' Privacy for client and server– based on simulation
    - Second example: strengthened Diffie-Hellman OT protocol
- Generalization: Private computation for any function
    - Definition
    - General protocol: Yao's protocol for secure 2-party computation of any function
- Efficient Implementation Frameworks and applications (mainly in tutorial / assignment)

Example Motivation: Private data retrieval

How to privately retrieve data?

- Server has N data items for sale (all same price).
- Client wants to buy and obtain one of them.

Security?

- **Privacy** for server: Don't reveal to client the items it didn't buy.
- **Privacy** for client: Don't reveal to server which item I retrieved/bought.

Q.: How to satisfy both of those (apparently contradictory) requirements simultaneously?

Possible A.: Use a private information retrieval (PIR) protocol!

First example of a Private Computation protocol: Diffie-Hellman Based Oblivious Transfer (OT)

1-of-2 Oblivious Transfer (OT): Most basic variant of PIR –

- Server has $N = 2$ items x_0, x_1.
- Client has a bit $s \in \{0, 1\}$ that selects one item, i.e. x_s.
- Each item $x_i \in \{0, 1\}$ is a single bit.

Setup of Diffie-Hellman OT protocol:

- Works in a cyclic group $G =< g >$ where Discrete-Logarithm (DL) problem is hard
- Public parameters: generator $g \in G$ for G, $h \hookleftarrow U(G)$ (no one knows DL x of h to base g).
- Denote order (size) of G by n (assumed prime).
  - e.g. (as in DSA digital signature standard): G a mutliplicative subgroup of $\mathbb{Z}_p^*$ (multiplicative group modulo p) for a prime p, where G is generated by $g \in \mathbb{Z}_p^*$, an element of prime order n, where n divides $p - 1$.

First example of a Private Computation protocol: Diffie-Hellman Based OT

Diffie-Hellman Based Oblivious Transfer (OT) Protocol:
Server (sender) has 2 items x_0, x_1, client (receiver) has a bit s and wants item x_s.

Sender		Receiver
$(x_0, x_1 \in \{0,1\})$		$(s \in \{0,1\})$
		$u \in_R \mathbb{Z}_n$ $h_s \leftarrow g^u$ $h_{1-s} \leftarrow h/g^u$
	$\xleftarrow{\quad h_0, h_1 \quad}$	
$u_0, u_1 \in_R \mathbb{Z}_n$ $(A_0, B_0) \leftarrow (g^{u_0}, h_0^{u_0} g^{x_0})$ $(A_1, B_1) \leftarrow (g^{u_1}, h_1^{u_1} g^{x_1})$		
	$\xrightarrow{\quad (A_0, B_0), (A_1, B_1) \quad}$	
		$x_s \leftarrow \log_g(B_s / A_s^u)$

FIGURE 7.2: $\binom{2}{1}$-OT protocol

Diffie-Hellman Based OT: Properties

Q: Why does it work?

A: Properties –

- Completeness: If Client and Server both follow protocol, Client will obtain desired item x_s.
- Privacy for server: Why can't the client also obtain the other server's bit x_{1-s}?
  - Assume first a honest but curious client – follows protocol steps, but analyzes received messages.
  - Intuition: bit x_{1-s} is encrypted with key $h_{1-s} = h/g^u = g^{x-u}$; client knows u but not $x = log_g(h)$ (DL)...
  - How to make intuition precise and prove it is correct? (next).
  - What if the client is malicious – client can change protocol steps to learn more? (later in this lecture.)
- Privacy for client: Why can't the server learn the client's selection s?
  - Intuition: Client cannot distinguish which of h_0, h_1 is g^u and which is h/g^u. Why?

Diffie-Hellman Based OT: Defining and proving Privacy

Intuition: Client does not learn anything about server's data (x_0, x_1) beyond what is revealed by protocol output (x_s).

Q: How to define and prove privacy for server?

A: Use simulation (similar to ZK) – Client can efficiently simulate the messages he sees in the protocol by itself, using only its input s and the protocol output x_s.

