Notes for Lecture 6

1 Last Time

Last time we saw how to use obfuscation and one way functions to build multiparty NIKE. Today, we'll see that multiparty NIKE implies broadcast encryption.

2 Broadcast Encryption Definition

Suppose we have a content distributor (like Netflix) and a number of users. Some users are are registered to receive a particular broadcast, and the content distributor wants to make one big broadcast so that only the registered users can decrypt the broadcast.

A broadcast encryption scheme consists of the following PPT algorithms:

- $\mathsf{Gen}(N, \lambda) \to (mpk, msk)$
- $\mathsf{Extract}(msk, i) \to sk_i$
- $\mathsf{Enc}(mpk, S, m) \to c$
- $\mathsf{Dec}(mpk, sk_i, S, c) \to m$

For correctness, if $i \in S$, then $\mathsf{Dec}(mpk, sk_i, S, \mathsf{Enc}(mpk, S, m)) = m$.

The most basic notion of security is that users outside of S cannot learn anything about the message. It turns out that this definition of security isn't good enough, since it leaves open the possibility of multiple excluded users colluding to decrypt the broadcast.

To deal with this issue, we define fully collusion resistant security, where the assumption is that every user outside the broadcast will collude.

The experiment works as follows. The adversary picks the broadcast set $S^* \subseteq [N]$. The challenger then generates $(mpk, msk) \leftarrow \mathsf{Gen}(N, \lambda)$ and sends mpk to the adversary. For all $i \notin S^*$, the challenger runs $sk_i \leftarrow \mathsf{Extract}(msk, i)$ and sends $\{sk_i\}_{i \notin S^*}$ back to the adversary. Then the adversary sends back messages m_0, m_1 and receives the cyphertext $c^* = \mathsf{Enc}(mpk, S, m_b)$ and tries to guess b'.

This is the static security version. We can also define an adaptive security version where the adversary can perform secret key queries before picking the set S^*, but we won't worry about that in this class.

3 Broadcast Encryption from PKE

It turns out that public key encryption is sufficient to build a simple statically secure broadcast encryption scheme:

- $\mathsf{Gen}(N, \lambda)$. For all $i \in [N]$, set $(pk_i, sk_i) \leftarrow \mathsf{Gen}_{PKE}(\lambda)$, and output $(mpk, msk) = (\{pk_i\}, \{sk_i\})$.
- $\mathsf{Extract}(msk, i)$. Output sk_i
- $\mathsf{Enc}(mpk, S, m)$. For all $i \in [S]$, compute $c_i = \mathsf{Enc}(pk_i, m)$, and then output $c = \{c_i\}_{i \in S}$

The problem with this scheme is that the broadcaster is essentially sending a separate ciphertext for each person in the broadcast, so the broadcast size is huge. Clearly, this is undesirable for settings where the number of users is very large.

4 Broadcast Encryption from Multiparty NIKE

What we want is for the cyphertext to be about the same size as the message, regardless of the number of users. Ideally, $|mpk|, |msk|, |sk_i|, |c|$ will all be polynomial in the security parameter λ, and have no dependence on N.

The scheme we'll give today won't quite get to these conditions. Instead, we'll achieve $|sk|, |c|$ polynomial in λ and $|mpk|, |msk|$ polynomial in N, λ. It is possible, however, to achieve the ideal parameters but we won't talk about that in this lecture.

The rough idea for this scheme will be that the users in the broadcast set S run a multiparty NIKE protocol to decrypt the message.

- $\mathsf{Gen}(N, \lambda)$ works by outputting $(sv_i, pv_i) \leftarrow \mathsf{Publish}(\lambda)$ for $i = 1, \ldots, N$. $msk = \{sv_i\}, mpk = \{pv_i\}$.
- $\mathsf{Extract}(msk, i) = sv_i$.
- $\mathsf{Enc}(msk, S, m)$ first computes $k \leftarrow \mathsf{KeyGen}(\{pv_i\}_{i \in S}, sv_j)$ and then outputs $c = k \oplus m$.
- $\mathsf{Dec}(mpk, sv_i, S, c)$ computes $k \leftarrow \mathsf{KeyGen}(\{pv_j\}_{j \in S}, sv_i)$ and $m = k \oplus c$.

(Aside: The reason why we include S in the decryption input is that S takes N bits, so if we don't make the input size depend on the size of S, it is not possible to get ideal parameter size for the cyphertext. It's possible to make an argument that no matter how you try to hide S, the cyphertext will still have to encode S somehow.)

Whoever is going to encrypt needs to be able to compute the shared key k. So we'll modify Enc and Dec as follows:

- $\mathsf{Enc}(msk, S, m)$ first generates $(pv_0, sv_0) \leftarrow \mathsf{Publish}(\lambda)$ and then computes $k \leftarrow \mathsf{KeyGen}(\{pv_i\})i \in S \cup \{0\}, sv_0)$, and then outputs $c = k \oplus m, pv_0$.
- $\mathsf{Dec}(mpk, sv_i, S, (c, pv_0))$ computes $k \leftarrow \mathsf{KeyGen}(\{pv_i\})i \in S \cup \{0\}, sv_i)$ and outputs $m = k \oplus c$.

Theorem 1. *The above broadcast encryption scheme is statically secure*

Proof. Let A be an adversary for the broadcast encryption scheme. We'll use A to produce an adversary B that breaks the security of multiparty NIKE.

