Degree-2 FE (and More) from Bilinear Maps

1 Bilinear Maps

We start with the notion of groups with bilinear maps. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups of prime order p endowed with a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ which obeys the following properties:

- Non-degeneracy: If $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are non-identity elements, $e(g_1, g_2)$ is a non-identity element in $\mathbb{G}_T$.
- Efficient computability: There is a polynomial-time algorithm to compute e given g_1 and g_2.
- Bilinearity: For all $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2, x_1, x_2 \in \mathbb{Z}_p$,

  $$e(g_1^{x_1}, g_2^{x_2}) = e(g_1, g_2)^{x_1 x_2}$$

  This lets you compute degree-2 polynomials "in the exponent", the source of the power of bilinear maps.

Such groups with a bilinear map can be constructed from elliptic curves. In this case, typically $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of points on an elliptic curve and $\mathbb{G}_T$ is a multiplicative subgroup of $\mathbb{F}_q^*$ for some q. We will not get into the details of how to construct bilinear maps here, but will refer the reader to Boneh and Franklin's 2003 paper for more details [BF03].

Computational Assumptions. We can assume that the **decisional Diffie-Hellman (DDH) problem** is hard in either $\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}_T$. That is,

$$(g, g^x, h, h^x) \approx_c (g, g^x, h, h^y) \tag{1}$$

for uniformly random $x, y \leftarrow \mathbb{Z}_p$ and a uniformly random elements $g, h \leftarrow \mathbb{G}_1$ (or $\mathbb{G}_2$ or $\mathbb{G}_T$, respectively). Equivalently,

$$(g, g^x, g^y, g^{xy}) \approx_c (g, g^x, g^y, g^z)$$

for uniformly random $x, y, z \leftarrow \mathbb{Z}_p$ and a uniformly random element $g \leftarrow \mathbb{G}_1$ (or $\mathbb{G}_2$ or $\mathbb{G}_T$, respectively).

However, one has to be careful: if $g \leftarrow \mathbb{G}_1$ and $h \leftarrow \mathbb{G}_2$ in assumption 1, it is false. Indeed, given (g, g', h, h'), one can compute the bilinear map on two pairs of elements and check for equality:

$$e(g, h') \stackrel{?}{=} e(g', h)$$

which will be true for the distribution on the left hand side in 1, but not for the one on the right hand side.

To remedy this, we look to a generalization of the DDH assumption. To do this, we introduce a piece of notation: for a matrix $\mathbf{A}$ with entries $a_{i,j}$, the notation $g^{\mathbf{A}}$ means a matrix consisting of $g^{a_{i,j}}$ for all i, j. It is not hard to see that the DDH assumption is equivalent to saying that for a random vector $\mathbf{A}, \mathbf{U} \leftarrow \mathbb{Z}_p^{1\times m}$ and $s \leftarrow \mathbb{Z}_p$,

$$(g^{\mathbf{A}}, g^{s\mathbf{A}}) \approx_c (g^{\mathbf{A}}, g^{\mathbf{U}})$$

for a polynomially large m. The **k-linear assumption (k-LIN)** is a generalization where $\mathbf{A}, \mathbf{U} \leftarrow \mathbb{Z}_p^{k\times m}$ and $\mathbf{s} \leftarrow \mathbb{Z}_p^{1\times k}$. That is,

$$(g^{\mathbf{A}}, g^{\mathbf{sA}}) \approx_c (g^{\mathbf{A}}, g^{\mathbf{U}})$$

The attack described above does not apply to the k-linear assumption for $k > 1$. Roughly speaking, the distinguisher above for DDH computed the determinant of a 2×2 submatrix of $[\mathbf{a} \,/\!/\, s\mathbf{a}]$ where $/\!/$ denotes vertical concatenation of matrices. The analog for 2-LIN would require computing the determinant of 3×3 matrices which is beyond the capability of bilinear maps.

Another assumption one could make, along the same lines, for symmetric bilinear maps where $\mathbb{G}_1 = \mathbb{G}_2$ is the perhaps confusingly named **decisional bilinear Diffie-Hellman (DBDH)** assumption which states that

$$(g, g^x, g^y, g^z, e(g,g)^{xyz}) \approx_c (g, g^x, g^y, g^z, e(g,g)^u)$$

where $g \in \mathbb{G}_1$ and $x, y, z, u \leftarrow \mathbb{Z}_p$ are uniformly random.