- Enough if client's simulation not exact but just computationally indistinguishable from the real protocol messages – i.e. computationally infeasible to distinguish simulation from real messages

Diffie-Hellman Based OT: Defining and proving Privacy

Efficient Simulator algorithm S for client's received messages in Diffie-Hellman OT protocol: Given $g, h \in G$, $s \in \{0,1\}$ and $x_s \in \{0,1\}$, S does following:

- Compute $h_s = g^u, h_{1-s} = h/g^u$, as in real protocol.
- Simulate $(A_s, B_s) = (g^{u_s}, h_s^{u_s} \cdot g^{x_s})$, for $u_s \hookleftarrow U(\mathbb{Z}_n)$, as in real protocol.
- Simulate $(A_{1-s}, B_{1-s}) = (g^{u_{1-s}}, h')$, for random $h' \hookleftarrow U(G)$ chosen independently.

Theorem (privacy for server). The above simulation of client's received messages is computationally indistinguishable from real protocol, assuming the hardness of Decision Diffie-Hellman (DDH) problem in G. (proof: see tute problem).

DDH Problem: Given $(g, g^a, g^b, y) \in G^4$ for $a, b \hookleftarrow U(\mathbb{Z}_n)$, distinguish REAL scenario ($y = g^{ab}$) from RAND scenario ($y \hookleftarrow U(G)$ independently).

Second example – strengthened Diffie-Hellman OT protocol

But, what if client is malicious and doesn't follow protocol? It can learn both x_0, x_1! How to strengthen the protocol for privacy against malicious clients?

General approach: Use ZK proofs to 'force' client to follow protocol!

- Problem: not very efficient in general.

Sometimes possible to get more efficient solutions...

Second example – strengthened Diffie-Hellman OT protocol

Strengthened Diffie-Hellman Based Oblivious Transfer (OT) Protocol (HL'10, Chapter 7): Server (sender) has 2 items x_0, x_1, client (receiver) has a bit σ and wants item x_σ.

1. The receiver R chooses $\alpha, \beta, \gamma \leftarrow_R \{1, \ldots, q\}$ and computes $\bar{a}$ as follows:
   a. If $\sigma = 0$ then $\bar{a} = (g^\alpha, g^\beta, g^{\alpha\beta}, g^\gamma)$.
   b. If $\sigma = 1$ then $\bar{a} = (g^\alpha, g^\beta, g^\gamma, g^{\alpha\beta})$.
   R sends $\bar{a}$ to S.
2. Denote the tuple $\bar{a}$ received by S by (x, y, z_0, z_1). Then, S checks that $x, y, z_0, z_1 \in \mathbb{G}$ and that $z_0 \neq z_1$. If not, it aborts outputting $\perp$. Otherwise, S chooses random $u_0, u_1, v_0, v_1 \leftarrow_R \{1, \ldots, q\}$ and computes the following four values:

   $$w_0 = x^{u_0} \cdot g^{v_0}, \qquad k_0 = (z_0)^{u_0} \cdot y^{v_0},$$
   $$w_1 = x^{u_1} \cdot g^{v_1}, \qquad k_1 = (z_1)^{u_1} \cdot y^{v_1}.$$

   S then encrypts x_0 under k_0 and x_1 under k_1. For the sake of simplicity, assume that one-time pad type encryption is used. That is, assume that x_0 and x_1 are mapped to elements of $\mathbb{G}$. Then, S computes $c_0 = x_0 \cdot k_0$ and $c_1 = x_1 \cdot k_1$ where multiplication is in the group $\mathbb{G}$.
   S sends R the pairs (w_0, c_0) and (w_1, c_1).
3. R computes $k_\sigma = (w_\sigma)^\beta$ and outputs $x_\sigma = c_\sigma \cdot (k_\sigma)^{-1}$.

Generalization: Private computation for any function

Private computation protocols have been extensively investigated and generalized to cover almost any imaginable scenario!

