1 Definition

This session is concerned with the idea of a *delegation scheme*. The idea is that a weak *verifier* wants to outsource the computation of $y = f(x)$ to a powerful entity (the *prover*), for some given f and x. A delegation scheme is called secure if, for any PPT prover, it is unlikely that the prover can convince the verifier that $y' = f(x)$ for some $y \neq y'$. More formally, a delegation scheme involves the following steps:

- $\textsc{Preprocessing}(f, 1^k) \to (ek, vk)$: the verifier performs this step once for a given function f to obtain a public evaluation key ek and a (possibly private) verification key vk. The security parameter is k.
- $\textsc{Query}(x, vk) \to (X, s)$: the verifier forms a query message X that encodes the input value x in some form. This is then sent to the prover. s is some extra state information that the verifier keeps for later.
- $\textsc{Eval}(X, ek) \to \pi$: for a given query X, the prover performs this step to compute some cryptographic proof π that will (1) enable the verifier to obtain $y = f(x)$ and (2) convince the verifier that y is indeed the correctly computed output of f.
- $\textsc{Verify}(s, \pi) \to (b, y)$: the verifier can now use a special $\textsc{Verify}$ procedure to obtain the value y. Here, $b \in \{0, 1\}$ indicates whether the verification was successful: $(1, y)$ indicates success, while $(0, \perp)$ indicates failure. Ideally, we want $\textsc{Verify}$ to take much less time than just computing $y = f(x)$ directly.

So we say that a delegation scheme as just defined is secure iff, for all PPT adversarial prover $\mathcal{ADV}$, it holds that:

$$\forall f, x. \Pr\left(\begin{array}{l} (ek, vk) \leftarrow \textsc{Preprocessing}(f); \\ (X, s) \leftarrow \textsc{Query}(x, vk); \\ (b, y') \leftarrow \textsc{Verify}(s, \mathcal{ADV}(X, ek)); \\ f(x) \neq y' \neq \perp \wedge b = 1 \end{array}\right) = \mathsf{negl}(k)$$

2 Outline

In this session we describe how to obtain a delegation scheme from FHE, at least with the added assumption of the FHE scheme having a deterministic evaluation function. After that, we slowly remove these restrictions, one by one. Finally, we talk about a variation of this technique, one which reduces the verifier's preprocessing time by introducing additional rounds of interactivity.

3 One-time Scheme with 50% Failure Rate

The one-time scheme is not really meant to be useful in any realistic situation, but is meant to be a simple starting point for more sensible schemes that we

describe later. Here, the goal is that we want to delegate the computation of $f(x)$, but in a rather restricted setting. In particular,

- We will tolerate a scheme that is secure for only a single invocation of Eval.
- Even for that single invocation, we are okay if the adversary has a 50% chance of winning the security game.

3.1 Approach 1

One idea is to simply ask the prover (who is also our adversary) to homomorphically evaluate on both $X_0 \leftarrow \mathsf{Enc}_{\mathrm{FHE}}(x)$ as well as $X_1 \leftarrow \mathsf{Enc}_{\mathrm{FHE}}(0)$. These are sent to the prover in random order, after which we can decrypt and verify if one of their decryption evaluates to $y_0 = f(0)$. This special check value y_0 can be computed during the Preprocessing phase. The hope here is that since the prover cannot distinguish between X_0 and X_1, it has to evaluate them both homomorphically, or else it will not produce y_0 for the verifier to check. The prover can of course risk getting caught by evaluating only one of these and returning random ciphertext for the other. In that case, there is a 50% chance that the prover gets caught, depending on what the prover chose.

Unfortunately, this is not quite secure: it is very easy for the prover to win every time by returning $\textsc{Eval}(X_1, ek)$ as a result of both inputs. However, it still has most of the ingredients we need for security. More importantly, it has the property that verification time does not depend on the complexity of f — it only depends on the size of x and y.

3.2 Approach 2

We fix the security hole above by assuming our FHE scheme provides deterministic evaluation. Our delegation scheme now looks like this:

- $\textsc{Preprocessing}(f, 1^k)$: Produces $(ek, vk) = (pk, (X_1, \bar{Y}_1, sk))$ as follows: $(pk, sk) \leftarrow \textsc{KeyGen}_{\mathrm{FHE}}(1^k)$. $X_1 \leftarrow \mathsf{Enc}_{\mathrm{FHE}}(0, pk)$, $\bar{Y}_1 = \textsc{Eval}_{\mathrm{FHE}}(X_1, pk)$.
- $\textsc{Query}(x, (X_1, \cdot, sk))$: Computes $X_0 \leftarrow \mathsf{Enc}_{\mathrm{FHE}}(x, sk)$, produces $X = (X_c, X_{1-c})$ and $s \leftarrow \{0, 1\}$
- $\textsc{Eval}((X_s, X_{1-s}), pk)$: Computes $\pi = (Y_s, Y_{1-s})$, where $Y_i = \textsc{Eval}_{\mathrm{FHE}}(X_i, pk)$.
- $\textsc{Verify}(s, (Y_s, Y_{1-s}), (\cdot, \bar{Y}_1, sk))$: Checks $Y_1 \stackrel{?}{=} \bar{Y}_1$. On success, produces $(1, \mathsf{Dec}_{\mathrm{FHE}}(Y_0, sk))$. On failure, it produces $(0, \perp)$.

The proof for security is fairly straightforward, albeit with a loose bound of just 50% verification rate. The idea is that protocol execution, from the point of view of the adversary executing Eval, is indistinguishable from an execution of a different run of the protocol where $x = 0$. But if $x = 0$, we can see that the adversary has a 50% chance of winning, by guessing s. Therefore, in the real execution where $x \neq 0$, the adversary still wins with the same probability.

4 Improving Verfication

The scheme as described still has two major issues: it is a single-use scheme, and the verification scheme can fail with a very high probability of 50%. In this section we solve the second issue. As often with probabilistic algorithms, simply repeating it multiple times is enough to reduce error bound. Concretely, we now create a new scheme that runs the above scheme in parallel k times with fresh randomness. Finally, at verification time, we do the following:

$\textsc{Verify}\left\{(s^{(i)}, Y_s^{(i)}, Y_{1-s}^{(i)}) \,\middle|\, i \in [1 \dots k]\right\}$: Checks $\forall i$, $Y_1^{(i)} \stackrel{?}{=} \bar{Y}_1^{(i)}$ and $\mathsf{Dec}_{\mathrm{FHE}}(Y_0^{(i)}, sk^{(i)}) \stackrel{?}{=} \mathsf{Dec}_{\mathrm{FHE}}(Y_0^{(0)}, sk^{(0)})$.

Each check can fail with a probability of 1/2, so the full scheme now fails with probability 2^{-k} (exercise question: do we need to check if the decrypted ciphertexts match? How would the error probability change if we didn't? Can you prove your answer?).

5 Multiple Use

Finally, we were not able to use this scheme multiple times so far since we must not let the adversary see the same X_1 values a second time. But we don't want to recompute fresh X_1 each time either, since that will require us to compute f each time — the thing we were trying to avoid in the first place.

The solution is conceptually straightforward: we just wrap everything in yet another layer of FHE scheme (with indepedently selected keys). This will allow us to send the same X_1 values each time, now encrypted, without allowing the adversary to see which values are being repeated.

6 Lower Preprocessing

Another interesting variant of this scheme is one that aims to reduce the initial preprocessing overhead, by allowing more interaction. Here, we let the server do all the preprocessing work under yet another layer of FHE, and then use PCP to check that it was indeed done correctly. This allows the verifier to avoid computing the function f k-times at initialization, which might be expensive.