Notation. To simplify presentation, we will resort to shorthand notation when describing group elements. We will let $[a]_1$ for $a \in \mathbb{Z}_p$ denote g_1^a for some fixed $g_1 \in \mathbb{G}_1$ which will be clear from the context. Similarly, $[a]_2$ for g_2^a with $g_2 \in \mathbb{G}_2$, and $[a]_T$ for g_T^a for some $g_T \in \mathbb{G}_T$. For example, we will write statement such as

$$e([a]_1, [b]_2) = e([1]_1, [ab]_2)$$

All vectors in this lecture are row vectors unless otherwise specified.

2 Degree-1 FE: A Warmup

We will start by constructing a functional encryption scheme for linear functions

$$F_{\alpha_1,\ldots,\alpha_\ell}(x_1, \ldots, x_\ell) = \sum_{i=1}^{\ell} \alpha_i x_i = \boldsymbol{\alpha}\mathbf{x}^T$$

We use an Abelian group $\mathbb{G}$ of order p, a prime. Here, we will not need bilinear maps, rather a group $\mathbb{G}$ where the DDH assumption holds.

- msk: $\mathbf{s} \leftarrow \mathbb{Z}_p^\ell$ and mpk: $g^{\mathbf{s}}$.
- $\mathsf{Enc}(\mathsf{mpk}, \mathbf{x})$: pick a random $r \leftarrow \mathbb{Z}_p$ and output

  $$\mathsf{ct} := (g^r, g^{r\mathbf{s}+\mathbf{x}})$$

  (Note that one can compute $g^{rs_i+x_i}$ using the public key g^{s_i} alone; no need to know s_i.)
- $\mathsf{Keygen}(\mathsf{msk}, \boldsymbol{\alpha})$: outputs

  $$\mathsf{sk}_{\boldsymbol{\alpha}} = \boldsymbol{\alpha}\mathbf{s}^T \in \mathbb{Z}_p$$

- Dec: Compute

  $$g^{\boldsymbol{\alpha}(r\mathbf{s}+\mathbf{x})^T} \cdot g^{-r\boldsymbol{\alpha}\mathbf{s}^T} = g^{\boldsymbol{\alpha}\mathbf{x}^T}$$

  where the first multiplicand is computed using the second component of the ciphertext and $\boldsymbol{\alpha}$, and the second multiplicand using the first component of the ciphertext and the functional secret key.

We will show security in just a moment, but a few things are worth noting.

1. We only get the output in the exponent. This is a shortcoming that will stay with us throughout this lecture (and indeed, something that we don't know how to overcome at all). Fortunately, at the end, in our application to XIO (cf. Neekon's lecture), we will compute functions whose outputs are binary, so given the output in the exponent, we can figure it out in the clear by brute force search.

2. There is no compression here. That is, by virtue of the function being linear, its description size (namely ℓ elements of $\mathbb{Z}_p$) is the same as the input length. Now, you could try using this to construct a degree-2 FE in the following manner. Write a quadratic function $Q(x_1, \ldots, x_\ell) = \sum_{i,j} \alpha_{i,j} x_i x_j$ as a linear function

$$F_{\boldsymbol{\alpha}}(\mathbf{x}) = \langle \boldsymbol{\alpha}, \mathbf{x} \otimes \mathbf{x} \rangle$$

That is, a linear function of $\mathbf{x}$ tensored with itself. However, now, the ciphertext size will be $O(\ell^2)$ making it non-compact. In our application to XIO via the JLS result, we will need the encryption to be compact, i.e. with size $O(\ell)$. To achieve this, we will need to turn to groups that admit bilinear maps.

2.1 Simulation Security

We will show how to simulate $\mathsf{mpk}, \mathsf{sk}_f$ and ct_x given only f and $f(x)$. (The generalization to multiple functional secret keys follows analogously.) Roughly speaking, this shows that mpk, sk_f and ct_x together reveal no more information than $f(x)$.

Let $f(x) = \sum_{i=1}^{\ell} \alpha_i x_i \bmod p$.

- Simulate the functional secret key sk_f for a linear function as a random number $s_f \leftarrow \mathbb{Z}_p$. Note that this is the correct distribution: the marginal distribution of sk_f is indeed uniformly random.

- Assume that $\alpha_1 \neq 0$. For fixed values of $s_2, \ldots, s_\ell$, note that

$$s_1 = \left(s_f - \sum_{i>1} \alpha_i s_i\right)/\alpha_1$$

- Simulate mpk as $(g^{s_1}, \ldots, g^{s_\ell})$ where s_1 is computed as above. Note that this can be done using s_f and $g^{s_2}, \ldots, g^{s_\ell}$.