For instance, how to privately compute:

- Set Intersection: e.g. police investigators have a list of terrorist suspects, airline has a list of flight passengers.
- Comparison: e.g. e-auctions – bidders submit bids to auctioneer, want to hide bid from auctioneer unless winning bid.
- Summation: e.g. e-voting – voters submit bids, authority wants to add votes, voters don't want to reveal vote to authority.

Generalizing private comp. to any functionality $f = (f_1, f_2)$:

- Let $f = (f_1, f_2)$ be functions to be computed privately by parties P_1, P_2 resp. (e.g. $f_1(x = (x_0, x_1), y = s) = \text{null}, f_2(x = (x_0, x_1), y = s) = x_s$ for OT).

Goal: Given any functionality $f = (f_1, f_2)$, construct a secure computation protocol π for f.

Generalization: Private computation for any function

Generalizing the properties we want secure protocol π for $f = (f_1, f_2)$ to have:

Completeness: For any inputs (x, y), if parties P_1 and P_2 follow protocol π then at the end, P_1 has $f_1(x, y)$ and P_2 has $f_2(x, y)$.

Privacy against 'Honest but Curious' (aka 'semi-honest') P_1 and P_2: same simulation idea!

- Let $\text{view}_i^\pi(x, y, n)$ denote the messages received by P_i in protocol π for inputs x, y and security parameter n, along with P_i's input (and any random inputs).
- e.g. in OT protocol, $\text{view}_1^{OT} = (g, h, x = (x_0, x_1), u_0, u_1, (h_0, h_1))$ and $\text{view}_2^{OT} = (g, h, y = s, u, (A_0, B_0), (A_1, B_1))$.
- Let $\text{output}^\pi(x, y, n)$ be the joint output of both parties in protocol π.

Definition 2.2.1 (security w.r.t. semi-honest behavior): *Let $f = (f_1, f_2)$ be a functionality. We say that* π securely computes f in the presence of static semi-honest adversaries *if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that*

$$\{(S_1(1^n, x, f_1(x,y)), f(x,y))\}_{x,y,n} \stackrel{c}{\equiv} \{(\text{view}_1^\pi(x,y,n), \text{output}^\pi(x,y,n))\}_{x,y,n},$$

$$\{(S_2(1^n, y, f_2(x,y)), f(x,y))\}_{x,y,n} \stackrel{c}{\equiv} \{(\text{view}_2^\pi(x,y,n), \text{output}^\pi(x,y,n))\}_{x,y,n},$$

$x, y \in \{0,1\}^*$ *such that* $|x| = |y|$, *and* $n \in \mathbb{N}$.

Generalization: Private computation for any function – Malicious Attacks

Generalizing the properties we want secure protocol π for $f = (f_1, f_2)$ to have (cont.):

Malicious security definition more complex than 'honest but curious' (cannot directly adapt 'simulation') because:

- Malicious P_1 can ignore its input x_1 and substitute another x_1'.
- Malicious P_1 might be able to choose its x_1' to depend on y, then output may leak information on y!

Use alternative way of defining security: For security against malicious P_i, ideally want π protocol's security as good as security of an ideal OT protocol.

Q: What is the ideal protocol for functionality $f = (f_1, f_2)$?

Possible A: Using a trusted party to do the computations privately!

Generalization: Private computation for any function – Malicious Attacks

Ideal protocol π_{ideal} for $f = (f_1, f_2)$, inputs (x, y), trusted party P^*:

- Honest P_1, P_2 send x', y' respectively to P^*.
- $P*$ computes and sends $f_1(x', y')$ and $f_2(x', y')$ to P_1 and P_2, respectively.
- Parties return outputs z_1, z_2 respectively.

Notation:

- Let REAL(x, y, n) denote output pair (z_1, z_2) in real protocol π with party inputs x, y and security parameter n.
- Let IDEAL(x, y, n) denote output pair (z_1, z_2) in ideal protocol π_{ideal} with party inputs x, y and security parameter n.

