Program Obfuscation II: The Puncturing Paradigm

1 IO is a "Best-Possible" Obfuscation

Assume that in the future someone discovers an obfuscator $\mathcal{O}'$ which has better hiding properties, e.g. given $\mathcal{O}'(C)$, it is hard for any PPT adversary to predict $\pi(C)$ for a class of predicates π. Assume that $\mathcal{O}'$ has overhead $q(\cdot)$ for some polynomial q in the sense that $|\mathcal{O}'(C)| \leq q(|C|)$.

We will show that $\mathcal{O}(\mathsf{pad}_q(C))$, where $\mathcal{O}$ is an indistinguishability obfuscator, has the *same* hiding properties where $\mathsf{pad}_q(C)$ makes the circuit C artificially bigger, of size $q(|C|)$. Assume, for contradiction, that there is an adversary $\mathcal{A}$ such that

$$\Pr[\mathcal{A}(\mathcal{O}(\mathsf{pad}_q(C))) = \pi(C)] \geq 1/2 + 1/p(\lambda).$$

Since $\mathsf{pad}_q(C)$ and $\mathcal{O}'(C)$ compute the same function and are of the same size,

$$\Pr[\mathcal{A}(\mathcal{O}(\mathcal{O}'(C))) = \pi(C)] \geq 1/2 + 1/p(\lambda) - \mathrm{negl}(\lambda)$$

by the indistinguishability property of $\mathcal{O}$. This gives us an adversary who contradicts the claimed security of $\mathcal{O}'$. The adversary $\mathcal{A}'$, on input $\mathcal{O}'(C)$, simply runs $\mathcal{A}$ on $\mathcal{O}(\mathcal{O}'(C))$, giving us

$$\Pr[\mathcal{A}'(\mathcal{O}'(C)) = \pi(C)] \geq 1/2 + 1/p(\lambda) - \mathrm{negl}(\lambda)\,,$$

contradicting the claimed security of $\mathcal{O}'$.

1.1 An Application: Time-lock Puzzles

A time-lock puzzle is defined by a puzzle generation algorithm Gen which outputs a puzzle k together with its solution s. We have an Eval algorithm that is given k, runs in *sequential* time T and outputs s. We require that no PPT machine that runs in *sequential time* significantly less than T should be able to output s given k, even if it can do arbitrary polynomial-time parallel computation $\gg T$.

A classical example is the Rivest-Shamir-Wagner construction where the puzzle is (g, x, N): $N = pq$ is an RSA modulus; g is a random element of $\mathbb{Z}_N^*$, and $x \in_R [0, N]$ is randomly chosen. The solution is $s = g^{x^T} \bmod N$. This can be computed in time proportional to T (think of T as being much larger than $\mathsf{poly}(\log N)$, the time to do basic operations in the group). The point is, s is easy to compute given the factorization of N which in particular tells us $\varphi(N)$. Given this, first compute $y = x^T \bmod \varphi(N)$ and then raise g to the power of y mod N. However, given only N, and assuming factoring is hard, it appears that any strategy would require computing iteratively $g^{x^i} \bmod N$ for $i = 1, 2, \ldots, T$.

Are there different constructions? In particular, constructions that don't have to make the seemingly ad-hoc assumption that exponentiation is inherently sequential?

A Construction using IO. If you think about it a bit, you realize that one has to assume *something*. i.e. that there exist computations that are inherently sequential. More formally, there is a Turing machine M_T such that on input x, computing $M_T(x)$ requires *sequential time* T. In other words, we have to assume that not all computations are parallelizable.

Consider a dummy Turing machine $M'_{T,s}$ that does nothing for T time steps, and eventually outputs the string s. The description of the puzzle is the IO obfuscation $k = \mathcal{O}(M'_{T,s})$. Here, we assume that one can in fact obfuscate Turing machines in a way that the size of the obfuscated TM does not depend on the runtime

of the TM. This can indeed be done assuming IO for circuits; see Canetti-Jain-Holmgren-Vaikuntanathan STOC 2015.

We claim that no $T' \ll T$ time algorithm can learn s given the puzzle k. Suppose, for contradiciton, that there is an algorithm that does this. Consider running the algorithm on input $\mathcal{O}(M''_{T,s})$ where $M''_{T,s}$ runs the inherently sequential machine M_T for T time steps, outputs s if M_T outputs 1, and outputs $\perp$ if M_T outputs 0.

