Homework 1

Problem 1: FHE from VBB. In this problem, you will show that VBB obfuscation (for circuits) implies fully homomorphic encryption. Unfortunately, the straightforward approach — obfuscating the circuit that decrypts, performs operations on the plaintexts, and then re-encrypts — will not quite work. To see this, consider an attempted proof of security. The proof starts with an adversary A for the FHE scheme, and will construct an adversary B for the VBB obfuscation of the homomorphism program. Such an adversary B is trying to learn a single bit of information about the homomorphism program, and can be simulated by a simulator with black box access. Then the proof would show that such a simulator is impossible. Unfortunately, there is no place in the VBB definition to give B a ciphertext, meaning that whatever bit of information B is trying to learn has nothing to do with the plaintext A is trying to learn.

The above problem can be fixed by defining an even stronger notion of VBB security that allows B to receive some auxiliary information correlated with the program (this auxiliary information would be the ciphertext). However, we need to use our definition of VBB obfuscation in order to use FHE in the VBB impossibility result.

Instead, we will tweak the scheme. We will start the encryption scheme based on a PRF $\mathsf{PRF}_1 : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$. The secret key is a key k for PRF_1 . A ciphertext for a single-bit message m has the form $(r, \mathsf{PRF}_1(k,r) + m)$, where addition is carried out modulo 2. To allow homomorphic operations, we obfuscate a program which, as above, decrypts, performs the operation on the plaintexts, and the re-encrypts. However, we make two changes to the scheme:

- First, instead of encrypting by choosing a random r and outputing $r, \mathsf{PRF}_1(k,r)+m$, we do the following. The secret key contains $2n$ "base" ciphertexts (encrypting n 0's and n 1's), and to encrypt, homomorphically add (using the obfuscated program) a random subset of them in such a way that the output is guaranteed to encrypt the desired value.
- We modify the obfuscated program to also output the "base" ciphertexts.

The reason for these changes will become more clear when working through the proof. More precisely, let $\mathsf{PRF}_1 : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ and $\mathsf{PRF}_2 : \{0,1\}^\lambda \times \{0,1\}^{2\lambda+3} \to \{0,1\}^\lambda$ be PRFs. The scheme encrypts just single bits, and is described as follows:

- $\mathsf{Gen}(\lambda)$ generates a random keys $k, k' \in \{0,1\}^\lambda$. Then, for $i = 1, \dots, n$ (for an n to be chosen later) and $b \in \{0,1\}$, it chooses a random $r_{i,b} \in \{0,1\}^\lambda$, and set $c_{i,b} = (r_{i,b}, \mathsf{PRF}_1(k, r_{i,b}) + b) \in \{0,1\}^{\lambda+1}$. Here, $+$ means addition modulo 2. The $c_{i,b}$ will be the "base" ciphertexts (with $c_{i,b}$ encrypting the bit b) that will be homomorphically added to encrypt.

  Next, Gen constructs the circuit $C_{k,k',\{c_{i,b}\}}(op, d_0, d_1)$, where $op \in \{\perp, +, \times\}$ and $d_b \in \{0,1\}^{\lambda+1}$ for $b \in \{0,1\}$.

  If $op = \perp$, then $C_{k,k',\{c_{i,b}\}}(op, d_0, d_1)$ just outputs $\{c_{i,b}\}$. If $op \in \{+\times\}$, then $C_{k,k',\{c_{i,b}\}}(op, d_0, d_1)$ does the following. It parses each d_b as (e_b, f_b) for $e_b \in \{0,1\}^\lambda$ and $f_b \in \{0,1\}$. Then it computes $m_b = f_b + \mathsf{PRF}_1(k, e_b)$, and computes $m = m_0 \ op \ m_1$ (here, the operation is carried out modulo 2). Let $e = \mathsf{PRF}_2(k', (op, d_1, d_1))$ (here, op is treated as a bit). Let $f = \mathsf{PRF}_1(k, e) + m$. Finally, output $d = (e, f)$.

  Gen obfuscates $C_{k,k',\{c_{i,b}\}}$, obtaining $\hat{C}$. Next, it lets $\oplus = \hat{C}(\oplus, \cdot, \cdot)$, $\otimes = \hat{C}(\otimes, \cdot, \cdot)$. It outputs $(k, \{c_{i,b}\}, \oplus)$ as the secret key, and $\oplus, \otimes$ as the homomorphic operations.

- $\mathsf{Enc}(\ (k, \{c_{i,b}\}, \oplus)\ , m)$: Choose a random string $x \in \{0,1\}^n$, subject to the constraint that the parity of x is m (that is, $\sum_{i=1}^n x_i \mod 2 = m$). Then, use the homomorphic operations to add the ciphertexts c_{i,x_i} for $i \in [n]$. That is, first run $c_2 = c_{1,x_1} \oplus c_{2,x_2}$, then $c_3 = c_2 \oplus c_{3,x_3}$, then $c_4 = c_3 \oplus c_{4,x_4}$, $\dots$, until finally computing $c_n = c_{n-1} \oplus c_{n,x_n}$. Output c_n as the ciphertext. Notice that the ciphertext encrypts $\sum_{i=1}^n x_i \mod 2 = m$, as desired.

- $\mathsf{Dec}(\ (k, \{c_{i,b}\}, \oplus)\ , (e, f))$: Output $\mathsf{PRF}_1(k, e) + f$

Prove that this encryption scheme is secure (according to the definition given in the lecture notes for lecture 2), assuming that VBB obfuscation is used. What do you need to set n to?