Malicious Security for π: For all x, y, for every efficient malicious attacker A_{real} corrupting either P_1 or P_2 in real protocol π, there is an efficient malicious attacker S_{ideal} in ideal protocol π_{ideal} such that the output pair REAL(x, y, n) and IDEAL(x, y, n) are computationally indistinguishable.

Generalization: Private computation for any function

Generalizing the construction of OT to any function f:
General theoretical result: Any efficiently computable function f can also be efficiently computed privately!
Theorem [Yao82]: For any function $f = (f_1, f_2)$, there is a secure computation protocol π_{Yao} for f, built from an OT protocol and a symmetric-key encryption scheme (satisfying some natural properties).

- π_{Yao} is known as Yao's Garbled Circuit Protocol.
- The communication cost for π_{Yao} is proportional to $(\ell_{sym} \cdot |C_f| + \ell_{in1} \cdot \ell_{OT})$, where
  - ℓ_{sym} is the ciphertext/key length for the encryption scheme,
  - $|C_f$ is the size (number of gates) in the Boolean circuit for computing f,
  - ℓ_{in1} is the input (x) length for P_1,
  - ℓ_{OT} is the communication cost for the OT protocol.
- Using recent optimizations, can actually be practical for circuits up to thousands or even millions of gates, depending on security required (e.g. semi-honest or malicious).

Private computation for any function – Yao's Protocol

Yao's Garbled Circuit Protocol

We will look at the basic variant of Yao's protocol: secure only against semi-honest attacks. Only briefly mention (less efficient) variants against malicious attacks.

Setup and Notation:

- P_1 has n-bit input $x = (x_1, \ldots, x_n)$, P_2 has n-bit input $y = (y_1, \ldots, y_n)$.
- P_2 wants to compute a bit $f(x, y) \in \{0, 1\}$. (assume for now P_1 has no output).
- Assume that C_f is a Boolean circuit for function f.
- Let $w_1, \ldots, w_n$ denote input wires of C_f corresponding to input bits $x_1, \ldots, x_n$.
- Let $w_{n+1}, \ldots w_{2n}$ denote inputs wires of C_f corresponding to input bits $y_1, \ldots, y_n$.

We will use two ingredients:

- Symmetric-key encryption scheme (E, D) ($c = E_k(m)$ denotes ciphertext for m under key k, and $D_k(c) = m$ denotes decryption of this c).
  - Secure under chosen plaintext attack (IND-CPA security).
  - Additional property (for correctness of π_{Yao}): $D_K(c)$ outputs fail with high probability if c is a random string).
- 1-of-2 Oblivious Transfer (OT) protocol secure against semi-honest attacks (e.g. Diffie-Hellman protocol).

Private computation for any function – Yao's Protocol

Yao's Garbled Circuit Protocol

Basic Idea: P_1 computes and sends to P_2 a garbled ('encrypted') version $G(C_f)$ of circuit C_f.

- $G(C_f)$ is a special type of encryption for C_f that allows restricted computation.
- $G(C_f)$ has same number of gates and wires as C_f.
- To each wire w of $G(C_f)$, P_1 associates two random encryption keys k_w^0 and k_w^1, corresponding to two possible values for this wire.
- For each gate g in C_f, P_1 produces a garbled gate $G(g)$ for $G(C_f)$.

Private computation for any function – Yao's Protocol

Yao's Garbled Circuit Protocol

Basic property of Garbled gates $G(g)$ and wire keys:

- Let g be a gate with input wires w_1, w_2 and output wire w_3.
- Given keys $k_{w_1}^{a}$ and $k_{w_2}^{b}$ corresponding to values a, b for input wires w_1, w_2 of gate g and the garbled gate $G(g)$, it is possible to decrypt the key $k_{w_3}^{g(a,b)}$ corresponding to value $g(a, b)$ for gate output wire w_3.

But – no information is revealed about relation between wire keys and wire values!

- Exception for the output wire – $G(C_f)$ reveals link between output wire w_o keys and values ($k_{w_o}^{0} = 0$ and $k_{w_o}^{1} = 1$).