On the one hand, if M_T outputs 1, $M''_{T,s}$ and $M'_{T,s}$ are functionally equivalent, so the adversary outputs s in this case, in time $T' \ll T$. On the other hand, if M_T outputs 0, $M''_{T,s}$ is functionally equivalent to the machine that always outputs $\perp$. Here, the adversary has information-theoretically no chance of outputting s in polynomial-time. Thus, being able to output s is a distinguisher between the cases that M_T outputs 1 and where it outputs 0. In summary, an adversary that breaks the time-lock puzzle gives us an algorithm to compute M_T in sequential time $T' \ll T$, breaking the assumption that M_T takes sequential time T to compute.

2 IO Engineering I: Using Puncturable PRFs

Puncturable pseudorandom functions (PPRFs) are extremely useful in constructions using IO. We define them below and show several applications. Informally, PPRFs are PRFs where you can release keys that allows one to compute values of the PRF at all inputs except for one "punctured" point.

Definition 1 (Puncturable PRFs)**.** *A puncturable family of PRFs is given by a triple of uniform PPT machines* $(\mathsf{PRFGen}, \mathsf{Punc}, \mathsf{PRF})$*, and a pair of computable functions* $n(\cdot)$ *and* $m(\cdot)$*, satisfying the following conditions:*

Correctness. *For all outputs* K *of* $\mathsf{PRFGen}(1^\lambda)$*, all points* $i \in \{0,1\}^{n(\lambda)}$*, and* $K\{i\} = \mathsf{Punc}(K, i)$*, we have that*

$$\mathsf{PRF}(K\{i\}, x) = \mathsf{PRF}(K, x)$$

for all $x \neq i$*.*

Pseudorandom at punctured point. *For every p.p.t. adversary* $(\mathcal{A}_1, \mathcal{A}_2)$*, there is a negligible function* μ*, such that in an experiment where* $\mathcal{A}_1(1^\lambda)$ *outputs a point* $i \in \{0,1\}^{n(\lambda)}$ *and a state* σ*,* $K \leftarrow \mathsf{PRFGen}(1^\lambda)$ *and* $K\{i\} = \mathsf{Punc}(K, i)$*, the following holds*

$$\left|\Pr[\mathcal{A}_2(\sigma, K\{i\}, i, \mathsf{PRF}(K, i)) = 1] - \Pr[\mathcal{A}_2(\sigma, K\{i\}, i, U_{m(\lambda)}) = 1]\right| \leq \mu(\lambda)$$

where μ *is called the* distinguishing gap *for* $(\mathcal{A}_1, \mathcal{A}_2)$*.*

Furthermore, we say that the puncturable PRF is δ-indistinguishable *if the above pseudorandom property holds with a distinguishing gap* μ *bounded by* δ*. Especially, the puncturable PRF is* sub-exponentially indistinguishable *if* $\mu(\lambda)$ *is bounded by* $2^{-\lambda^\epsilon}$ *for a constant* ϵ*.*

The GGM tree-based construction of PRFs from pseudorandom generators (PRGs) yields puncturable PRFs for any (efficiently computable) $n(\cdot)$ and $m(\cdot)$. Furthermore, it is easy to see that if the PRG underlying the GGM construction is sub-exponentially hard (and this can in turn be built from sub-exponentially hard OWFs), then the resulting puncturable PRF is sub-exponentially pseudo-random.

2.1 Application: A Different Public-Key Encryption Scheme from IO

This is (likely?) the construction that Diffie and Hellman had in mind. The idea is to start from a secret-key encryption scheme, obfuscate its encryption algorithm, and publish the obfuscated program as the public key. More precisely, the program to be obfuscated contains the secret key sk, takes in a message m and a random string r, and outputs $\mathsf{SKEnc}(\mathsf{sk}, m; r)$. However, this has to be done with care.

As a first try, let us take the simple secret-key encryption scheme where the secret key sk is the seed of a pseudorandom function (PRF), and the encryption algorithm evaluates the PRF on a random string and uses it as a one-time pad for the message. That is,

$$\mathsf{SKEnc}(\mathsf{sk}, m; r) = (r, \mathsf{PRF}(\mathsf{sk}, r) \oplus m) .$$

The public-key now is an obfuscated program that takes (m, r) as input and outputs $\mathsf{SKEnc}(\mathsf{sk}, m; r)$. Is the resulting public-key encryption scheme secure?

