COS 597C: Recent Developments in Program Obfuscation — Princeton University
Lecture 5 (9/29/16)
Lecturer: Mark Zhandry — Scribe: Fermi Ma

Notes for Lecture 5

1 Last Time

Last time, we saw that we can get public key encryption (PKE) from indistinguishability obfuscation (iO) and one way functions (OWF). At the end, we looked at multiparty non-interactive key exchange (NIKE). Today, we'll continue looking at multiparty NIKE.

2 Non-interactive vs. Interactive Key Exchange

We first consider the two person case. In this setting, Alice and Bob have a channel through which they can communicate over many rounds. The goal is for Alice and Bob to agree on a shared key k that an eavesdropper (who can see every round of communication) is unable to deduce anything about.

We can solve this using public key encryption, as follows. Alice generates $(ek, dk) \leftarrow \mathsf{Gen}()$ and then sends ek to Bob. Bob picks a random key $k \leftarrow \{0,1\}^\lambda$ and sends back $c \leftarrow \mathsf{Enc}(ek, k)$. So Bob has k and Alice can learn k by just decrypting $k = \mathsf{Dec}(dk, c)$.

In 2-party NIKE, Alice and Bob both post to a public bulletin board at the same time and establish a shared key k. It's non-interactive because they don't wait to see each other's messages.

Why is *non-interactive* key exchange particularly interesting?

1. It is interesting to see if we can compress communication to just a single, non-interactive round.

2. The messages/public values can be reused. The fact that this procedure is non-interactive means that Alice and Bob can post their public values to establish a shared key, and later, if Charlie wants to establish a shared key with just Bob, Bob doesn't have to post any new messages. Charlie just runs his half of the protocol, and posts his public value. Then Bob can compute the shared key k_{BC} right away.

Re-use of messages means we can use less communication overall. Suppose we have N users and each pair of users wants to establish a shared key. This just takes N

messages in the non-interactive setting, but requires $\Omega(N^2)$ messages in the interactive setting.

3 History

Before we move onto a construction of multiparty NIKE, we'll briefly survey the history of this problem.

In 1976, Diffie and Hellman came up with a 2-party NIKE scheme using a computational problem involving finite fields [DH76].

In the 1990's, people realized this could be extended to elliptic curves, which gives certain advantages that we won't discuss in this class.

In the 1990's and 2000's, people came up with an extension to lattices.

In 2004, Joux came up with 3 party NIKE using pairings on elliptic curves [Jou04].

In 2012, Garg, Gentry, and Helevi came up with an N-party scheme using multilinear maps [GGH13]. It turns out this scheme is broken.

4 Multiparty NIKE from iO

We define a multiparty NIKE scheme to consist of the following PPT algorithms

- $\mathsf{Setup}(\lambda, N) \to \mathsf{params}$
- $\mathsf{Publish}(\mathsf{params}, i) \to (sv_i, pv_i)$
- $\mathsf{KeyGen}(\mathsf{params}, \{pv_i\}_{i=1,\ldots,N}, sv_j) \to k$

Here we're allowing a trusted setup to occur first. Think of this as Facebook publishing parameters to a public bulletin board, and then the Facebook users running a multiparty NIKE scheme afterwards. It's still non-interactive, but there is an additional step in the beginning.

Correctness is straightforward; for all j, everyone gets the same key.

The scheme is secure if no polynomial time adversary can distinguish between the following two cases.

- Case 1. The adversary receives k, params, $\{pv_i\}_{i=1,\ldots,N}$ where $\mathsf{params} \leftarrow \mathsf{Setup}(\lambda, N)$, $\forall i\ (pv_i, sv_i) \leftarrow \mathsf{Publish}(\mathsf{params}, i)$, and key $k \leftarrow \mathsf{KeyGen}(\mathsf{params}, \{pv_i\}_{i=1,\ldots,N}, sv_1)$.

- Case 2. The adversary receives $R, \mathsf{params}, \{pv_i\}_{i=1,\ldots,N}$. This is generated the same way as it was in case 1, except R is a totally random string $R \leftarrow \{0,1\}^\lambda$. $\mathsf{params} \leftarrow \mathsf{Gen}(\lambda)$.

We first give a broken protocol to build intuition. We consider the situation where all users have access to some black box F. We claim that for each user, their public value should be the result of a one way function applied to their secret value. If this isn't the case for some user, then the adversary can potentially learn things about this user's secret value.

The (broken) protocol works as follows. Say the users are A, B, C. Everybody posts their public value. F has the secret key k_{PRF} for some PRF in its head. F just returns $PRF(k_{PRF}, (pv_A, pv_B, pv_C))$, and the users agree on this as the shared key.

Obviously this doesn't work since this key relies solely on public information. The adversary could just submit pv_A, pv_B, pv_C to the black box and know the shared key. So to make the scheme work, we modify the black box to only give back the key if the user has supplied their secret value first.

We start by defining $\mathsf{Setup}(\lambda, N)$ to work as follows. First, sample a totally random key $k_{PRF} \leftarrow \{0,1\}^\lambda$. Define the program $P_{k_{PRF},\lambda,N}(pv_1, \ldots, pv_N, i, sv_i)$ to have the following behavior.

```
Check if $pv_i = OWF(sv_i)$.
If not, abort and output $\perp$.
If so, set $k \leftarrow PRF(k_{PRF}, (pv_1, \ldots, pv_N)$ and output $k$.
```

$\mathsf{Setup}(\lambda, N)$ will output params where $\mathsf{params} \leftarrow iO(P_{k_{PRF},\lambda,N})$.