- To simulate the ciphertext, note that

$$\begin{aligned} g^{r s_1 + x_1} &= g^{r s_f \alpha_1^{-1}} \cdot \prod_{i>1} g^{-r s_i \alpha_i \alpha_1^{-1}} \cdot g^{x_1} \\ &= g^{r s_f \alpha_1^{-1}} \cdot \prod_{i>1} g^{-(r s_i + x_i)\alpha_i \alpha_1^{-1}} \cdot \prod_{i>1} g^{x_i \alpha_i \alpha_1^{-1}} \cdot g^{x_1} \\ &= g^{r s_f \alpha_1^{-1}} \cdot g^{(\sum \alpha_i x_i)\cdot \alpha_1^{-1}} \cdot \prod_{i>1} g^{-(r s_i + x_i)\alpha_i \alpha_1^{-1}} \end{aligned}$$

This can be generated using g^r, $g^{r s_i + x_i}$ and $\sum \alpha_i x_i$ alone (together with other public information the simulator knows, such as the α_i and s_f). In fact, $\sum \alpha_i x_i$ is unnecessary; it suffices to have $g^{\sum \alpha_i x_i}$, an observation that will come in handy later.

However, $g^r, g^{r s_2 + x_2}, \ldots, g^{r s_\ell + x_\ell}$ are pseudorandom by the Diffie-Hellman assumption. Thus, the simulator can do its job and generate a computationally indistinguishable simulation given only α_i and $\sum \alpha_i x_i$ (and in particular, no other information about the x_i).

2.2 An Extension

Before proceeding further, let us mention an extension of this scheme where the input $\mathbf{x}$ and the function $\boldsymbol{\alpha}$ are given in the exponent. That is, Keygen is given $[\boldsymbol{\alpha}]_2$ and Enc is given $[\mathbf{x}]_1$. (We also switch to the compact notation.)

- msk: $\mathbf{s} \leftarrow \mathbb{Z}_p^\ell$ and mpk: $[\mathbf{s}]_1$.
- $\mathsf{Enc}(\mathsf{mpk}, \mathbf{x})$: pick a random $r \leftarrow \mathbb{Z}_p$ and output

  $$\mathsf{ct} := [r]_1, [r\mathbf{s} + \mathbf{x}]_1$$

  Note that encryption requires only $[\mathbf{x}]_1$, not $\mathbf{x}$ "in the clear".
- $\mathsf{Keygen}(\mathsf{msk}, \boldsymbol{\alpha})$: outputs

  $$\mathsf{sk}_{\boldsymbol{\alpha}} = [\boldsymbol{\alpha}^T \mathbf{s}]_2$$

  Note that key generation requires only $[\boldsymbol{\alpha}]_2$, not $\boldsymbol{\alpha}$ "in the clear".
- Dec: Compute, with the aid of the bilinear map,

  $$[\boldsymbol{\alpha}^T(r\mathbf{s} + \mathbf{x}) - r\boldsymbol{\alpha}^T\mathbf{s}]_T = [\boldsymbol{\alpha}^T\mathbf{x}]_T$$

  Note that decryption also requires only $[\boldsymbol{\alpha}]_2$, not $\boldsymbol{\alpha}$ "in the clear", and, as before, the result is obtained in the exponent of $\mathbb{G}_T$.

We will leave the proof of security as an exercise but note that as above, simulation should only require $[\boldsymbol{\alpha}^T\mathbf{x}]_1$ and not $\boldsymbol{\alpha}^T\mathbf{x}$ in the clear.

2.3 FE Through The Lens of PSM Protocols

Many (but not all) modern FE constructions are best viewed from the lens of a simple, information-theoretic, game called a private simulataneous messages (or PSM) game. In such a game, Alice has a function f, Bob has an input x, and they share a private random string that we will suggestively call msk. The goal is for both of them to send *a single message* to Charlie at the end of which we want:

- Correctness: Charlie should be able to compute $f(x)$ given Alice and Bob's messages.
- Security: Alice and Bob's messages should reveal nothing other than $f(x)$. This is formalized via a simulator in the usual cryptographic way.

PSM for Linear Functions. Imagine a simple setting where Alice holds a vector $\boldsymbol{\alpha}$ that defines a linear function, Bob holds an input $\mathbf{x}$ and the function

$$F(\boldsymbol{\alpha}, \mathbf{x}) = (\boldsymbol{\alpha}, \langle \boldsymbol{\alpha}, \mathbf{x} \rangle \bmod q)$$

A protocol to do this would have the shared random string be $\mathbf{s} \in \mathbb{Z}_p^n$, Bob sends $\mathbf{s} + \mathbf{x} \bmod p$ and Alice sends $\boldsymbol{\alpha}$ and $\langle \boldsymbol{\alpha}, \mathbf{s} \rangle \bmod p$. Charlie computes

$$\langle \boldsymbol{\alpha}, \mathbf{s} + \mathbf{x} \rangle - \langle \boldsymbol{\alpha}, \mathbf{s} \rangle \bmod p = \langle \boldsymbol{\alpha}, \mathbf{x} \rangle \bmod p$$