A sends $S^* \subseteq [N]$ to B. B defines $l = |S^*|$. B receives from his challenger $\{pv_i\}_{i=0,1,\ldots,l}, K$, where K is either the shared key for these users or is uniformly randomly generated. He keeps the first public value pv_0 for himself. Let $S^* = \{i_1, \ldots, i_l\}$. B assigns $\overline{pv_{i_j}} = pv_j$. For users outside of S^*, B generates their public and secret values. He then sends $mpk = \{\overline{pv}_i\}$ to A. He also generates secret values for users outside of S^*, so he sends $\{sv_i\}_{i \notin S^*}$ to A as well. Then A sends back m_0, m_1 and B randomly chooses $b \leftarrow \{0,1\}$ and then sends back $k \oplus m_b, pv_0$. Then A sends back b' and B outputs $b \oplus b'$.

If k is the actual shared key, then the view of A is exactly what A would expect, where the cyphertext does actually encrypt m_b. The probability that B outputs 0 is $\frac{1}{2} + \epsilon$ where ϵ is A's advantage.

If k is random, then the probability that B outputs 0 is identically 1/2. By the security of multiparty NIKE, these two probabilities need to be negligibly close, so A's advantage ϵ is negligible. □

5 Broadcast Encryption from iO / diO

Here we sketch what happens if we instantiate the protocol using the NIKE protocol from last lecture, obtaining a broadcast scheme from indistinguishability obfuscation. We just write out Gen and $\mathsf{Extract}$:

- $\mathsf{Gen}(N, \lambda)$ outputs $s_i \leftarrow \{0,1\}^\lambda, x_i \leftarrow PRG(s_i)$ and $\hat{P} = iO(P)$, where $P(y_1, \ldots, y_{N+1}, i, s_i)$ works as follows. It first checks $PRG(s_i) = y_i$, and if not, it aborts. If the check passes, it outputs $PRF(k, (y_1, \ldots, y_N))$. So $mpk = (\{x_i\}, \hat{P})$ and $msk = \{s_i\}$.

- $\mathsf{Extract}(msk, i) = s_i$.

Now, this protocol has large public keys. The issue is two-fold: first, the set of $\{x_i\}$ necessarily grows with the number of users. Second, $\hat{P}$ takes as input a sequence of values which is potentially as large as the number of users.

Here, we will discuss one approach to the issue of large $\hat{P}$. We can think of the obfuscated program as acting on three inputs: v_S, i, π. Here, v_S is some value dependent on S. i is the user number, and π is a "proof" that user i has "access" to v_S. We want to set things up so that only users $i \in S$ can construct valid proofs. The program checks that π is a valid proof that i has "access" to v_S. If not, it aborts, but otherwise it outputs $PRF_k(v_S)$. We want to shrink v_S so it doesn't grow with the number of users. Namely, $|v_S| << |S|$.

We can set $v_S = H(S)$ where H is a hash function. Interpret S as being a bit string where each bit says whether that user is in S.

Since H is compressing (its output is much smaller than its input), there are guaranteed to be many collisions. Consider two S, S' that hash to the same value: $v_S = H(S) = H(S') = v_{S'}$. Suppose I broadcast to the set S. Let $i \in S' \setminus S$. We need that i cannot produce a valid proof that i has access to v_S. But at the same time, we need that i *can* produce a valid proof that i has access to $v_{S'}$. But since $v_S = v_{S'}$, we apparently have a contradiction.

Fortunately, sets S, S' that hash to the same value are hard to find if we assume H is collision resistant. This means that user i should not be able to compute the set S' that collides with S, and therefore should not be able to compute a valid proof that it has access to $v_{S'}$. However, the collision resistance of H is insufficient for actually proving security with iO. The reason is that, while $i \notin S$ should not be able to compute a valid proof, these valid proofs are guaranteed to exist. They are just hard to find. Yet for iO, we need that valid proofs for users outside of S do not exist at all.

This motivates the definition of Differing Inputs Obfuscation (diO), which is stronger than indistinguishability obfuscation. For iO, if two equal-length programs are the same on all inputs, then their obfuscations are indistinguishable. In diO, we say that if two equal-length programs are such that we cannot find inputs where the programs differ, then the obfuscations are indistinguishable.

Consider $(C_0, C_1, aux) \leftarrow \mathsf{Samp}()$. We say that $\mathsf{Samp}()$ is a differing inputs sampler if $\forall$ PPT algorithms A,

$$Pr[A(C_0, C_1, aux) = x \text{ s.t. } C_0(x) \neq C_1(x) \text{ where } (C_0, C_1, aux) \leftarrow \mathsf{Samp}()] < \mathsf{negl}.$$

In other words, a differing inputs sampler gives a distribution on pairs of circuits C_0, C_1 and auxiliary information such that given this input, no PPT algorithm can find a point where the circuits differ (with non-negligible probability).

diO is a differing inputs obfuscator if for all differing inputs samplers Samp, and for all adversaries B,

$$|Pr[B(diO(C_0), C_0, C_1, aux) = 1] - Pr[B(diO(C_1), C_0, C_1, aux) = 1]| < negl$$

Where $(C_0, C_1, aux) \leftarrow \mathsf{Samp}$.

We do not have a proof one way or another that diO exists. However, there are some arguments that diO is unlikely to exist, at least in its most general form as outlined above. However, diO is known to exist for very restricted families of differing inputs samplers.

Even assuming diO, shrinking the public key size as above is non-trivial. The reason is that coming up with a proof π that is short is problematic. Indeed, the simple proof would be to just supply the set S, but this defeats the whole purpose. Instead, the proof must be much shorter than $|S|$. Surprisingly, using a special type of hash function called Merkle Trees, this is possible. Merkle trees can be built from any collision resistant hash function.

More recent constructions of broadcast encryption have shown how to remove diO, so we only need iO, as well as removing the need for collision resistance. That is, now we have broadcast encryption where all parameters are independent of N form just iO and one-way functions.