Lecturer: Mark Zhandry Scribe: Divyarthi Mohan and Fermi Ma

Notes for Lecture 8

Note: These lecture notes cover lecture 8 and the beginning of lecture 9, where we finished the Order Revealing Encryption security proof.

1 Last Time

Last time, we saw that iO and lossy encryption implies Fully Homomorphic Encryption (FHE). We had to use a tool called complexity leveraging. Essentially, the reduction was so bad that we had to assume our usual primitives were subexponentially secure. Today, we'll see another setting where complexity leveraging is helpful / necessary.

2 Order Revealing Encryption

We will see a new kind of encryption called Order Revealing Encryption (ORE) that allows us to publicly learn the ordering of information but nothing else. We will then see how to construct an ORE scheme from iO and one-way functions (OWF).

2.1 Motivation

Suppose we have a database of, say, medical records that are ordered according to the patients' blood pressure. We might want to outsource this to a cloud provider, but we don't want the cloud to learn private patient information. Suppose we now want to retrieve information from the cloud, such as ciphertexts corresponding to all patients whose blood pressure is higher than some threshold. We ideally want the cloud to do a binary search, which requires the cloud to be able to perform comparison computations on ciphertexts. Note that FHE isn't good enough to solve this problem, since FHE schemes use circuit computations and thus a binary search requires linear size.

In this setting, the cloud will learn information about how the ciphertexts are ordered. However, we don't want to give away anything beyond the ordering.

2.2 Formalism

An ORE scheme parameterized by N consists of the four algorithms Gen, Enc, Dec and Comp defined as follows,

- $\mathsf{Gen}()$ outputs a secret key, public key pair sk, pk.
- $\mathsf{Enc}(\mathsf{sk}, m)$ takes as input a secret key sk and a message $m \in [0, N]$, and outputs cyphertext c, the encryption of m under sk.
- $\mathsf{Dec}(\mathsf{sk}, c)$ takes as input a secret key sk and a cyphertext c, and outputs m, the decryption of c.
- $\mathsf{Comp}(\mathsf{pk}, c_0, c_1)$ takes as input a public key pk and cyphertexts c_0, c_1. If $c_0 < c_1$, it outputs $<$, and otherwise it outputs $\geq$.

If we give the cloud the encrypted data and pk, it can perform a binary search using Comp.

2.3 Correctness

Let $\mathsf{sk}, \mathsf{pk} \leftarrow \mathsf{Gen}()$.

For all $m \in [0, N]$, we require $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{sk}, m)) = m$.

For all $m_0 < m_1$, $\mathsf{Comp}(\mathsf{pk}, \mathsf{Enc}(\mathsf{sk}, m_0), \mathsf{Enc}(\mathsf{sk}, m_1))$ must output $<$

2.4 Security

We need to use secret key encryption because otherwise an adversary can encrypt and use a binary search to decrypt any cyphertext. The encryption key and the compare key should not be available to an adversary together.

For the same reason, we cannot achieve CPA-security. Recall that for CPA-security, the adversary A gets the public key pk from the challenger C and then does polynomially many encryption queries (that is, A sends m_i and gets the ciphertext $c_i = \mathsf{Enc}(\mathsf{sk}, m_i)$). Then A sends 2 distinct plaintexts m_0^*, m_1^* different from all the m_i's and C sends back c^*, an encryption of m_b^* for some $b = 0, 1$. A can then do more encryption queries (not on m_0* or m_1^*) and outputs a guess b' for b. We want $\Pr(b' = b) < 1/2 + \mathsf{negl}$.

This is NOT attainable because having oracle access to encryption allows us to decrypt. For example if A queries on an m between m_0^* and m_1^* and gets the ciphertext $c = \mathsf{Enc}(\mathsf{sk}, m)$, then running $\mathsf{Comp}(\mathsf{pk}, c, c^*)$ reveals b'.

So we will require that $[m_0^*, m_1^*] \cap \{m_i\}_i = \emptyset$. This is the 'best possible' security that we can get.

We also note that we can assume Enc is deterministic because we can tell if we have encrypted the same text twice by running Comp on the ciphertexts.

It is possible to get achieve adaptive security, but today we just consider the weaker notion of static security.