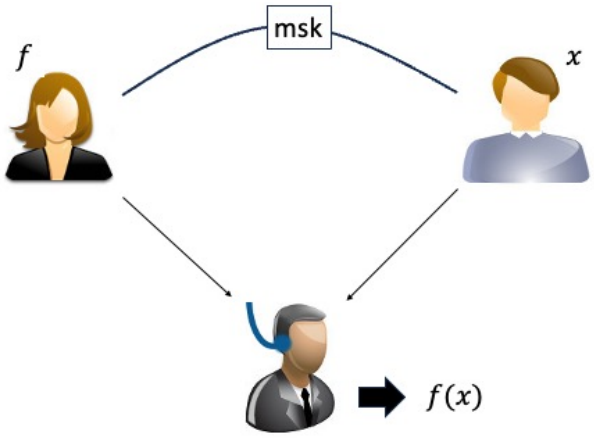

Figure 1: A Private Simultaneous Messages (PSM) Protocol.

as desired. In the language of FE, this gives us a one-ciphertext, one-key, secret-key FE scheme for linear functions.

It is clear that security is compromised if Bob uses the same msk to "encrypt" two different $\mathbf{x}$. This is where groups and the Diffie-Hellman assumption come in and give us the scheme that we just presented. On the other hand, one can show that security is *not* compromised if Alice uses the same msk for many different linear functions. However, that is a fortuitous coincidence.

In general, the process of designing an FE scheme can be thought of as first designing a PSM protocol for the corresponding functionality, and then "lifting it" using cryptographic assumptions to many-key, many-ciphertext security, and possibly also to the public-key setting.

PSM for Inner Products. Imagine now that you want to do the same thing, but Alice wants to keep $\boldsymbol{\alpha}$ private as well. Here is how one does this.

- msk is $\mathbf{r}, \mathbf{s}$.
- Alice sends $\boldsymbol{\alpha} + \mathbf{r}$ and Bob sends $\mathbf{x} + \mathbf{s}$.
- Charlie can now compute
$$\langle \boldsymbol{\alpha} + \mathbf{r}, \mathbf{x} + \mathbf{s} \rangle = \langle \boldsymbol{\alpha}, \mathbf{x} \rangle + \text{cross-terms}$$
where either Bob or Alice can compute each element of the cross-terms.
- In addition to the above, Alice sends Charlie $\langle \boldsymbol{\alpha}, \mathbf{s} \rangle + r'$ where msk contains a scalar r' in addition to the above.
- In addition to the above, Bob sends Charlie $\langle \mathbf{r}, \mathbf{x} + \mathbf{s} \rangle - r'$.

I will leave checking correctness and proving security of this protocol as an exercise.

3 Degree-2 FE from Bilinear Maps

Let's start with a PSM protocol for the function

$$F(\boldsymbol{\alpha}, (\mathbf{x}, \mathbf{y})) = (\boldsymbol{\alpha}, (\mathbf{x} \otimes \mathbf{y}) \cdot \boldsymbol{\alpha}^T \bmod p) \tag{2}$$

where $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_p^\ell$ and $\boldsymbol{\alpha} \in \mathbb{Z}_p^{\ell^2}$. (Note that we treat vectors as row vectors so $\mathbf{x} \otimes \mathbf{y}$ is a row vector, and the column vector $\boldsymbol{\alpha}^T$ denotes the transpose of $\boldsymbol{\alpha}$.)

We'd like the communication from Bob to be of size $O(\ell)$. As noted before, achieving communication $O(\ell^2)$ is trivial by appealing to the protocol for linear functions, but we want to (and need to) do better.

So, let's build up a PSM protocol for F from first principles.

- The msk is a pair of strings $\mathbf{r}, \mathbf{s} \leftarrow \mathbb{Z}_p^\ell$.
- Bob needs to hide $\mathbf{x}$ and $\mathbf{y}$ so he "one-time pads" them and sends $\mathbf{x} + \mathbf{r} \bmod p$ and $\mathbf{y} + \mathbf{s} \bmod p$ to Charlie.
- Alice sends $\boldsymbol{\alpha}$ to Charlie.
- What can Charlie do with this information? He can compute

  $$((\mathbf{x} + \mathbf{r}) \otimes (\mathbf{y} + \mathbf{s})) \cdot \boldsymbol{\alpha}^T = (\mathbf{x} \otimes \mathbf{y}) \cdot \boldsymbol{\alpha}^T + \text{cross-terms} \tag{3}$$

  so if he can remove the cross-terms somehow, he is all set.

