CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 22
April 9, 2012

Oblivious Transfer of One Secret Out of Two

Privacy-Preserving Multiparty Computation

The Millionaire's Problem

A General Security Model

Private Circuit Evaluation

Homomorphic Encryption

One-of-two oblivious transfer

In *one-of-two oblivious transfer*, Alice has two secrets, s_0 and s_1.

Bob always gets exactly one of the secrets, each with probability $1/2$.

Alice does not know which one Bob gets.

The locked box protocol is one way to implement one-of-two oblivious transfer.

Another is based on a public key cryptosystem (such as RSA) and a symmetric cryptosystem (such as AES).

This protocol given next does not rely on the cryptosystems being commutative.

Intuitive idea

In this protocol, Alice chooses two PKS key pairs and sends the public keys to Bob.

Bob chooses a random key k for a symmetric cryptosystem, encrypts it with one of Alice's two keys chosen at random, and sends the ciphertext to Alice.

Alice decrypts Bob's ciphertext using both of her decryption keys and obtains two numbers $\{k_0, k_1\}$. One of them is Bob's k; the other is garbage. She encrypts one secret using k_0 and one using k_1 and sends to Bob.

Bob decrypts the one that was encrypted with k.

A one-of-two OT protocol

	Alice		Bob
1.	Secrets s_0 and s_1. Choose two PKS key pairs (e_0, d_0) and (e_1, d_1).	$\xrightarrow{e_0, e_1}$	
2.		$\xleftarrow{c}$	Choose random key k for symmetric cryptosystem $(\hat{E}, \hat{D})$. Choose random $b \in \{0, 1\}$. Compute $c = E_{e_b}(k)$.
3.	Let $k_i = D_{d_i}(c)$, $i \in \{0, 1\}$. Choose $b' \in \{0, 1\}$. Let $c_i = \hat{E}_{k_i}(s_{i \oplus b'})$, $i \in \{0, 1\}$.	$\xrightarrow{c_0, c_1}$	
4.			Output $s = s_{b \oplus b'} = \hat{D}_k(c_b)$.

Analysis

In step 2, Bob encrypts a randomly chosen key k for the symmetric cryptosystem using one of the PKS encryption keys that Alice sent him in step 1.

He then selects one of the two encryption keys from Alice, uses it to encrypt k, and sends the encryption to Alice.

In step 3, Alice decrypts c using both decryption keys d_0 and d_1 to get k_0 and k_1.

One of the k_i is Bob's key k (k_b to be specific) and the other is garbage, but because k is random and she doesn't know b, she can't tell which is k.

Analysis (cont.)

She then encrypts one secret with k_0 and the other with k_1, using the random bit b' to ensure that each secret is equally likely to be encrypted by the key that Bob knows.

In step 4, Bob decrypts the ciphertext c_b using key his key $k = k_b$ to recover the secret $s = s_{b \oplus b'}$.

He can't decrypt the other ciphertext $c_{1 \oplus b}$ since he doesn't know the key $k_{1 \oplus b}$ used to produce it, nor does he know the decryption key $d_{1 \oplus b}$ that would allow him to find it from c.

A subtle security problem

Unfortunately, this protocol has a subtle security problem. We claimed that Alice doesn't know Bob's value b after step 2. But why should that be true?

c the encryption of k using E_{e_0} or E_{e_1}. We would need to know that outputs of $E_{e_0}(k)$ for random k are indistinguishable from outputs of $E_{e_1}(k)$. We have no grounds for believing that.

For example, we could take our favorite PKS and construct a new one where $E'_e(k) = e \cdot E_e(k)$. This is just as secure as the original since e is already known to the adversary. However, the ciphertext here reveals the public key used for encryption.