Hence, given keys for all input wire values x, y, P_2 can sequentially decrypt keys for gate output wire values, gate-by-gate. Until P_2 decrypts output wire key value – hence obtains output bit $f_2(x, y)$!

Generalization: Private computation for any function – Yao's Protocol

Yao's Garbled Circuit Protocol – How to garble a circuit?

Given circuit C_f, P_1 produces garbled circuit $G(C_f)$ as follows:

- For each wire w of C_f (and $G(C_f)$) pick two random keys k_w^0 and k_w^1 corresponding to values 0 and 1 resp. for w. (keys for symmetric encryption scheme (E, D)).
- For each gate g of C_f with input wires w_1, w_2 and output wire w_3, compute a garbled gate $G(g)$ consisting of the four 'garbled gate truth table' values (in a random order):

$$E_{k_{w_1}^0}\left(E_{k_{w_2}^0}\left(k_{w_3}^{g(0,0)}\right)\right), E_{k_{w_1}^0}\left(E_{k_{w_2}^1}\left(k_{w_3}^{g(0,1)}\right)\right), E_{k_{w_1}^1}\left(E_{k_{w_2}^0}\left(k_{w_3}^{g(1,0)}\right)\right), E_{k_{w_1}^1}\left(E_{k_{w_2}^1}\left(k_{w_3}^{g(1,1)}\right)\right).$$

- For output gate g in C_f, set $k_{w_3}^0 = 0$ and $k_{w_3}^1 = 1$.

Example garbled gate table $G(g)$ for an OR gate g:

input wire w_1	input wire w_2	output wire w_3	garbled computation table
k_1^0	k_2^0	k_3^0	$E_{k_1^0}(E_{k_2^0}(k_3^0))$
k_1^0	k_2^1	k_3^1	$E_{k_1^0}(E_{k_2^1}(k_3^1))$
k_1^1	k_2^0	k_3^1	$E_{k_1^1}(E_{k_2^0}(k_3^1))$
k_1^1	k_2^1	k_3^1	$E_{k_1^1}(E_{k_2^1}(k_3^1))$

Private computation for any function – Yao's Protocol

Yao's Garbled Circuit Protocol – How to use garbled circuit?

So far, P_1 sent P_2 the garbled circuit $G(C_f)$. If P_2 would have

- keys $k_{w_1}^{x_1}, \ldots, k_{w_n}^{x_n}$ corresponding to P_1's input x, and
- keys $k_{w_{n+1}}^{y_1}, \ldots, k_{w_{2n}}^{y_n}$ corresponding to P_2's input y,

then P_1 can compute with $G(C_f)$ the desired output value $f_2(x, y)$.

Q: How does P_2 get those keys?

A: In the case of $k_{w_1}^{x_1}, \ldots, k_{w_n}^{x_n}$: P_1 just sends them to P_2.

- Does not reveal anything on x since $k_{w_i}^{x_i}$ chosen randomly by P_1.

What about $k_{w_{n+1}}^{y_1}, \ldots, k_{w_{2n}}^{y_n}$ corresponding to P_2's input y?

- P_1 cannot directly send them, as he doesn't know y_j's.
- P_1 could send both keys $k_{w_j}^0, k_{w_j}^1$ for all $j = n+1, \ldots, 2n$, but this would allow P_2 to compute $f_2(x, y')$ for any y'...

We already know a solution: 1-of-2 OT for each y_j!

Yao's Garbled Circuit Protocol – Summary

PROTOCOL 3.4.1 (Yao's Two-Party Protocol)

- **Inputs:** P_1 has $x \in \{0,1\}^n$ and P_2 has $y \in \{0,1\}^n$.
- **Auxiliary input:** A boolean circuit C such that for every $x, y \in \{0,1\}^n$ it holds that $C(x,y) = f(x,y)$, where $f: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. We require that C is such that if a circuit-output wire leaves some gate g, then gate g has no other wires leading from it into other gates (i.e., no circuit-output wire is also a gate-input wire). Likewise, a circuit-input wire that is also a circuit-output wire enters no gates.
- **The protocol:**