Let's write down

$$\Big((\mathbf{x} + \mathbf{r}) \otimes \underbrace{(\mathbf{y} + \mathbf{s})}_{:=\mathbf{z}}\Big) \cdot \boldsymbol{\alpha}^T = (\mathbf{x} \otimes \mathbf{y}) \cdot \boldsymbol{\alpha}^T + \underbrace{(\mathbf{x} \otimes \mathbf{s}) \cdot \boldsymbol{\alpha}^T + (\mathbf{r} \otimes \mathbf{z}) \cdot \boldsymbol{\alpha}^T}_{\text{cross-terms}} \tag{4}$$

There are two ways to handle the cross-terms.

1. The first way is to observe that the cross terms are a linear function of $\mathbf{x}$ and $\mathbf{z}$. Bob knows $\mathbf{x}$ and $\mathbf{z}$, and Alice knows the description of the *secret* linear function (secret because it depends on $\mathbf{r}, \mathbf{s}$ in addition to $\boldsymbol{\alpha}$.) Now, invoke the PSM protocol for computing inner products from our previous discussion. This will give us a PSM protocol for degree-2 functions with $O(\ell)$ communication. This can in turn be converted into an FE scheme by "putting the elements in the exponent" appropriately. But we will take a different route.

2. The second way, which will lead us to our FE scheme, is to write down the cross-terms differently, as a *public* linear function of $\mathbf{x}, \mathbf{z}, \mathbf{s}, \mathbf{r}$. Now, the obvious way of doing this will result in a linear function in ℓ^2 dimensions and therefore a protocol with communication $O(\ell^2)$. But we will do better using cryptography.

   Imagine for a moment that $\mathbf{r} = r\mathbf{a}_1$ and $\mathbf{s} = s\mathbf{a}_2$ are scalar multiples of *public* vectors $\mathbf{a}_1, \mathbf{a}_2$. Then,

   $$\text{cross-terms} = (\mathbf{x} \otimes \mathbf{s}) \cdot \boldsymbol{\alpha}^T + (\mathbf{r} \otimes \mathbf{z}) \cdot \boldsymbol{\alpha}^T = (\mathbf{x} \otimes s\mathbf{a}_2) \cdot \boldsymbol{\alpha}^T + (r\mathbf{a}_1 \otimes \mathbf{z}) \cdot \boldsymbol{\alpha}^T = [\mathbf{x} \otimes s \parallel r \otimes \mathbf{z}] \cdot \underbrace{\begin{bmatrix} \mathbf{I} \otimes \mathbf{a}_2 \\ \mathbf{a}_1 \otimes \mathbf{I} \end{bmatrix} \cdot \boldsymbol{\alpha}^T}_{\text{linear fn}} \tag{5}$$

   which is a public linear function in $(s\mathbf{x}, r\mathbf{z})$. This can be computed using our first PSM protocol, the one for linear functions.

   The problem, however, is that $\mathbf{s}$ and $\mathbf{r}$ are now not random and therefore leak information. We will solve this by "putting them in the exponent" and invoking an appropriate version of the Diffie-Hellman assumption (in particular, k-LIN for $k > 1$).

3.1 The FE Scheme

A First Try. Our first attempt will have essentially all the ideas except it will be insecure because of the failure of the decisional Diffie-Hellman assumption. We will nevertheless be able to fix it under the k-LIN assumption for $k > 1$. (Recall that 1-LIN is DDH.)

- mpk is $[\mathbf{a}_1]_1, [\mathbf{a}_1]_2, [\mathbf{a}_2]_2, [\mathbf{w}]_1$, and $\mathsf{msk} = (\mathbf{a}_1, \mathbf{a}_2, \mathbf{w})$.
- $\mathsf{Enc}(\mathsf{mpk}, (\mathbf{x}, \mathbf{y}))$ picks random $r, s \leftarrow \mathbb{Z}_p$ and computes $[\mathbf{x} + r\mathbf{a}_1]_1$ and $[\mathbf{y} + s\mathbf{a}_2]_2$. It also picks $s' \leftarrow \mathbb{Z}_p$ and computes $[s']_1$ and $[s'\mathbf{w} + (s\mathbf{x} \| r\mathbf{z})]_1$ and outputs all of them as the ciphertext. That is,
$$\mathsf{ct} = \left([\mathbf{x} + r\mathbf{a}_1]_1, [\mathbf{y} + s\mathbf{a}_2]_2, [s']_1, [s'\mathbf{w} + (s\mathbf{x} \| r\mathbf{z})]_1\right)$$
- $\mathsf{Keygen}(\mathsf{msk}, \boldsymbol{\alpha})$ outputs
$$\mathsf{sk}_{\boldsymbol{\alpha}} := [\mathbf{w}\boldsymbol{\beta}^T]_2$$
where
$$\boldsymbol{\beta}^T = \begin{bmatrix} \mathbf{I} \otimes \mathbf{a}_2 \\ \mathbf{a}_1 \otimes \mathbf{I} \end{bmatrix} \cdot \boldsymbol{\alpha}^T$$
Note that this can be computed given $[\mathbf{a}_1]_2, [\mathbf{a}_2]_2, \mathbf{w}$ and $\boldsymbol{\alpha}$.
- Decryption uses the last two components of the ciphertext to compute
$$[(s'\mathbf{w} + (s\mathbf{x} \| r\mathbf{z})) \cdot \boldsymbol{\beta}^T - s'(\mathbf{w}\boldsymbol{\beta}^T)]_T = [(s\mathbf{x} \| r\mathbf{z}) \cdot \boldsymbol{\beta}^T]_T$$
using the fact that mpk allows Dec to compute $[\boldsymbol{\beta}^T]_2$. One can now recover the output by computing
$$[((\mathbf{x} + r\mathbf{a}_1) \otimes (\mathbf{y} + s\mathbf{a}_2)) \cdot \boldsymbol{\alpha}^T]_T = [(\mathbf{x} \otimes \mathbf{y}) \cdot \boldsymbol{\alpha}^T + (s\mathbf{x} \| r\mathbf{z}) \cdot \boldsymbol{\beta}^T]_T$$
and dividing by $[(s\mathbf{x} \| r\mathbf{z}) \cdot \boldsymbol{\beta}^T]_T$.

All seems well except that given $[\mathbf{a}_1]_2$, DDH does not hold in $\mathbb{G}_1$ w.r.t. $\mathbf{a}_1$, i.e. $[r\mathbf{a}_1]_1$ does not look random any more. Indeed, letting the first coordinates of $\mathbf{a}_1$ be (a_{11}, a_{12}), we can check the equation
$$e([ra_{11}]_1, [a_{12}]_2) \stackrel{?}{=} e([a_{11}]_1, [ra_{12}]_2)$$
which is a distinguisher. Since $[r\mathbf{a}_1]_1$ is not pseudorandom any more, we cannot use it as a one-time pad to hide $\mathbf{x}$.

It seems necessary to publish $[\mathbf{a}_1]_2$ in the mpk for the decryption algorithm to do its job. So, we seem to be stuck.

The Actual Scheme. Since DDH failed, let's lean in on k-LIN for $k > 1$. Recall that the DDH assumption says $[\mathbf{a}]_1, [s\mathbf{a}_1]_1$ is pseudorandom where $\mathbf{a} \in \mathbb{Z}_p^{1\times m}$ for a polynomially large m. We just saw that this is not true once one gets their hands on $[\mathbf{a}]_2$ as well.

The (bilateral) k-LIN assumption says that
$$([\mathbf{A}]_i, [\mathbf{sA}]_i)_{i\in\{1,2,T\}} \approx_c ([\mathbf{A}]_i, [\mathbf{u}]_i)_{i\in\{1,2,T\}}$$
where $\mathbf{A} \leftarrow \mathbb{Z}_p^{k\times m}$. Now, one can check that the attack we had before does not go through any more. This assumption is believed to hold and is a special case of an "uber-assumption" on bilinear maps [BBG05, Appendix A].

With this, one can write down the final scheme.

- mpk is $[\mathbf{A}_1]_1, [\mathbf{A}_1]_2, [\mathbf{a}_2]_2, [\mathbf{w}]_1$, and $\mathsf{msk} = (\mathbf{A}_1, \mathbf{a}_2, \mathbf{w})$, where $\mathbf{A}_1 \leftarrow \mathbb{Z}_p^{2\times\ell}$.
- $\mathsf{Enc}(\mathsf{mpk}, (\mathbf{x}, \mathbf{y}))$ picks random $r \leftarrow \mathbb{Z}_p^2$, $s \leftarrow \mathbb{Z}_p$ and computes $[\mathbf{x} + \mathbf{r}\mathbf{A}_1]_1$ and $[\mathbf{y} + s\mathbf{a}_2]_2$. It also picks $s' \leftarrow \mathbb{Z}_p$ and computes $[s']_1$ and $[s'\mathbf{w} + (s\mathbf{x} \parallel \mathbf{r} \otimes \mathbf{z})]_1$ and outputs all of them as the ciphertext. That is,
$$\mathsf{ct} = \left([\mathbf{x} + \mathbf{r}\mathbf{A}_1]_1, [\mathbf{y} + s\mathbf{a}_2]_2, [s']_1, [s'\mathbf{w} + (s\mathbf{x} \parallel \mathbf{r} \otimes \mathbf{z})]_1\right)$$
- $\mathsf{Keygen}(\mathsf{msk}, \boldsymbol{\alpha})$ outputs
$$\mathsf{sk}_{\boldsymbol{\alpha}} := [\mathbf{w}\boldsymbol{\beta}^T]_2$$
where
$$\boldsymbol{\beta}^T = \begin{bmatrix} \mathbf{I} \otimes \mathbf{a}_2 \\ \mathbf{A}_1 \otimes \mathbf{I} \end{bmatrix} \cdot \boldsymbol{\alpha}^T$$
Note that this can be computed given $[\mathbf{A}_1]_2, [\mathbf{a}_2]_2, \mathbf{w}$ and $\boldsymbol{\alpha}$.
- Decryption is exactly as before. It uses the last two components of the ciphertext to compute
$$[(s'\mathbf{w} + (s\mathbf{x} \| \mathbf{r} \otimes \mathbf{z})) \cdot \boldsymbol{\beta}^T - s'(\mathbf{w}\boldsymbol{\beta}^T)]_T = [(s\mathbf{x} \| \mathbf{r} \otimes \mathbf{z}) \cdot \boldsymbol{\beta}^T]_T$$
using the fact that mpk allows Dec to compute $[\boldsymbol{\beta}^T]_2$. One can now recover the output by computing
$$[((\mathbf{x} + \mathbf{r}\mathbf{A}_1) \otimes (\mathbf{y} + s\mathbf{a}_2)) \cdot \boldsymbol{\alpha}^T]_T = [(\mathbf{x} \otimes \mathbf{y}) \cdot \boldsymbol{\alpha}^T + (s\mathbf{x} \| \mathbf{r} \otimes \mathbf{z}) \cdot \boldsymbol{\beta}^T]_T$$
and dividing by $[(s\mathbf{x} \| \mathbf{r} \otimes \mathbf{z}) \cdot \boldsymbol{\beta}^T]_T$.

3.2 From Degree-$(1, O(1))$ FE to Degree-$(2, O(1))$ FE

We now need to compute
$$F(p, (\boldsymbol{\alpha}, \mathbf{x}, \mathbf{y})) = (p, \boldsymbol{\alpha}, (\mathbf{x} \otimes \mathbf{y}) \cdot p(\boldsymbol{\alpha})^T \bmod p) \tag{6}$$
where $p(\boldsymbol{\alpha})$ denotes a vector each of whose coefficients is a degree-$O(1)$ polynomial computed on $\boldsymbol{\alpha}$. What we've been doing so far is the case of p being the constant function which ignores $\boldsymbol{\alpha}$ and outputs a constant coefficient vector. Note that neither p nor $\boldsymbol{\alpha}$ needs to be hidden.

One could go through the calculations as above and figure out that the cross terms are
$$\mathsf{cross\text{-}terms} = [\mathbf{x} \otimes s \mathbin{/\!/} r \otimes \mathbf{z}] \cdot \underbrace{\begin{bmatrix} \mathbf{I} \otimes \mathbf{a} \\ \mathbf{a} \otimes \mathbf{I} \end{bmatrix} \cdot p(\boldsymbol{\alpha})^T}_{\text{linear fn}} \tag{7}$$
which, as a function of $\mathbf{x}$, $\mathbf{y}$ and $\boldsymbol{\alpha}$, is degree-1 in $\mathbf{x}$ and $\mathbf{z}$ and degree-$O(1)$ in $\boldsymbol{\alpha}$. Thus, given a degree-$(1, O(1))$-FE scheme, we can immediately build a degree-$(2, O(1))$-FE scheme.

We will not be able to get to the construction of a degree-$(1, O(1))$-FE scheme in the lecture but (a) we will present a degree-$(0, O(1))$-FE scheme, aka an ABE scheme for constant-degree polynomials. In fact, we will do one better and tell you about an ABE scheme for NC^1) functions, generalizing constant-degree polynomials; and (b) we will refer you to Wee's paper [Wee20] for the actual construction of degree-$(1, O(1))$-FE.

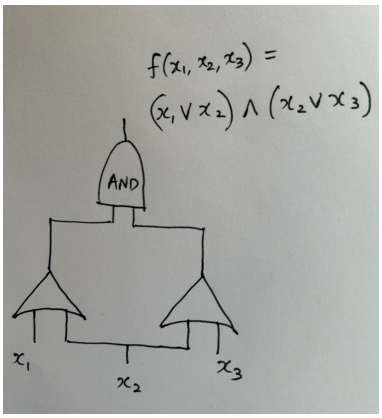

Figure 2: A Boolean formula on variables x_1, x_2, x_3.

4 ABE for NC^1 Functions

4.1 A Tool: Linear Secret Sharing (LSSS)

Fix a function $f : \{0,1\}^\ell \to \{0,1\}$. Say you have a secret number s and you'd like to share it among ℓ parties such that if a subset $S \subseteq [\ell]$, also denoted by its characteristic vector $\mathbf{x}_S \in \{0,1\}^\ell$, come together, they can learn s if $f(\mathbf{x}_S) = 1$; and they cannot otherwise. For this to make sense, we will need f to be monotone: i.e. flipping a 0 to a 1 in the input cannot flip the output from 1 to 0.

We will show a scheme to do this where the size of the shares is polynomial in the size of the monotone Boolean formula (i.e. a formula with AND and OR gates) computing f. We will use the formula in Figure 2 as a running example.

To secret share s w.r.t f, do the following, starting from the output wire:

- label the output wire with the secret s;
- For each AND gate whose output wire w has a label s_w, assign labels s_u and s_v to the input wires u and v respectively, where s_u, s_v are random subject to $s_u + s_v = s_w$.

  In the example above, we will let these labels be random s_1 and s_2 s.t. $s_1 + s_2 = s$.
- For each OR gate whose output wire w has a label s_w, assign labels s_u and s_v to the input wires u and v respectively, where $s_u = s_w$ and $s_v = s_w$.
- The i-th share consists of the labels for all input wires that use the variable x_i.

  In the example above, we will let the label of the x_1 input wire be s_1; that of the x_3 input wire be s_2; and that of the x_2 input wire be the pair (s_1, s_2).

Given labels corresponding to a set of input wires $\{i : x_i = 1\}$, it is not hard to see that we can recover the labels of every wire that evalutes to 1 on input $\mathbf{x} = (x_1, \ldots, x_\ell)$, in particular the output wire. On the other hand, for every wire that evalutes to 0, the label is completely hidden. This gives us correctness and

secrecy. Furthermore, the reconstruction function which takes as input a subset of the shares (corresponding to a vector $\mathbf{x}$) and outputs the secret (if $f(\mathbf{x}) = 1$) is a *linear function*. So the scheme above is a linear secret sharing scheme.

In general a linear secret sharing scheme consists of:

- A sharing algorithm that takes as input a secret $s \in \mathbb{Z}_p$ and a monotone Boolean function f, and outputs a sequence of numbers $(\alpha_1, \ldots, \alpha_m)$ together with a function $\rho : [m] \to [\ell]$.
- A reconstruction algorithm which, for every $\mathbf{x} \in \{0,1\}^\ell$ s.t. $f(\mathbf{x}) = 1$, gets a subset of the shares

$$\alpha_{\mathbf{x}} := \left(\alpha_i : x_{\rho(i)} = 1\right)$$

  and outputs the secret s. Moreover, this is a linear function: there is a vector $f_{\mathbf{x}}$ such that the output of the reconstruction algorithm is $f_{\mathbf{x}} \alpha_{\mathbf{x}}^T$.

4.2 The GPSW ABE scheme

- $\mathsf{msk} = s$ and $\mathsf{mpk} = [t_1]_1, \ldots, [t_\ell]_1, [s]_T$.
- $\mathsf{Enc}(\mathsf{mpk}, x_1, \ldots, x_\ell, m)$ where $x_i \in \{0,1\}$ computes

$$[r t_1 \cdot x_1]_1, \ldots, [r t_\ell \cdot x_\ell]_1, [rs + m]_T$$

- $\mathsf{Keygen}(\mathsf{msk}, f)$ where f is an NC^1 circuit does the following: using the LSS scheme for f, compute a set of numbers $(\alpha_1, \ldots, \alpha_m)$ together with a mapping $\rho : [m] \to [\ell]$ denoting which index / party each share belongs to. The secret key for f consists of

$$\left([\alpha_i \cdot t_{\rho(i)}^{-1}]_2\right)_{i \in [m]}$$

- Dec first computes

$$[r\alpha_i]_T$$

  for each $i \in [m]$ for which $x_{\rho(i)} = 1$. From this, using LSS reconstruction, compute $[rs]_T$ and therefore $[m]_T$.

References

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. Cryptology ePrint Archive, Paper 2005/015, 2005.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[Wee20] Hoeteck Wee. Functional encryption for quadratic functions from k-lin, revisited. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*, pages 210–228. Springer, 2020.