Another 1-of-2 OT protocol using blinding [1]

	Alice		Bob
1.	Secrets s_0 and s_1. Choose RSA key (n, e, d). Let $y_i = E_e(s_i)$, $i \in \{0, 1\}$.	$\xrightarrow{n,e,y_0,y_1}$	
2.		$\xleftarrow{c}$	Choose random $b \in \{0, 1\}$. Choose random $r \in \mathbf{Z}_n^*$. Compute $c = y_b E_e(r) \bmod n$.
3.	Let $c' = D_d(c) \equiv s_b r \pmod{n}$.	$\xrightarrow{c'}$	
4.			Output $c' r^{-1} \bmod n = s_b$.

[1] This protocol is adapted from notes by David Wagner, U.C. Berkeley, CS276, lecture 29, May 2006.

Analysis

This protocol is much simpler.

- In step 1, Alice sends Bob encryptions of both secrets.
- In step 2, Bob chooses one of Alice's encryptions, blinds it, and returns the result to Alice.
- In step 3, Alice decrypts whatever Bob sends her, which allows Bob to unblind the decryption and recover the secret he chose in step 2.

Alice's other secret is safe assuming semi-honest parties (see lecture 19) as long as RSA is secure under a limited chosen ciphertext attack (since that is what Alice permits in step 3).

Bob's blinding prevents Alice from knowing which secret he learned.

Privacy-Preserving Multiparty Computation

Privacy

We have looked at many protocols whose goal is to keep Alice's information secret from an adversary, or sometimes even from Bob himself.

We now look at other protocols whose goal is to control the release of information about Alice's secret. Just enough information should be released to carry out the purpose of the protocol but no more.

This will become clearer with an example.

The Millionaire's Problem

The Millionaire's Problem

Alice and Bob want to know who is the richer without revealing how much they are actually worth.

Alice is worth I million dollars; Bob is worth J million dollars.

They want to determine whether or not $I \geq J$, but at the end of the protocol, neither should have learned any more about the other person's wealth than is implied by the truth value of the predicate $I \geq J$.

Privacy-preserving multiparty computation

The Millionaire's problem, introduced by Andy Yao in 1982, began the study of *privacy-preserving multiparty computation*.

Another example is vote-counting.

Each voter has an input $v_i \in \{0, 1\}$ indicating their no/yes vote on an issue.

The goal is to collectively compute $\sum v_i$ while maintaining the privacy of the individual v_i.

A solution to Yao's problem

For simplicity, assume that $I, J \in \{1, 2, \ldots, 10\}$.

Let N be a security parameter, and assume that Alice has public and private RSA keys (e, n) and (d, n), respectively, where $n = \bar{p}\bar{q}$, and $|\bar{p}| \approx |\bar{q}| \approx \frac{N}{2}$.

A protocol that intuitively works is shown on the next slide.[2]

[2]Adapted from web page "Solution to the Millionaire's Problem".

The protocol

	Alice		Bob
1.			Choose x of length N.
			Let $C = E_{(e,n)}(x)$.
		$\xleftarrow{m}$	Let $m = (C - J + 1) \bmod n$.
2a.	$Y_i = D_{(d,n)}(m+i-1)$, $i \in [1, 10]$. [Note: $Y_J = x$.]		
2b.	Choose prime p of length $N/2$ s.t. $\lvert Z_i - Z_j \rvert \geq 2$ for $i \neq j$, where $Z_i = (Y_i \bmod p)$, $i \in [1, 10]$.		
2c.	Let $W_i = (Z_i + (i > I)) \bmod p$, $i \in [1, 10]$.	$\xrightarrow{p, W_1, \ldots, W_{10}}$	
3.		$\xleftarrow{\text{result}}$	$\text{result} = (W_J \equiv x \pmod p)$.

Verbal description

Alice decrypts $m, m+1, \dots, m+9$ to get $Y_1, \dots, Y_{10}$.

Y_J is Bob's secret, x, but Alice doesn't know which it is since all of the Y_i's "look" random.

She reduces them all mod random prime p to get $Z_1, \dots, Z_{10}$. Note that $Z_J = x \bmod p$ and the other Z_i's look random.

Finally, she adds 1 $(\operatorname{mod} p)$ to each of the numbers Z_i for which i is greater than her own wealth I. If she adds 1 to Z_J, this means that $J > I$; if not $J \leq I$.