1. P_1 constructs the garbled circuit $G(C)$ as described in Section 3.3, and sends it to P_2.
2. Let $w_1, \ldots, w_n$ be the circuit-input wires corresponding to x, and let $w_{n+1}, \ldots, w_{2n}$ be the circuit-input wires corresponding to y. Then,
   a. P_1 sends P_2 the strings $k_1^{x_1}, \ldots, k_n^{x_n}$.
   b. For every i, P_1 and P_2 execute a 1-out-of-2 oblivious transfer protocol in which P_1's input equals (k_{n+i}^0, k_{n+i}^1) and P_2's input equals y_i.
   The above oblivious transfers can all be run in parallel.
3. Following the above, P_2 has obtained the garbled circuit and $2n$ keys corresponding to the $2n$ input wires to C. Party P_2 then computes the circuit, as described in Section 3.3, obtaining $f(x, y)$.

Yao's Garbled Circuit Protocol – Security

Possible to prove semi-honest security: **Theorem.** Yao's protocol achieves semi-honest security against client or server, assuming the OT is secure against semi-honest attack and the encryption scheme is secure under chosen plaintext attack (IND-CPA security).

Will not cover proof in detail (see HL, Chapter 3).

Intuition:

- Security Against P_1: P_1 just sees the OT protocol message from P_2 - security follows from OT protocol privacy for P_2 (use OT simulator for P_1's view).
- Security Against P_2: P_2 receives garbled circuit $G(C_f)$ and keys corresponding to P_1's input x. Simulator for P_2's view just sends fake garbled circuit (gates only encrypt same output key for all 4 input key combinations), and output gate encrypts $f_2(x, y)$ for all 4 input combinations.
  - Idea: P_2 cannot distinguish fake from real garbled circuit, since it only gets keys for one input combination of each gate. Other gate outputs are indistinguishable by IND-CPA security of encryption scheme. Also need to rely on OT security against P_2.

Private computation for any function – Yao's Protocol

Yao's Protocol – How to secure against malicious parties?

Current techniques for strengthening Yao's protocol for security against malicious attacks generally add a significant cost overhead. We Will not cover in detail.

Basic idea of common approach (see [HL, Chapter 4]):

- Use a strengthened OT subprotocol
- P_2 verifies that P_1 garbled C_f correctly using cut and choose:
  - P_1 sends to P_2 multiple (independent) garbled circuits $G(C_f)_i$ for $i = 1, \ldots, N$.
  - P_2 asks P_1 to open (provide all keys) for a random half of the garbled $G(C_f)_i$'s, and checks them for correctness.
  - If all opened circuits are correct P_2 computes $f(x, y)$ using all remaining unopened circuits and takes majority as output.
  - Idea: extremely unlikely that a majority of unopened circuits incorrect, yet all opened circuits correct!
  - But, other complications need to be handled, e.g. need to check that P_2, P_1 use same inputs for all garbled circuits!

Private computation for any function – Yao's Protocol

Yao's Garbled Circuit Protocol – Implementation Frameworks

Significant work on optimized implementations of Yao's protocol

Several implementation frameworks available (more in tute/assignment), e.g.:

- Fairplay (2004): http://www.cs.huji.ac.il/project/Fairplay/Fairplay.html
  - Compiler from 'C style' function f specfication language (SFDL) to Boolean circuit language (SHDL)
  - Compiler from circuit language (SHDL) to a Yao protocol (semi-honest).
  - Sample performance: Comparing two 32-bit integers (254 gates) – 1.25 sec on 2.4GHz machines.
- TASTY (2010): https://github.com/tastyproject/tasty
  - Improved performance in some applications, combining Yao with other techniques
  - Sample performance: 32k gates – 6 sec setup, 1 sec online on 3GHz machines.
- Might Be Evil (2011): https://mightbeevil.org
  - Allow Combination of high level and circuit level Java code for f specification.
  - Optimize Yao approach
  - Sample performance: 100k gates/sec, Hamming distance on 900 bits: 50msec.