Lecture Notes 21

53 The Millionaire’s Problem

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

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 \( I \geq J \).

For simplicity, we assume that \( I \) and \( J \) are both in the set \{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 in Figure 53.1.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( Y_i = D_{(d,n)}(m + i - 1), i \in [1, 10]. ) [Note: Y_J = x.]</td>
<td>Choose random ( x ) such that (</td>
</tr>
<tr>
<td>( Y_i ) is Bob’s secret ( x ), and the ( Y_i )'s look random. Pick a random prime ( p ) s.t. (</td>
<td>p</td>
</tr>
<tr>
<td>( W_i ) is random but ( W_J ) looks like ( x ).</td>
<td>( p, W_1, \ldots, W_{10} ) ( \xrightarrow{\text{result}} ) result = ((W_J \equiv x \mod p)).</td>
</tr>
</tbody>
</table>

Figure 53.1: Solution to Yao’s Millionaire’s problem.

In step 1, Bob sends Alice a random-looking number \( m \). The number \( m + J - 1 \) is the encryption of Bob’s secret \( x \). Alice decrypts the numbers \( m, m+1, \ldots, m+9 \) to get corresponding \( Y_1, \ldots, Y_{10} \). The number \( Y_J \) is Bob’s secret \( x \), but Alice doesn’t know which it is since all of the \( Y_i \)'s “look” random. She then reduces them all mod a random prime \( p \), resulting in \( Z_1, \ldots, Z_{10} \). Note that \( Z_J = x \mod p \) and the other \( Z_i \)'s look random. Finally, she adds \( 1 \mod p \) 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 \( J \geq I \); if not \( J \leq I \). Bob can tell with is the case from the numbers that Alice sends him in step 2. Namely, if \( W_J \equiv x \mod p \), this means that 1 was not added, so \( I \geq J \). Otherwise, \( I < J \).

Clearly, all that Alice learns from Bob is a set of random-looking numbers \( m, \ldots, m + 9 \), one of which corresponds to Bob’s wealth \( J \), but she has no way of telling which, since any number

\[ \text{Adapted from web page “Solution to the Millionaire’s Problem”} \]
in \( \mathbb{Z}_n^* \) is the RSA encryption of some plaintext message. Bob on the other hand receives \( p \) and \( W_1, \ldots, W_{10} \) from Alice in step 2. However, he does not know any \( Z_i \) for \( i \neq J \) since he cannot 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 other’s secrets after the fact.

### 54 Ideal versus Real Protocol Security Model

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 voluntarily giving up secrecy by broadcasting their wealth. Thus, we can’t hope to find a protocol that will prevent all kinds of cheating. What we do instead is to compare a given “real” protocol with a corresponding very simple “ideal” protocol involving a trusted party. The idea is that 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.

For example, the ideal protocol for the millionaire’s problem has just two steps: In step 1, Alice and Bob send their secrets \( I \) and \( J \), respectively, to the trusted party across a private, secure channel. In 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.

### 55 Functionality

But what is it that an ideal multiparty protocol computes? 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 (multi-party) 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 simply as the problem of securely computing the (deterministic) functionality \( F(I, J) = ((I \geq J), (I \geq J)) \) in the semi-honest model.

### 56 Security Parameters for Multiparty Protocols

There is no simple choice of the “right” security model for multiparty computations. As with other aspects of cryptography, different kinds of threats are deemed significant in different situations. We list below some of the parameters that are important in the literature on secure multiparty computation.

**Channels:**

- Reliable but not private.
- Private channel.
Computational limitations:

- Adversary assume to be a probabilistic polynomial time Turing machine.
- Adversary has unbounded computational power.

Choice of corrupted parties:

- Adversary may dynamically corrupt parties as the computation progresses.
- Non-adaptive: Adversary chooses dishonest parties before the protocol begins (but corrupted parties are not known to the honest parties).

Adversary actions:

- Malicious adversary: Actively disrupts the protocol.
- Semi-honest: Follows the protocol but gathers information (which it may share with other corrupted parties).

Protocol disruption:

- Adversary may prematurely terminate the protocol.
- No premature abort.