Bob can tell which is the case from the numbers that Alice sends him in step 2c. Namely, if $W_J \equiv x \pmod{p}$, this means that 1 was not added, so $I \geq J$. Otherwise, $I < J$.

Detailed description

	Alice		Bob
1.		$\xleftarrow{m}$	Choose x of length N. Let $C = E_{(e,n)}(x)$. Let $m = (C - J + 1) \bmod n$.
2a.	$Y_i = D_{(d,n)}(m+i-1)$, $i \in [1, 10]$. [Note: $Y_J = x$.]		

$C = (m + J - 1) \bmod n$ is the encryption of Bob's random secret x.

The numbers in $\mathcal{M} = \{m \bmod n, \dots, (m + 9) \bmod n\}$ are "random-looking," and all are possible ciphertexts. Why?

Alice knows that $C \in \mathcal{M}$ but doesn't know which element it is.

After decryption, she knows that some $Y_i = x$ but not which one.

Detailed description (cont.)

	Alice	Bob
2b.	Choose prime p of length $N/2$ s.t. $\lvert Z_i - Z_j \rvert \geq 2$ for $i \neq j$, where $Z_i = (Y_i \bmod p)$, $i \in [1, 10]$.	

The numbers in $\mathcal{Y} = \{Y_1, \ldots, Y_{10}\}$ have no particular pattern. In all likelihood, no pair are at all close together.

Similarly, for most choices of p, no pair of Z_j's will be close.

Detailed description (cont.)

	Alice		Bob
2c.	Let $W_i = (Z_i + (i > I)) \bmod p$, $i \in [1, 10]$.	$\xrightarrow{p, W_1, \ldots, W_{10}}$	
3.		$\xleftarrow{\text{result}}$	$\text{result} = (W_J \equiv x \pmod{p})$.

The W_i's are distinct and separated by at least 2, so j is the unique i such that $(W_i - x) \bmod p \in \{0, 1\}$. I don't know why this uniqueness condition is needed. Perhaps the intention is that Alice shuffle the W_i before sending.

Since $Y_J = x$, then $Z_J \equiv x \pmod{p}$.

Hence, if $W_J \equiv x \pmod{p}$, then $J \leq I$, otherwise $J > I$.

Privacy

Clearly, all that Alice learns from Bob is a set of random-looking numbers $m, \dots, m+9$, one of which corresponds to Bob's wealth J, but she has no way of telling which, since any number in $\mathbf{Z}_n^*$ is the RSA encryption of some plaintext message.

Bob on the other hand receives p and $W_1, \dots, W_{10}$ from Alice in step 2. However, he does not know any Z_i for $i \neq J$ since he cannot decrypt the corresponding numbers $m + i - 1$.

He also cannot recover Y_i from W_i because of the information loss implicit in the "mod p" operation. Thus, he also learns nothing about Alice's wealth I except for the value of the predicate $I \geq J$.

We remark that this protocol works only in the semi-honest model in which both Alice and Bob follow their protocol, but both will try to infer whatever they can about the others secrets after the fact.

A General Security Model

How can we define multiparty security?

How to define security in a multiparty protocol is far from obvious.

For example, in the millionaire's problem, there is no way to prevent either Alice or Bob from lying about their wealth, nor is it possible to prevent either of them from voluntarily giving up secrecy by broadcasting their wealth.

Thus, we can't hope to find a protocol that will prevent all kinds of cheating.

Ideal versus real protocol security model

What we do instead is to compare a given "real" protocol with a corresponding very simple "ideal" protocol involving a trusted third party.

The real protocol should simulate the ideal protocol, much the same as the simulator of a zero knowledge proof system simulates the real interaction between prover and verifier.

The real protocol is deemed to be secure if any bad things that can happen in the real protocol are also possible in the ideal protocol.

Example of an ideal protocol

The ideal protocol for the millionaire's problem has just two steps:

- Step 1: Alice and Bob send their secrets I and J, respectively, to the trusted party across a private, secure channel.
- Step 2: the trusted party computes the value of the predicate $I \geq J$ and sends the result back to both Alice and Bob.