The construction that actually works is a clever twist on this, and is due to Sahai and Waters (Proceedings of ACM STOC 2014).

The Construction. Let $\mathcal{O}$ be an indistinguishability obfuscator, $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a pseudorandom generator, and $(\mathsf{PRFGen}, \mathsf{Punc}, \mathsf{PRF})$ be a puncturable PRF family. The public-key encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ works as follows.

- $\mathsf{KeyGen}(1^n)$: The secret key is a uniformly random string $K \leftarrow \{0,1\}^\lambda$. Let Π_K be the following program that takes as input $m \in \{0,1\}^{2n}$ and $r \in \{0,1\}^n$.

  > $\Pi_K(m, r)$ **does the following:**
  >
  > - Compute $y = G(r)$.
  > - Output $(y, \mathsf{PRF}(K, y) \oplus m)$.

  The public key is an obfuscation of Π_K and the secret key is the PRF key K. That is,

  $$\mathsf{pk} = \widehat{\Pi}_K = \mathcal{O}(\Pi_K) \quad \text{and} \quad \mathsf{sk} = K.$$

- $\mathsf{Enc}(\mathsf{pk}, m)$: Sample a random string $r \leftarrow \{0,1\}^n$ and output $\widehat{\Pi}_K(r, m)$.
- $\mathsf{Dec}(\mathsf{sk}, c)$: Run the secret-key decryption algorithm. That is, interpret c as a pair (c_0, c_1) and output $c_1 \oplus \mathsf{PRF}(K, c_0)$ as the message.

Theorem 2. $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a semantically secure public-key encryption scheme.*

Proof. Correctness is immediate. We now show semantic security.

Hybrid 0. The real distribution of $\mathsf{pk} = \mathcal{O}(\Pi_K)$ and $\mathsf{Enc}(\mathsf{pk}, m^*) = (y^* = G(r^*), \mathsf{PRF}(K, y^*) \oplus m^*)$ for $r^* \leftarrow \{0,1\}^n$.

Hybrid 1. We change the public key from $\mathsf{pk} = \mathcal{O}(\Pi_K)$ to $\mathsf{pk} = \mathcal{O}\left(\Pi'_{y^*,K\{y^*\},\mathsf{PRF}(K,y^*)}\right)$, where $\Pi'_{y^*,K\{y^*\},z^*}(m,r)$ is the following program, taking as input $m \in \{0,1\}^{2n}$ and $r \in \{0,1\}^n$.

$\Pi'_{y^*,K\{y^*\},z^*}(m,r)$:

- Compute $y = G(r)$.
- If $y = y^*$, output $(y, z^* \oplus m)$.
- Else, output $(y, \mathsf{PRF}(K\{y^*\}, y) \oplus m)$.

Since we hard-code $z^* = \mathsf{PRF}(K, y^*)$ into the program, $\Pi'_{y^*,K\{y^*\},\mathsf{PRF}(K,y^*)}$ has the exact same functionality as Π_K, and thus we can invoke IO security (as well as PPRF correctness) to argue indistinguishability from hybrid 0.

Hybrid 2. We substitute $y^* = G(r^*)$ for $y^* \leftarrow \{0,1\}^{2n}$ in both the public key and ciphertext.

This is indistinguishable from hybrid 1 by security of the PRG. (The only reference to r^* is via $y^* = G(r^*)$, so this is valid.)

Hybrid 3. We change the public key into the obfuscation of the following program $\Pi''_{y^*,K\{y^*\}}$, taking as input $m \in \{0,1\}^{2n}$ and $r \in \{0,1\}^n$.

$\Pi''_{y^*,K\{y^*\}}(m,r)$:

- Compute $y = G(r)$.
- If $y = y^*$, output $\perp$.
- Else, output $(y, \mathsf{PRF}(K\{y^*\}, y) \oplus m)$.

The only difference in functionality between $\Pi''_{y^*,K\{y^*\}}$ and $\Pi'_{y^*,K\{y^*\},\mathsf{PRF}(K,y^*)}$ is in the case when $G(r) = y^*$. However, since G is length-doubling and $y^* \leftarrow \{0,1\}^{2n}$ is uniformly random, the probability that there exists *any* $r \in \{0,1\}^n$ such that $G(r) = y^*$ is at most 2^{-n}. Therefore, with probability at least $1 - 2^{-n}$ over y^*, the check "$G(r) \stackrel{?}{=} y^*$?" will never pass on any input r. Thus, with probability at least $1 - 2^{-n}$, $\Pi''_{y^*,K\{y^*\}}$ and $\Pi'_{y^*,K\{y^*\},\mathsf{PRF}(K,y^*)}$ are functionally identical, allowing us to invoke IO security to argue indistinguishability from hybrid 2.