Bounds on corruption: The number of dishonest parties may be bounded, e.g., < n/2 or < n/3.

57 Oblivious Transfer

$OT^k_1$, 1-out-of-$k$ oblivious transfer, is the functionality

$$ OT^k_1((\sigma_1, \ldots, \sigma_k), i) = (\lambda, \sigma_i) $$

where $\sigma_1, \ldots, \sigma_k \in \{0, 1\}$ and $i \in \{1, \ldots, k\}$. Here, the first party (Alice) has $k$ secret bits $\sigma_1, \ldots, \sigma_k$. Bob has a secret index $i$. At the end of the protocol, Bob learns only $\sigma_i$ and Alice learns nothing ($\lambda$). In particular, Alice does not know which of her bits was learned by Bob, and Bob does not learn $\sigma_j$ for any $j \neq i$.

Oblivious transfer is central to many of the constructions for secure multiparty computation. We give a protocol for $OT^k_1$ in the semi-honest model. Our construction assumes an enhanced trapdoor permutation $\{f_\alpha : D_\alpha \rightarrow D_\alpha\}_{\alpha \in I}$. (See appendix of Volume 2 for technical information on exactly what it means to be enhanced.) Roughly speaking, given a security parameter $1^n$ and a random string $r$, there is an algorithm $G(1^n, r)$ that returns a pair $(\alpha, t)$, where $\alpha$ is the index of a trapdoor permutation $f_\alpha$ and $t$ is a trapdoor. (Think of RSA key generation, where we randomly generate a public and private key pair.) Moreover, there is a feasible algorithm for computing $f_\alpha^{-1}(y)$ given $\alpha$ and $x$, and there is a feasible algorithm for computing $f_\alpha^{-1}(y)$ given $t$ and $y$. We also assume that $b$ is a hard-core predicate for $f_\alpha$.

Our construction is shown in Figure 57.1. The idea behind the protocol is that Bob sends Alice a $k$-tuple $(y_1, \ldots, y_k)$ of numbers. All $y_j$ are random except for $y_i$; $y_i$ is the encryption of a random number $x_i$. Alice decrypts them all to get $z_1, \ldots, z_k$, XOR’s the associated hard-core predicate $b(z_j)$ with each of her secrets $\sigma_j$, and sends the encrypted secrets $(c_1, \ldots, c_k)$ to Bob. Because $y_i$ is the encryption of $x_i$, then $z_i = x_i$, so Bob is able to compute $b(z_i)$ and decrypt $c_i$ to obtain $\sigma_i$. He doesn’t learn the other secrets since for $j \neq i$, he knows only $x_j = f_\alpha(z_i)$. He cannot predict $b(z_i)$ given $x_j$ with more than an negligible advantage since $b$ is hard-core for $f_\alpha$. 
Common input: Security parameter $1^n$.

**Alice**

Private input: $(\sigma_1, \ldots, \sigma_k)$.

1. Choose random $r$.  
   Compute $(\alpha, t) = G(1^n, r)$.

2. 

3. For $j \in \{1, \ldots, k\}$:
   \begin{align*}
   z_j &= f^{-1}_\alpha(y_j). \\
   c_j &= \sigma_j \oplus b(z_j).
   \end{align*}
   (Note: $z_i = x_i$ and $c_i = \sigma_i \oplus b(x_i)$.)

4. 

**Bob**

Private input: $i$.

Choose random $x_1, \ldots, x_k \in D_\alpha$.

Let $y_j = \left\{ \begin{array}{ll} f_\alpha(x_j) & \text{if } j = i \\ x_j & \text{if } j \neq i. \end{array} \right.$

$(y_1, \ldots, y_k) \leftarrow (c_1, \ldots, c_k)$

Output $c_i \oplus b(x_i)$.

Figure 57.1: An OT$_1^k$ oblivious transfer protocol.

**Theorem 1** Suppose $\{f_\alpha : D_\alpha \to D_\alpha\}_{\alpha \in I}$ is a collection of enhanced trapdoor permutations and $b$ is a hard-core predicate for them. Then the protocol of Figure 57.1 privately computes OT$_1^k$ in the semi-honest model.