The goal of the real protocol is that Alice and Bob don't learn any more than they could learn in the ideal protocol.

What does an ideal protocol compute?

What does an ideal multiparty protocol compute? Suppose there are m parties to the protocol, $P_1, \ldots, P_m$.

Each P_i has a private input x_i and receives a private output y_i.

We say that F is a (multiparty) *functionality* if F is a random process that maps m inputs to m outputs.

As a special case, we say that *F is deterministic* if the m outputs are uniquely determined by the m inputs.

The millionaire's problem can be expressed succinctly as the problem of securely computing the (deterministic) functionality

$$F(I, J) = ((I \geq J), (I \geq J))$$

in the semi-honest model.

Simple application of oblivious transfer

Consider the problem of privately evaluating a Boolean function $f(x, y)$, where x is private to Alice and y is private to Bob. This corresponds to privately computing the functionality

$$F(x, y) = (f(x, y), f(x, y)).$$

We use a slight variant of the one-out-of-two secrets oblivious transfer protocol presented last time:

In OT^2_1, the secrets are numbered s_0 and s_1. Bob requests and gets the secret of his choice, but Alice does not learn which secret he got.

This can be generalized to the case k secrets, where OT^k_1 lets Bob choose one out of k.

The protocol

Here's the protocol.

1. Alice, with private input $x \in \{0,1\}$, prepares a table T:

y	$f(x,y)$
0	$f(x,0)$
1	$f(x,1)$

   She doesn't know y, but she does know that the correct value $f(x,y)$ is in her table. It's either $f(x,0)$ or $f(x,1)$.
2. Bob, with private input y, obtains line y of the table using OT^2_1. Bob outputs $f(x,y)$ without learning x.
3. Bob sends $f(x,y)$ to Alice, who also outputs it.

Remarks

While this functionality seems almost too trivial to be interesting, it's really not.

For example, if $f(x, y) = x \wedge y$ and Alice knows $x = 0$, then the answer $f(x, y)$ does not tell her Bob's value y, so it's important that the protocol also not leak y in this case.

Similarly, when Bob requests the value corresponding to row 0, he gets no information about x when the result $f(x, 0) = 0$ comes back.

(In fact he knew that already before getting row 0 from Alice.)

Private Circuit Evaluation

Privacy-preserving Boolean function evaluation

We now generalize the simple example to any function $\bar{z} = f(\bar{x}, \bar{y})$, where $\bar{x}$, $\bar{y}$, and $\bar{z}$ are bit strings of lengths n_x, n_y, and n_z, respectively, and $f(\bar{x}, \bar{y})$ is computed by a polynomial size Boolean circuit with $n_x + n_y$ input wires and n_z output wires.

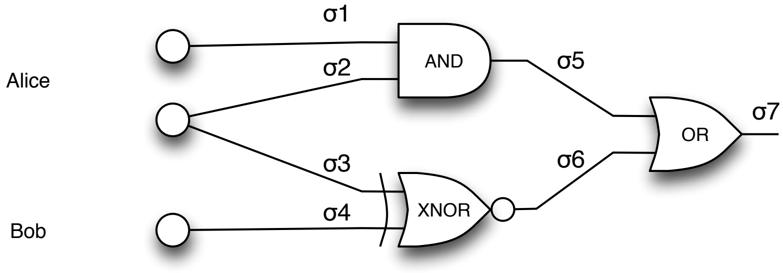
The corresponding functionality is

$$F(\bar{x}, \bar{y}) = (f(\bar{x}, \bar{y}), f(\bar{x}, \bar{y})).$$

Alice furnishes the (private) input data to the first n_x input wires. Bob furnishes the input data for the remaining n_y input wires.

Alice and Bob should learn nothing about each others inputs or the intermediate values of the circuit, other than what is implied by their own inputs and the n_z output values.

A Boolean circuit



Non-private circuit evaluation

A non-private evaluation of the circuit associates a Boolean value σ_w with each wire of the circuit.