If we had black box obfuscation, this would be good enough. But to make a proof of security work with just iO, we need to make a few changes. We turn the one way function into a PRG, and the PRF into a puncturable PRF. The PRG will be length doubling: $\{0,1\}^\lambda \rightarrow \{0,1\}^{2\lambda}$. So the working protocol we give is:

- $\mathsf{Setup}(\lambda, N)$ samples a totally random $k_{PRF} \leftarrow \{0,1\}^\lambda$, and outputs $iO(P_{k_{PRF},\lambda,N})$, where $P_{k_{PRF},\lambda,N}$ is as defined above (with the given modifications).
- $\mathsf{Publish}(\hat{P})$ sets $sv \leftarrow \{0,1\}^\lambda$, $pv \leftarrow PRG(sv)$, and then outputs pv, sv.
- $\mathsf{KeyGen}(\hat{P}, \{pv_i\}_{i=1,\ldots,N}, i, sv_i)$ outputs $\hat{P}(\{pv_i\}, i, sv_i)$.

Theorem 1. *The above protocol is secure.*

Proof. We use a series of hybrids to prove security. Hybrid 0 will correspond to the first case of the security experiment, and Hybrid 5 will correspond to the second case.

In Hybrid 0, the adversary gets a properly generated key $k \leftarrow PRF(k_{PRF}, \{\overline{pv}_i\})$ and $params, \{\overline{pv}_i\}$. Here $\overline{pv}_i = PRG(\overline{sv}_i)$. The bar in $\overline{pv}, \overline{sv}$ is simply to distinguish these original values.

Hybrid 1 is the same as Hybrid 0, except we change the public values to be truly random, so $\forall i \; pv_i \leftarrow \{0,1\}^{2\lambda}$. Security of the PRG allows us to replace all the public values with truly random values.

Hybrid 2 is the same as Hybrid 1, except now we set $\mathsf{params} = iO(P')$, where $P'_{z,k\{z\}PRF,\lambda,N}(pv_1, \ldots, pv_N, i, sv_i)$ has the following behavior (for shorthand, we are now setting $z = \{\overline{pv}_i\}$).

```
Check that $pv_i = PRG(sv_i)$.
Check that $z \neq \{pv_i\}$.
If either check fails, output $\perp$.
Otherwise, set $k \leftarrow PRF(k_{PRF}\{z\}, pv_1, \ldots, pv_N)$, and output $k$.
```

To get from P to P', we changed the PRF to a punctured PRF, and we added a new line to check if $z \neq \{pv_i\}$.

We claim this hasn't changed the function. For the second check to fail in a case where the first check wouldn't already fail, the input must be $\overline{pv}_1, \ldots, \overline{pv}_N, i, sv_i$ where $PRG(sv_i) = \overline{pv}_i$. This last equality is overwhelmingly unlikely to be satisfied, since $\overline{pv}_i$ is randomly drawn from a set $2^{2\lambda}$ and $\overline{sv}_i$ is drawn from a set of size 2^{λ}.

Since these programs are equivalent w.h.p., iO tells us that this is indistinguishable from Hybrid 1.

Hybrid 3 is the same as Hybrid 2, except we set k to be a randomly drawn value $k \leftarrow \{0,1\}^{\lambda}$. In Hybrid 2, k was the value of a punctured PRF at the punctured point. Now we're changing it to be totally random, which by punctured security is impossible to distinguish. So this is indistinguishable from Hybrid 2.

In each of the remaining hybrids, we undo the changes above by applying the same arguments.

In Hybrid 4, we set $\mathsf{params} = iO(P_{k_{PRF},\lambda,N})$.

In Hybrid 5, we set $\overline{pv}_i = PRG(\overline{sv}_i)$. We are back to where we started, but now there is true randomness in the key. □

We can modify this protocol to get rid of Setup, which is undesirable since the authority can learn k.

What if Alice just generates the parameters? Alice can publish $\hat{P}$ along with pv_A. Alice can do this since none of the users need to see $\hat{P}$ in order to generate their own public values.

This protocol has some weaknesses. Multiparty NIKE was motivated by reusability of messages, but here if David wants to join and compute a shared key with a set of users not including Alice, they can't use the same $\hat{P}$. The proper way to do it is that each person submits their own $\hat{P}$ and the users can use the lexicographically first program.

Another issue is that once Alice publishes her program, it won't allow more than N users.

5 Stronger Security Models for NIKE

The original model we proved security in assumes that the users establish a key in a vacuum. But the whole point of multiparty NIKE is that there is reusability. So the model should take this into consideration.

Let's suppose an adversary joins the protocol. Moreover, let's say that the adversary obfuscates the following program (instead of the intended program)

$$P(pv_1, \ldots, pv_N, i, sv_i) = sv_i.$$

After running the protocol, Bob (for example) will think his secret value is the shared group key. The adversary will get messages from Bob where he encrypts with his secret key, and can potentially learn Bob's secret value. Then the adversary can use this to learn other shared keys in the future.

There are stronger security models that capture this use case. The strongest is adaptive security, but we only do not yet know how to achieve this from just iO and OWF.

References

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov 1976. 2

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. *Candidate Multilinear Maps from Ideal Lattices*, pages 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. 2

[Jou04] Antoine Joux. A one round protocol for tripartite diffie–hellman. *Journal of Cryptology*, 17(4):263–276, 2004. 2