Hybrid 4. In the ciphertext, we replace $\mathsf{PRF}(K, y^*) \oplus m^*$ with a uniformly random string $c^* \leftarrow \{0,1\}^{2n}$.

Since the public key now depends only on $K\{y^*\}$ and not K, by PPRF security, the value $\mathsf{PRF}(K, y^*)$ is indistinguishable from random, and by a one-time pad argument, the same holds for c^*.

Hybrid 4 is independent of the message m^*, so we are done with the proof.

□

3 When IO Alone is Not Enough

3.1 Fully Homomorphic Encryption

A natural idea is to obfuscate a program that, on input any two ciphertexts C_0 and C_1 encrypting bits b_0 and b_1 respectively, decrypts them using a hardcoded secret-key SK, computes the NAND function on b_0 and b_1 and re-encrypts them under PK. That is,

$$\Pi_{\mathsf{PK},\mathsf{SK}}(C_0, C_1) = \mathsf{Enc}\Big(\mathsf{PK}, \mathsf{Dec}(\mathsf{SK}, C_0)\ \mathsf{NAND}\ \mathsf{Dec}(\mathsf{SK}, C_1); r\Big)$$

An immediate question is where the randomness r for Enc comes from. It clearly shouldn't be hardcoded into Π; neither should it be left up to the adversary to input the randomness into Π. The only option left is to hardcode a PRF key K into Π, and apply it to the input (C_0, C_1) to generate r. That is,

$$\Pi_{\mathsf{PK},\mathsf{SK},K}(C_0, C_1) = \mathsf{Enc}\Big(\mathsf{PK}, \overline{\mathsf{Dec}(\mathsf{SK}, C_0)\ \wedge \mathsf{Dec}(\mathsf{SK}, C_1)}; \mathsf{PRF}(K, (C_0, C_1))\Big)$$

This is indeed the idea that will work out for us. In fact, it is not hard to see that if $\mathcal{O}$ is a VBB obfuscation scheme, this construction gives us a secure FHE scheme. However, all we have is IO, and that requires quite a bit of care.

It turns out that any IO scheme plus a *perfectly rerandomizable encryption scheme* implies FHE. Both the construction and the proof is instructive and offers another technique to use IO in a world where IO and one-way functions together are not enough.

The Construction. Let's consider where the natural construction of obfuscating the circuit $\Pi_{\mathsf{PK},\mathsf{SK},K}$ as above and releasing it publicly as the homomorphic evaluation key goes wrong. We'd like to argue that the evaluation key is indistinguishable from the obfuscation of a circuit

$$\Pi'_{\mathsf{PK},K}(C_0, C_1) = \mathsf{Enc}\Big(\mathsf{PK}, 0; \mathsf{PRF}(K, (C_0, C_1))\Big)$$

which simply ignores the input (except to generate the encryption randomness) and outputs an encryption of 0. Note that this circuit does not use SK at all.

We will also make one more change: let the input and output encryption schemes use distinct keys. That is, we will have programs

$$\Pi_{\mathsf{PK}_i,\mathsf{SK}_{i-1},K_i}(C_0, C_1) = \mathsf{Enc}\Big(\mathsf{PK}_i, \overline{\mathsf{Dec}(\mathsf{SK}_{i-1}, C_0)\ \wedge \mathsf{Dec}(\mathsf{SK}_{i-1}, C_1)}; \mathsf{PRF}(K_i, (C_0, C_1))\Big)$$

for $i = 0, \ldots, D-1$ to evaluate circuits of depth up to D. We will show that this sequence of obfuscated programs is indistinguishable from the sequence of programs

$$\Pi'_{\mathsf{PK}_i,K_i}(C_0, C_1) = \mathsf{Enc}\Big(\mathsf{PK}_i, 0; \mathsf{PRF}(K_i, (C_0, C_1))\Big)$$

