Lecture 29: 2-round Key Agreement and Public-key Encryption
Suppose there is a 2-round Key-Agreement protocol. This means that there exists a protocol where

- Bob sends the first message $m_B$
- Alice sends the second message $m_A$
- Now, parties can compute a secret key $key$ that is hidden from an eavesdropper (who got to see the first message by Bob and the second message by Alice)
- For example, Diffie-Hellman key-exchange protocol. Bob sends $m_B = g^b$, Alice sends $m_A = g^a$, and both parties compute the key $key = g^{ab}$, but it remains hidden from the adversary.

Using this 2-round key-agreement protocol we can construct a public-key encryption scheme. For example, using the Diffie-Hellman key-exchange protocol, we shall construct the ElGamal public-key encryption scheme.
Suppose we have a protocol $\Pi_{2-KA}$, which is a 2-round key-agreement protocol that looks like the following:

- Alice:
  - Computes key $k$

- Bob:
  - Sends $m_B$
  - Receives $m_A$

Note that $\Pi_{2-KA}$ can be any 2-round key-agreement protocol. One such example is the Diffie-Hellman key-agreement protocol. The next slide presents this protocol in this template.
For example, we consider $\Pi_{2\text{-KA}}$ to be the Diffie-Hellman key agreement protocol.

\[
m_B = g^b \\
m_A = g^a
\]

*Alice*

Compute key $k = m_B^a$

*Bob*

Compute key $k = m_A^b$
Suppose we have a private-key encryption scheme \((\text{Gen}, \text{Enc}, \text{Dec})\). Without loss of generality, we can assume that \(\text{Gen}()\) outputs a uniformly random key \(sk\) from a set \(S\). Recall that a private-key encryption scheme looks as follows:

\[
c = \text{Enc}_{sk}(m)
\]

\[
\tilde{m} = \text{Dec}_{sk}(c)
\]
Consider, for example, the one-time pad encryption scheme

\[ c = m \circ sk \]

\[ \tilde{m} = c \circ \text{inv}(sk) \]
Combining to obtain a Public-key Encryption Scheme I

If the key of the first component is random over the set \( S \) (from which the private-key of the second-component is chosen) then we can stick together these two protocols as follows:

\[
\text{Alice} \\
\quad m_B \\
\quad m_A \\
Compute \quad \text{key } k \\
Set \ sk = k \\
\quad c = \text{Enc}_{sk}(m) \\
\quad \tilde{m} = \text{Dec}_{sk}(c) \\
\text{Bob} \\
\quad \tilde{m} \\
\quad m_A \\
\quad m_B \\
\text{Compute} \quad \text{key } k \\
\text{Set} \ sk = k
\]
We can merge the message $m_A$ and $c$ into one-single message. And we get the following scheme.

\begin{align*}
\text{Alice} & \quad m_B \\
\text{Bob} & \\
\text{Compute key } k & \\
\text{Set } sk = k & \quad m_A, c = \text{Enc}_{sk}(m) \\
\text{Compute key } k & \\
\text{Set } sk = k & \quad \tilde{m} = \text{Dec}_{sk}(c)
\end{align*}
Every time we want to encrypt a message $m$, we calculate a fresh key $k$. And we get the following scheme.

Alice

Compute key $k$
Set $sk = k$

$\text{Compute } k$
Set $sk = k$

Bob

$m_A$, $c = \text{Enc}_{sk}(m)$

$\tilde{m} = \text{Dec}_{sk}(c)$
Finally, we interpret the message $m_B$ as the public-key for Bob. And the messages $(m_A, c)$ as the encryption of the message $m$. This gives us our public-key encryption scheme!

\begin{align*}
\text{Alice} & \quad \text{pk} = m_B \\
\text{Bob} & \quad \text{Compute key } k \\
& \quad \text{Set } sk = k \\
& \quad c = (m_A, c'), \quad \text{where } c' = \text{Enc}_{sk}(m) \\
& \quad \tilde{m} = \text{Dec}_{sk}(c)
\end{align*}
Suppose our first component is Diffie-Hellman key-agreement protocol and the second component is one-time pad. Then we get the following public-key encryption scheme.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>( pk = g^b )</td>
<td></td>
</tr>
</tbody>
</table>

- Compute
- key \( k = g^{ab} \)
- Set \( sk = k \)

\[
c = (m_A = g^a, c' = m \cdot g^{ab})
\]

- Compute
- key \( k = g^{ab} \)
- Set \( sk = k \)

\[
\tilde{m} = c' \cdot \text{inv}(g^{ab})
\]
This is the ElGamal public-key encryption scheme!