Hint 1: First prove that the view of the adversary in the standard FHE experiment is indistinguishable from the view where $c_{i,b}$ are all set to encryptions of 0 (rather than $c_{i,1}$ being encryptions of 1)

Hint 2: Once you've followed Hint 1, you may find the following lemma useful:

Lemma 1. *Let D be some distribution over some set V. Let $Y_{i,b}$ for $i \in [n], b \in \{0,1\}$ be independent variables sampled according to D. Given a vector $x \in \{0,1\}^n$, let $Z_x = (Y_{1,x_1}, \dots, Y_{n,x_n})$. Let $h : V^n \to \{0,1\}^\ell$ and $g : V^{2n} \times \{0,1\}^\ell \to \{0,1\}$ be potentially probabilistic functions. Then*

$$\Pr_{Y_{i,b} \leftarrow D,\ x \leftarrow \{0,1\}^n} [\ g(\{Y_{i,b}\}_{i \in [n], b \in \{0,1\}}, h(Z_x)) = \sum_i x_i \mod 2\ ] \le \frac{1}{2} + 2^{-0.2n+\ell+1}$$

Problem 2: Universal Samplers from iO. A universal sampler, intuitively, is a program U, that on input a sampling procedure D, outputs a sample s from D. The program, however, is deterministic, so that repeated calls for the same D give the same sample. Therefore, if you and I both know U and run it on the same D, we will get the same sample s.

Ideally, for security, we would have that the sampler is "as good as" having a black box in the sky that takes as input a sampling procedure D, and returns a truly random sample generated by D. Moreover, this black box has ensures that repeated calls on the same D have consistent outputs, so you and I get the same sample.

It is possible to formalize a VBB-style security notion for universal samplers, but it will be unattainable just like VBB obfuscation. Instead, we will settle for the following weaker definition. A universal sampler is a triple of algorithms $(\mathsf{Gen}, \mathsf{Samp}, \mathsf{Sim})$ where:

- $\mathsf{Gen}(\lambda, m, n)$ takes as input the security parameter λ, a "description size" m, and an output length n. It is a PPT procedure that outputs parameters Params.
- $\mathsf{Samp}(\mathsf{Params}, D)$ is a deterministic procedure that takes as input Params, as well as a sampling procedure D given as a circuit whose size is at most m and whose samples have size at most n. It outputs a sample s.
- $\mathsf{Sim}(\lambda, m, n, D^*, s^*)$ is a PPT procedure that takes as input λ, m, n as before, a sampling procedure D^* of size at most m and output size at most n, and a sample s^* of size at most n. It outputs "simulated" parameters Params.

For correctness, we require that, if $\mathsf{Params} \leftarrow \mathsf{Sim}(\lambda, m, n, D^*, s^*)$, then $\mathsf{Samp}(\mathsf{Params}, D^*)$ outputs s^*. In other words, the simulated parameters have the sample for D^* programmed to s^*.

For security, we require that, for any m, n and sampler D^* of size at most m and output size at most n, the following holds:

$$\mathsf{Params} \leftarrow \mathsf{Gen}(\lambda, m, n) \quad \approx_c \quad \mathsf{Params} \leftarrow \mathsf{Sim}(\lambda, m, n, D^*, s^*) \text{ where } s^* \leftarrow D^*()$$

where $\approx_c$ means that the distributions on the left and right are indistinguishable, and $s^* \leftarrow D^*()$ means that s^* is a random sample from D^*. Intuitively, this means that for any D^*, the s^* generated as $\mathsf{Samp}(\mathsf{Params}, D^*)$ is "as good as" a random sample from D^*, since it can be simulated using a truly random sample.

Construct Universal Samplers from Indistinguishability Obfuscation. Prove the security of your construction.

Hint: Params *will be an obfuscated circuit that takes as input a sampling procedure D, and runs D on coins r for some r to obtain the output s. How should r be generated?*

Problem 3. Multiparty NIKE for an unbounded number of users using iO for Turing machines? Consider the construction of non-interactive multiparty key exchange seen in class. The program being obfuscated takes as input N public values $x_1, \ldots, x_n$, as well as an index $i \in [N]$ and seed s. The program checks that $\mathsf{PRG}(s) = x_i$, and if so, it outputs $\mathsf{PRF}(k,\ (x_1, \ldots, x_n)\ )$.

Because we obfuscate a circuit, which has bounded input size, once the obfuscated program is produced for N users, the maximum number of users that can exchange keys is bounded by N.

Now, what if instead we used a Turing machine obfuscator? We can define a Turing machine that is independent of N and carries out the above computation for any number of users. If we had a Turing machine obfuscator, then we could obfuscate this Turing machine. Then, the protocol could be run with any number of users.

Unfortunately, the above approach cannot be proven secure. Explain Why

Hint: Go through the proof seen in class. How big must the obfuscated program be to make the proof go through?

Problem 4. Construct Multiparty Non-interactive Key Exchange (NIKE) using Universal Samplers and Public Key Encryption (PKE). Prove that your construction is secure.

Hint: Each user's secret value will be the secret decryption key for a PKE, the public value will be the corresponding public encryption key. The trusted setup will involve generating the parameters for a universal sampler.

This gives an alternative approach to building multiparty NIKE than the construction seen in class.