The security experiment is as follows. The adversary A commits to messages m_0^*, m_1^* as well as query messages $m_1, m_2, \ldots, m_t$ such that $m_i \notin [m_0^*, m_1^*]$, and sends these to the challenger. The challenger responds by sending $\mathsf{pk}, c^* = \mathsf{Enc}(sk, m_b^*), c_i = \mathsf{Enc}(sk, m_i)\ \forall i \in [t]$. Then A outputs a guess b' for b.

The security requirement is that for all PPT A,

$$\Pr[b' = b] \leq 1/2 + negl.$$

3 Construction of ORE using iO

There are some difficulties if we construct ORE using the straightforward encryption approach. Suppose we have

- $\mathsf{Gen}()$. Define $P_k((r_0, c_0), (r_1, c_1))$ as follows:

$$\begin{aligned} &\text{for } b = 0, 1 \\ &\qquad m_b \leftarrow c_b \oplus \mathsf{PRF}(k, r_b) \\ &\text{if } m_0 < m_1, \text{ output } < \\ &\text{if } m_0 \geq m_1, \text{ output } \geq \end{aligned}$$

  Output $(\mathsf{sk}, \mathsf{pk}) = (k, \mathsf{iO}(P_k))$

- $\mathsf{Enc}(\mathsf{sk}, m)$. Choose a random $r \leftarrow \{0,1\}^\lambda$, and output $(r, \mathsf{PRF}(\mathsf{sk}, r \oplus m))$.
- $\mathsf{Dec}(\mathsf{sk}, (r, c,))$. Output $c + \mathsf{PRF}_1(k_1, r)$.
- $\mathsf{Comp}(\mathsf{pk}, (r_0, c_0), (r_1, c_1))$. P_k is given in pk. Output $P_k((r_0, c_0), (r_1, c_1))$

Claim: The above scheme is not secure, even if we replace iO with VBBO.

(For simplicity let us assume instead of XOR ($\oplus$) we are doing + mod N)

Consider the following attack. The adversary A chooses some m_0^* and $m_1^* = m_0^* + 2$. Also, query on $m = m_1^* + 1$. The challenger sends back $(r^*, c^*) = \mathsf{Enc}(sk, m_b^*)$ and $(r, c) = \mathsf{Enc}(sk, m)$. We want to set c' to be an encryption of $m_b + 2$, so we set $c' = \mathsf{PRF}(k, r^*) + (m_b^* + 2)$. Thus $(r^*, c') \approx \mathsf{Enc}(k, m_b^* + 2)$. By running $\hat{P}((r^*, c'), (r, c))$

we can easily distinguish between $\mathsf{Enc}(m_0^*)$ and $\mathsf{Enc}(m_1^*)$. If $b = 0$ then $\hat{P}$ outputs $<$, else if $b = 1$ then $\hat{P}$ outputs $\geq$. Thus A correctly guess b.

This attack was possible because we could change the ciphertext such that it corresponded to an encryption of a changed plaintext. We call such ciphertexts malleable. We want an encryption that is non-malleable; tweaking the ciphertext should not give an encryption of another plaintext.

The non-malleable scheme uses three pseudorandom functions $\mathsf{PRF}_1, \mathsf{PRF}_2, \mathsf{PRF}_3$ and is as follows.

- $\mathsf{Gen}()$. Define $P_{k_1,k_2,k_3}((r_0, c_0, d_0), (r_1, c_1, d_1))$ as follows:

$$\begin{aligned}
&\text{for } b = 0, 1 \\
&\qquad m_b \leftarrow c_b - \mathsf{PRF}_1(k_1, r_b) \\
&\qquad \text{if } r_b \neq \mathsf{PRF}_3(k_3, m_b) \text{ then ABORT} \\
&\qquad \text{if } \mathsf{PRG}(d_b) \neq \mathsf{PRG}(\mathsf{PRF}_2(k_2, (r_b, m_b))) \text{ then ABORT} \\
&\text{if } m_0 < m_1, \ \text{output } \ < \\
&\text{if } m_0 \geq m_1, \ \text{output } \ \geq
\end{aligned}$$

  Output $(\mathsf{sk}, \mathsf{pk}) = ((k_1, k_2, k_3), \mathsf{iO}(P_k))$

- $\mathsf{Enc}(\mathsf{sk}, m)$. We have $\mathsf{sk} = (k_1, k_2, k_3)$. Set $r = \mathsf{PRF}_3(k_3, m)$; output $(r, \mathsf{PRF}_1(k_1, r) + m, \mathsf{PRF}_2(k_2, (r, m)))$

- $\mathsf{Dec}(\mathsf{sk}, (r, c, d))$. We have $\mathsf{sk} = (k_1, k_2, k_3)$. Output $c - \mathsf{PRF}_1(k_1, r)$.