The input wires are associated with the corresponding input values.

Let G be a gate with input wires u and v and output wire w that computes the Boolean function $g(x, y)$.

If σ_u is the value on wire u and σ_v the value on wire v, then the value on wire w is $g(\sigma_u, \sigma_v)$.

A complete evaluation of the circuit first assigns values to the input wires and then works its way down the circuit, assigning a value to the output wire of any gate whose inputs have already received values.

Private circuit evaluation

To carry out the evaluation privately, we split the value σ_w on each wire w into two random shares a_w and b_w, where $\sigma_w = a_w \oplus b_w$.

Neither share alone gives any information about σ_w, but together they allow σ_w to be computed.

After having computed shares for all of the wires, Alice and Bob exchange their shares a_w and b_w for each output wire w.

Obtaining the shares

We now describe how Alice and Bob obtain their shares while maintaining the desired privacy.

There are three cases, depending on whether w is an input wire controlled by Alice, an input wire controlled by Bob, or the output wire of a gate G.

Alice's input wires

1. Input wire controlled by Alice:

   Alice knows σ_w.

   She generates a random share $a_w \in \{0, 1\}$ for herself and sends Bob his share $b_w = a_w \oplus \sigma_w$.

Bob's input wires

2. Input wire controlled by Bob:

   Bob knows σ_w.

   Alice chooses a random share $a_w \in \{0, 1\}$ for herself.

   She prepares a table T:

σ	$T[\sigma]$
0	a_w
1	$a_w \oplus 1$.

   Bob requests $T[\sigma_w]$ from Alice via OT^2_1 and takes his share to be $b_w = T[\sigma_w] = a_w \oplus \sigma_w$.

Obtaining shares for gate output wires

3. Output wire of a gate G:
   Let G have input wires u, v and compute function $g(x, y)$.
   Alice chooses random share $a_w \in \{0,1\}$ for herself.
   She computes the table

$$\begin{aligned} T[0,0] &= a_w \oplus g(a_u, a_v) \\ T[0,1] &= a_w \oplus g(a_u, a_v \oplus 1) \\ T[1,0] &= a_w \oplus g(a_u + 1, a_v) \\ T[1,1] &= a_w \oplus g(a_u + 1, a_v + 1) \end{aligned}$$

   (Equivalently, $T[r,s] = a_w \oplus g(a_u \oplus r, a_v \oplus s)$.)

   Bob requests $T[b_u, b_v]$ from Alice via OT_1^4 and takes his share to be $b_w = T[b_u, b_v] = a_w \oplus g(\sigma_u, \sigma_v)$.

Remarks

1. Alice and Bob's shares for w are both independent of σ_w.
   - Alice's share is chosen uniformly at random.
   - Bob's share is always the XOR of Alice's random bit a_w with something independent of a_w.
2. This protocol requires n_y executions of OT_1^2 to distribute the shares for Bob's inputs, and one OT_1^4 for each gate.[3]
3. This protocol assumes semi-honest parties.
4. This protocol generalizes readily from 2 to m parties.
5. Bob does not even need to know what function each gate G computes. He only uses his private inputs or shares to request the right line of the table in each of the several OT protocols.

[3]Note: The n_y executions of OT_1^2 can be eliminated by having Bob produce the shares for his input wires just as Alice does for hers. Our approach has the advantage of being more uniform since Alice is in charge of distributing the shares for all wires.

Private circuit evaluation using garbled circuits

A very different approach to private circuit evaluation is the use of *garbled circuits*.

The idea here is that Alice prepares a garbled circuit in which each wire has associated with it a tag corresponding to 0 and a tag corresponding to 1.

Associated with each gate is a template that allows the tag that represent the correct output value to be computed from the tags representing the input values.

This is all done in a way that keeps hidden the actual values that the tags represent.

A sketch of the protocol

After creating the circuit, Alice, who knows all of the tags, uses OT_1^2 to send Bob the tags corresponding to values on the input wires that he controls.