That is, we will show that

$$\Big(\mathcal{O}(\Pi_{\mathsf{PK}_1,\mathsf{SK}_0,K_1}), \ldots, \mathcal{O}(\Pi_{\mathsf{PK}_D,\mathsf{SK}_{D-1},K_D})\Big)_{i=0,\ldots,D-1} \approx_c \big(\mathcal{O}(\Pi'_{\mathsf{PK}_1,K_1}), \ldots, \mathcal{O}(\Pi'_{\mathsf{PK}_D,K_D})\big)_{i=0,\ldots,D-1} \tag{1}$$

We will do this in a series of hybrids starting with replacing $\mathcal{O}(\Pi_{\mathsf{PK}_D,\mathsf{SK}_{D-1},K_D})$ in the LHS by $\mathcal{O}(\Pi'_{\mathsf{PK}_D,K_D})$, leaving everything else in tact. (Do you see why this is the right order to do the hybrids?)

OK, so how does one show that given SK_{D-1}, an obfuscation of $\mathcal{O}(\Pi_{\mathsf{PK}_D,\mathsf{SK}_{D-1},K_D})$ is indistinguishable from an obfuscation of $\mathcal{O}(\Pi'_{\mathsf{PK}_D,K_D})$? We again go through hybrids and a puncturing argument, this time

as many hybrids as the number of possible inputs to either circuit. Note that this is exponentially many! Let's number the inputs from 0 to $\ell := 2^{|C|+1} - 1$ where $|C|$ is the bit-length of a ciphertext.

Hybrid 0. Define $\Pi^{(0)} := \Pi_{\mathsf{PK}_D,\mathsf{SK}_{D-1},K_D}$. The adversary sees SK_{D-1} and $\mathcal{O}(\Pi^{(0)})$.

...

Hybrid i. Define $\Pi^{(i)}$ to be the program that on input (C_0, C_1), checks if it is less than i. If yes, output $\Pi'_{\mathsf{PK}_D,K_D}(C_0, C_1)$, else output $\Pi_{\mathsf{PK}_D,\mathsf{SK}_{D-1},K_D}(C_0, C_1)$. The adversary sees SK_{D-1} and $\mathcal{O}(\Pi^{(i)})$.

...

Hybrid $\ell + 1$. Define $\Pi^{(\ell+1)}$ to be the program that on input (C_0, C_1), checks if it is less than $\ell + 1$. If yes, output $\Pi'_{\mathsf{PK}_D,K_D}(C_0, C_1)$, else output $\Pi_{\mathsf{PK}_D,\mathsf{SK}_{D-1},K_D}(C_0, C_1)$. The adversary sees SK_{D-1} and $\mathcal{O}(\Pi^{(\ell+1)})$.

Note that in the first hybrid, the adversary sees an obfuscation of $\Pi_{\mathsf{PK}_D,\mathsf{SK}_{D-1},K_D}$ and in the last, she sees an obfuscation of Π'_{PK_D,K_D}. If an adversary can distinguish between the first and last hybrids with advantage ϵ, she must be able to distinguish between some two adjacent hybrids with advantage at least $\epsilon/2^\ell$. We will (try to) show that this is impossible invoking the security of (a) the IO scheme, (b) the encryption scheme and (c) the PRF.

The Problem. The first step is to replace the PRF key K with a key punctured at input i; this is OK by IO. The next step is to replace the value of the PRF at i with a truly random string; this is OK by the puncturing security of the PRF. Now, the output of the programs on input i is either

$$\mathsf{Enc}_{\mathsf{PK}_i}(\mathsf{Dec}_{\mathsf{SK}_{i-1}}(C_0) \text{ NAND } \mathsf{Dec}_{\mathsf{SK}_{i-1}}(C_1); r) \quad \text{or} \quad \mathsf{Enc}_{\mathsf{PK}_i}(0; r)\,,$$

in the two circuits respectively. In the last step, we want to switch from the first to the second of these ciphertexts using the semantic security of the encryption scheme.

In each of these steps, one uses the subexponential security of the underlying primitive: IO, PRF, or the encryption scheme. The first two are OK by increasing the key sizes and the security parameters sufficiently, but we run into trouble with the encryption scheme. Recall that we will get a distinguisher for the encryption scheme with advantage $1/(\mathsf{poly}(\lambda) \cdot 2^{|\mathsf{ct}|+1})$ where ct is a ciphertext in the scheme. However, *any encryption scheme can be broken with such small advantage*! (Do you see how?)