- $\mathsf{Comp}(\mathsf{pk}, (r_0, c_0, d_0), (r_1, c_1, d_1))$. P_{k_1,k_2,k_3} is given in pk. Output $P_{k_1,k_2,k_3}((r_0, c_0, d_0), (r_1, c_1, d_1))$

Theorem 1. *(informal) The above construction is secure if we assume subexponentially secure* iO*.*

Proof. We prove security for the case where $m_1^* = m_0^* + 1$. We simply repeat this argument to extend it to arbitrary m_0^*, m_1^*, but note that this requires subexponentially secure iO.

As usual, the proof will be done via a sequence of hybrids.

Hybrid 0. This is the real experiment with $b = 0$.

Hybrid 1. Let $h^* = \mathsf{PRG}(\mathsf{PRF}_2(k_2, (r_1^*, m_1^*)))$. Define P' as follows.

$P'_{h^*,k_2\{r_1^*,m_1^*\}}((r_0,c_0,d_0),(r_1,c_1,d_1))$:

$$
\begin{array}{l}
\text{for } b = 0,1 \\
\quad m_b \leftarrow c_b - h_b \\
\quad \text{if } \mathsf{PRF}_3(k_3, m_b) \neq r_b \text{ then ABORT} \\
\quad \text{if } (r_b, m_b) = (r_1^*, m_1^*) \text{ then } h_b = h^* \\
\quad \text{else } h_b = \mathsf{PRG}(\mathsf{PRF}_2(k_2\{r_1^*, m_1^*\}, (r_b, m_b))) \\
\quad \text{if } h_b \neq \mathsf{PRG}(d_b) \text{ then ABORT} \\
\text{if } m_0 < m_1, \text{ output } < \\
\text{if } m_0 \geq m_1, \text{ output } \geq
\end{array}
$$

This hybrid will be the same as Hybrid 0, except $\mathsf{pk} = \mathsf{iO}(P')$.

Since $P' \equiv P$, by iO we have Hybrid 0 $\approx_c$ Hybrid1.

Hybrid 2. This hybrid will be the same as Hybrid 1, except we set $h^* = \mathsf{PRG}(s)$ where s is truly random.

By punctured PRF security we have Hybrid 1 $\approx_c$ Hybrid 2.

Hybrid 3. This hybrid will be the same as Hybrid 2, except h^* is truly random.

By PRG security we have Hybrid 2 $\approx_c$ Hybrid 3.

So now we will never (as in, with only negl probabilty) decrypt to m_1^*.

Hybrid 4. Define P'' as follows.

$P''_{h^*,k_2\{r_1^*,m_1^*\}}((r_0,c_0,d_0),(r_1,c_1,d_1))$:

$$
\begin{array}{l}
\text{for } b = 0,1 \\
\quad m_b \leftarrow c_b - h_b \\
\quad \text{if } \mathsf{PRF}_3(k_3, m_b) \neq r_b \text{ then ABORT} \\
\quad \text{if } (r_b, m_b) = (r_1^*, m_1^*) \text{ then } h_b = h^* \\
\quad \text{else } h_b = \mathsf{PRG}(\mathsf{PRF}_2(k_2\{r_1^*, m_1^*\}, (r_b, m_b))) \\
\quad \text{if } h_b \neq \mathsf{PRG}(d_b) \text{ then ABORT} \\
\text{if } \{m_0, m_1\} = \{m_0^*, m_1^*\} \text{ then ABORT} \\
\text{if } m_0 < m_1, \text{ output } < \\
\text{if } m_0 \geq m_1, \text{ output } \geq
\end{array}
$$

This hybrid will be the same as Hybrid 3, except we set $\mathsf{pk} = \mathsf{iO}(P'')$ with h^* still random.

Hybrid 5. We revert back to $h^* = \mathsf{PRG}(s)$

Hybrid 6. We revert back to $h^* = \mathsf{PRG}(\mathsf{PRF}_2(k_2, r_1^*, m_1^*))$

Hybrid 7. We set $\mathsf{pk} = \mathsf{iO}(P''')$, where P''' is a program identical to P but aborts if $\{m_0, m_1\} = \{m_0^*, m_1^*\}$.

(At this point, we transitioned to lecture 9.)

To finish the proof, we need to puncture the PRFs so that encryptions of m_0^* and m_1^* are truly random strings.

We need PRF_3 to be injective. It turns out that we can build PRFs that are injective and puncturable. We need m_0^* and m_1^* to map to unique r's and also be different from all other r's.

We puncture PRF_3 at m_0^*, m_1^*. In doing so, we can replace r_0^*, r_1^* (the random outputs corresponding to m_0^*, m_1^*) with random strings.

Similarly, we puncture PRF_1 at r_0^*, r_1^*. We can replace c_0^*, c_1^* (the corresponding outputs) with random stringsas well.

Finally, we puncture PRF_2 at (r_0^*, m_0^*) and (r_1^*, m_1^*) and replace d_0^*, d_1^* with random strings. Note that this requires PRF_2 to be puncturable at two spots, but it turns out that this is very easily achievable.

After doing this, we get to the following program (we won't write out all the subscripts but they are there)

$P'''((r_0, c_0, d_0), (r_1, c_1, d_1))$:

$$
\begin{aligned}
&\text{for } b = 0, 1 \\
&\qquad \text{if } (r_b, c_b, d_b) = (r_e^*, c_e^*, d_e^*), \text{ set } m_b \leftarrow m^* + e \\
&\qquad \text{else if } r_b \in \{r_0^*, r_1^*\}, \text{ then ABORT} \\
&\qquad \text{else } m_b \leftarrow c_b \oplus PRF_1(k_1\{r_0^*, r_1^*\}, r_b) \\
&\qquad \text{if } m_b \in \{m_0^*, m_1^*\} \text{ then ABORT} \\
&\qquad \text{if } r_b \neq \mathsf{PRF}_3(k_3\{m_0^*, m_1^*\}, m_b) \text{ then ABORT} \\
&\qquad \text{if } \mathsf{PRG}(d_b) \neq \mathsf{PRG}(PRF_2(k_2\{,\}, (r_b, m_b))) \text{ then ABORT} \\
&\text{if } m_0 < m_1, \text{ output } < \\
&\text{if } m_0 \geq m_1, \text{ output } \geq
\end{aligned}
$$

So r_0^*, c_0^*, d_0^* and r_1^*, c_1^*, d_1^* are hardcoded into the program, but the behavior is the exactly the same on both inputs.

What we've shown is that an encryption of m_0^* is computationally indistinguishable from $m_0^* + 1$, and $m_0^* + 1$ is computationally indistinguishable from $m^* + 2$, and so in general m^* is indistinguishable from m'. Potentially there is an exponential distance between the messages, so to actually get order revealing encryption, we need to assume subexponential hardness of all the primitives.

$\square$

4 Summary

In ORE we can publicly learn some information but not everything. This is a special case of a more general idea called *Functional Encryption (FE)*.

FE consists of four algorithms Gen,Enc,Dec,KeyGen defined as follows.

- $\mathsf{Gen}()$ gives a secret key msk.
- $\mathsf{Enc}(\mathsf{msk}, m)$ outputs the cipher text c, which is the encrption of message m.
- $\mathsf{Dec}(\mathsf{msk}, c)$ outputs a plaintext m, which is the decryption of c.
- $\mathsf{KeyGen}(\mathsf{msk}, f)$ outputs a key sk_f given a function f.
- $\mathsf{Dec}(\mathsf{sk}_f, c)$ outputs $f(m)$ where $m = \mathsf{Dec}(\mathsf{msk}, c)$.

We can also consider the case where f is multivariable function. In case of a ORE f is a two variable function and sk_f is Comp.

We can also define a corresponding public key version of FE. The 'right notion' of FE gives iO. We can also construct FE using iO. In fact, with some effort it is even possible to construct FE from iOwithout the exponential loss, meaning we do not need to rely on sub-exponentially secure iO. The result also gives ORE without the exponential loss.