She also sends him the tags corresponding to the values on the input wires that she controls.

Bob then evaluates the circuit all by himself, computing the output tag for each gate from the tags on the input wires.

At the end, he knows the tags corresponding to the output wires.

Alice knows which Boolean values those tags represent, which she sends to Bob (either before or after he has evaluated the circuit).

In this way, Bob learns the output of the circuit, which he then sends to Alice.

Role of the tags

The scrambled gate is a 4-line table giving the output tag corresponding to each of the possible 4 input values.

Each line of the table is encrypted differently.

The input tags to the gate allow the corresponding table item to be decrypted.

Evaluating the circuit then amounts to decrypting ones way though the circuit, gate by gate, until getting the output tag.

Remarks

1. The OT^2_1 protocol steps used to distribute the tags for the wires that Bob controls keeps his inputs private from Alice. The privacy of Alice's inputs and intermediate circuit values from Bob relies on the encryption function used to hide the association between tags and values.
2. The security of the protocol relies on properties of the encryption function that we have not stated.
3. This protocol requires only n_y executions of OT^2_1 and hence should be considerably faster to implement than the share-based protocol.
4. This protocol also assumes semi-honest parties.
5. Doesn't easily generalize to more than two parties.
6. Bob doesn't need to know the function each gate computes. He only needs the associated templates.

Homomorphic Encryption

Homomorphic property

An encryption function $E(\cdot)$ is said to be *homomorphic* with respect to an operator $\odot$ if one can compute $E(x \odot y)$ from $E(x)$ and $E(y)$ without decrypting either ciphertext.

Several well-known cryptosystems have a homomorphic property.

RSA $E(x \cdot y) \equiv (xy)^e \equiv x^e \cdot y^e \equiv E(x) \cdot E(y) \pmod{n}$.

ElGamal

$$\begin{aligned} E(xy) &= (g^{r_x+r_y}, (xy)h^{r_x+r_y}) \\ &= (g^{r_x}, xh^{r_x}) \cdot (g^{r_y}, yh^{r_y}) \\ &= E(x) \cdot E(y), \end{aligned}$$

where $\cdot$ on pairs means componentwise multiplication.

Goldwasser-Micali cryptosystems

Goldwasser-Micali Public key is $n = pq$, $y \in \mathrm{QNR}_n$, $E(b) = r^2 y^b \bmod n$ for random r.

$$\begin{aligned} E(b_1) \cdot E(b_2) \bmod n &= (r_1^2 y^{b_1})(r_2^2 y^{b_2}) \bmod n \\ &= (r_1 r_2)^2 y^{b_1 + b_2} \bmod n \end{aligned}$$

While this is not equal to $E(b_1 \oplus b_2) = (r_1 r_2)^2 y^{b_1 \oplus b_2}$, is equal to $r^2 y^{b_1 \oplus b_2}$ for some possibly different choice of r. Hence, $E(b_1) \cdot E(b_2)$ is a valid encryption of $b_1 \oplus b_2$, as desired.

Benaloh cryptosystem

Benaloh This generalizes the Goldwasser-Micali scheme to give

$$E\left(\sum_{i=1}^{k} b_i\right) = \prod_{i=1}^{k} E(b_i)$$

As with Goldwasser-Micali, this is a randomized encryption scheme, so equality means only that the product is one of the possible encryptions of the sum of the b_i's.

Application to secret ballot elections

Homomorphic encryption can be applied to verifiable secret ballot elections.

- Each voter i has a vote b_i. To cast the vote, the voter computes $c_i = E(b_i)$ using the public encryption function of the voting authority and submits c_i. Here we assume the Benaloh scheme.
- The voting authority publishes the c_i's for all of the voters, gives the tally $t = \sum b_i$, and gives the random string that shows $E(t) = \prod_{i=1}^{k} E(b_i)$.
- Any voter can check that her own vote appears and can check the total, but she cannot determine anyone else's votes.
- This makes sense in the situation where the voting authority is trusted to respect the privacy of votes but not to count the votes correctly.