The Fix. We will rely on an additional primitive: a perfectly lossy encryption scheme. This is an encryption scheme with two types of keys: regular keys PK that come with a matching secret decryption key SK; and lossy keys $\widetilde{\mathsf{PK}}$ where a random ciphertext of 0 is *identically distributed* to a random ciphertext of 1. Moreover, a random PK and a random $\widetilde{\mathsf{PK}}$ are computationally indistinguishable.

Armed with a lossy encryption scheme, we proceed as follows. We change the big hybrids (see equation 1) so that after every step i, we change the public keys from real to lossy. That is, while before, we were trying to go from an obfuscation of $\mathcal{O}(\Pi_{\mathsf{PK}_D,\mathsf{SK}_{D-1},K_D})$ to an obfuscation of $\mathcal{O}(\Pi'_{\mathsf{PK}_D,K_D})$ directly, we insert two hybrids in between, namely $\mathcal{O}(\Pi_{\widetilde{\mathsf{PK}}_D,\mathsf{SK}_{D-1},K_D})$ and $\mathcal{O}(\Pi'_{\widetilde{\mathsf{PK}}_D,K_D})$. We then proceed as follows.

$\mathcal{O}(\Pi_{\mathsf{PK}_D,\mathsf{SK}_{D-1},K_D}) \approx \mathcal{O}(\Pi_{\widetilde{\mathsf{PK}}_D,\mathsf{SK}_{D-1},K_D})$. This follows from the indistinguishability of lossy and real public keys.

$\mathcal{O}(\Pi'_{\mathsf{PK}_D,K_D}) \approx \mathcal{O}(\Pi'_{\widetilde{\mathsf{PK}}_D,K_D})$. This follows from the indistinguishability of lossy and real public keys.

$\mathcal{O}(\Pi_{\widetilde{\mathsf{PK}}_D,\mathsf{SK}_{D-1},K_D}) \approx \mathcal{O}(\Pi'_{\widetilde{\mathsf{PK}}_D,K_D})$. Here, we follow the hybrid argument as above except we note that in the step where one invokes the IND-CPA security of encryption, we have *perfect* security! Thus, a distinguisher breaking the indistinguishability of the two corresponding hybrids with *any non-zero* advantage gives us a contradiction.

This observation finishes the construction and the proof.

Remark One might wonder if the lossy encryption scheme in the construction is necessary; turns out it is. Bitansky, Degwekar and Vaikuntanathan constructed an oracle world where IO and OWF (even OWP) exist, but where $\mathbf{SZK} = \mathbf{P}$. Since every FHE scheme can be broken in $\mathbf{SZK}$, it turns out that in this oracle world, FHE does not exist. Therefore, there are no *black-box* FHE constructions from IO and OWP. (The notion of black-boxness has to be formulated carefully to allow obfuscating programs that invoke the OWF/OWP; for more details, we refer the reader to the paper of Bitansky, Degwekar and Vaikuntanathan.)

4 IO Engineering II: Using (Zero Knowledge) Proofs

4.1 Bootstrapping IO: From Shallow Circuits to All Circuits

There are many ways to construct an IO for all poly-size circuits starting from an IO for relatively simple complexity classes. We start by describing perhaps the simplest of these, which also gives us more practice on how to use IO.

From Fully Homomorphic Encryption. We can show that if there is IO for NC^1 circuits and an FHE scheme whose decryption can be done in NC^1, then there is an IO for all of P.

The idea is to encrypt the circuit C using an FHE key into $\widehat{C} = \mathsf{FHE.Enc}(\mathsf{sk}, C)$ so that on any input x, one can homomorphically compute $\widehat{C(x)} \in \mathsf{FHE.Enc}(\mathsf{sk}, C(x))$. The question is how to recover $C(x)$. For that purpose, the first, and somewhat naïve, idea is to provide an obfuscation of the following program P_{sk}.

$$\text{On input } y\text{: Output } \mathsf{FHE.Dec}(\mathsf{sk}, y).$$

Clearly, this is insecure as $\mathcal{O}(P_{\mathsf{sk}})$ can be used to decrypt $\widehat{C}$ and recover the original circuit C.

A better idea is to obfuscate the "check-then-decrypt" program $P_{\mathsf{sk},c}$ which has the secret key sk and the ciphertext $c = \widehat{C}$ encoded, and takes a ciphertext y, an input x, and a "proof" π as input. If the proof checks out, it decrypts y. