Notes for Lecture 10

Construction of Obfuscators

We will now procede to the second part of the course where we will see tools for building an obfuscator.

1 Introduction to Multilinear Maps

Today we will see what Multilinear Maps (MMaps) are and a direct application of MMaps to get $(k+1)$-party key agreement. We will first see the definition of a Cryptogrphic Groups and how to get 2-party key agreement using them. MMaps facilitates interatction across different crypto groups.

1.1 Cryptographic Group

$(G, +)$ is a Cryptographic Group if the following hold.

- G is a cyclic group (often of prime order).
- Thus we have a generator g of G, which is usually public.
- The group addition $(+)$ is efficient, which inturn allows us to do effcient scalar multiplications (by repeated addition).
- G has some crypto properties, that is, certain computational problems in this group is hard (We will see some examples later).

1.2 2-party key agreement

We can use cryptographic groups to get a protocol for 2-party key agreement. Suppose we have a cryptographic group G with a generator g. Consider the following protocol.

A and B pick some $a, b \in \mathbb{Z}_p$ respectively. They both publish ag and bg. The shared key is $\mathsf{k} = abg$. Now A can get abg from a and bg. Similary B gets abg.

Security requirement: An advesray can't distinguish when its view is (ag, bg, k) or (ag, bg, R), that is, $(g, ag, bg, abg) \approx_c (g, ag, bg, R)$, where $R = cg \in G$ is random. Here we can either have g as a global generator or A and B just pick some random generator. This computational equivalence is called the Decisional Diffie Hellman.

The following are some crypto properties that we usually assume.

- *Decisional Diffie Hellman (DDH)* - $(g, ag, bg, abg) \approx_c (g, ag, bg, cg)$. That is distinguishing between abg and a random cg given (g, ag, bg) is hard.
- *Computation Diffie Hellman* - For all *PPT Adv* we have $\Pr(Adv(g, ag, bg) = abg) < \mathsf{negl}(\lambda)$, where λ is the security parameter (usually $\lambda = \log p$). This weaker than DDH.
- *Discrete log* - For all $\Pr(Adv(g, ag) = a) < \mathsf{negl}$. This is even weaker.

To generalize this protocol for k-party key agreement we will use multilinear maps.

1.3 Definition of MMaps and $k+1$-party key agreement

We will now consider *Symmetric Multilinear Maps*. We have cyclic groups $G_1, \ldots, G_k$ where each G_i is a cyclic group with $+_i$ as the group operator. Let $g_1, \ldots, g_k$ be their generators. We have an MMap $\times : G_i * G_j \to G_{i+j}$ defined by $ag_i \times bg_j = abg_{i+j}$ if $i + j \leq k$.

Let us now consider a 3-party key agreement protocol. We have crypto groups G_1 and G_2 with a multinear map. A, B and C pick $a, b, c \in \mathbb{Z}_p$ respectively and publish ag_1, bg_1 and cg_1. The shared key is $abcg_2$. A can get $abcg_2$ with a, bg_1 and cg_1. Similarly B and C also compute $abcg_2$.

For secuity, we need Bilinear DDH. That is, $(ag_1, bg_1, cg_1, abcg_2) \approx_c (ag_1, bg_1, cg_1, dg_2)$ where $dg_2 \in G_2$ is random. We need g_1 and g_2 to be global generators (they are fixed so that we can use the bilinear operator).

For $k+1$-party key agreement security we need Multilinear DDH. Observe that by the functionality of the group we can't get a Decisional Discrete log. But it doesn't break DDH, CDH or Discrete log. It would need exponential number of operations to get anything with more than negl probabilty.

The idea is that ag is some sort of encoding of a and the MMap is similar to a homomorphic encryption.

1.4 Notation

The different cyclic groups corresponds to encodings at different level. The group operation gives a homomorphic operator within a level and the MMap gives a homo-

morphic operator across levels.

- $[a]_i = ag_i$ is a level i encoding of a.
- $[a]_0 = a$ is the level 0 encoding of a.
- $[a]_i + [b]_i = [a + b]_i$ for any level i.
- $[a]_i \times [b]_j = [ab]_{i+j}$ for levels i and j such that $i + j \leq k$.

We can formulate all the assumptions in this notation.

2 Graded Encodings

We will now consider a relaxation of MMaps since we don't know how to obtain MMaps for $k > 2$. Graded encodings are like MMaps but the encodings are not unique. We will assume some trusted setup.

2.1 Formalization

Symmetric Graded Encoding consists of algorithms $\mathsf{Setup}()$, $\mathsf{Sample}()$, $+$ and $\times$ defined ad follows.

- $params \leftarrow \mathsf{Setup}(k, \lambda)$. $params$ contains $[1]_1$.
- $[a]_i$ (for a random $a \in \mathbb{Z}_p$) $\leftarrow \mathsf{Sample}(params, i)$. It may not even know what a is even though is gives an encoding of a.
- $[a + b]_i \leftarrow [a]_i + [b]_i$.
- $[ab]_{i+j} \leftarrow [a]_i \times [b]_j$.

So a 3-party key agreement using this looks as follows. The Setup publishes the $params$. A, B and C run $\mathsf{Sample}(params, 0)$ separately and get $[a]_0, [b]_0, [c]_0$ respectively. A can now each sends $[a]_1$ from $[a]_0 \times [1]_1$ (since $[1]_1$ is given by $params$). Similarly B and C send $[b]_1$ and $[c]_1$. The shared key is $[abc]_2$.

This doesn't quite work because we are just getting the level 1 encoding by multiplying $[1]_1$ (which is public) so we might be able to reverse it. That is, if $[a]_1 = [a]_0 \times [1]_1$ then we can get $[a]_0$ by 'dividing' $[a]_1$ by $[1]_1$.

To get around this we need a fresh sample of a at level 1 instead of just using $[a]_0 \times [1]_1$. So we include the algorithm $\mathsf{Rerand}()$. For any encoding of a at level i ($[a]_i$), $\mathsf{Rerand}(params, [a]_i)$ outputs $\mathrm{Hash}([a]_i)$.

This rerandomization is as good as getting a fresh encoding from $\mathsf{Sample}()$. That is, for any random a and any encoding $[a]_i$ $\mathsf{Rerand}(params, [a]_i) \approx_s \mathsf{Sample}(params, i)$.

But this means $[abc]_2$ of A and B may not be same. So we have to extract a key from this encoding. So we include the algorithm $\mathsf{Extract}()$. For any encoding of a in level k, $\mathsf{Extract}(params, [a]_k)$ outputs a key k_a. Note that we can only extract from the top most level.

For security we need Bilinear DDH - $([a]_1, [b]_1, [c]_1, \mathsf{Extract}(params, [abc]_2) \approx_c ([a]_1, [b]_1, [c]_1, R)$, where $[a]_1, [b]_1, [c]_1$ are got from $\mathsf{Sample}()$ and R is random.

2.2 Assymetric Graded Encoding

This is similar to the symmetric graded enconding except how the levels interact with each other. Instead of the levels being integers from 1 to k, we now have subsets of $[k]$ as the levels. So we have 2^k different levels.

The operations are defined as follows,

- $[a + b]_S \leftarrow [a]_S + [b]_S$
- $[ab]_{S \cup T} \leftarrow [a]_S \times [b]_T$, if $S \cap T = \emptyset$
- $\mathsf{Extract}$ at level $[k]$

A 3-party key agreement using assymetric GE is as follows. A gets $[a]_\emptyset$ from $\mathsf{Sample}()$ and sends $[a]_{\{1\}}, [a]_{\{2\}}$. Similary B and C send $[b]_{\{1\}}, [b]_{\{2\}}$ and $[c]_{\{1\}}, [c]_{\{2\}}$.

We can prove that DDH is easy on Bilinear symmetric maps (see homework 2). But DDH might still be hard for assymetric maps. We believe this because the adversary can no longer find an encoding of ab given $[a]_i$ and $[b]_i$ because we can only multiply when the subsets are disjoint.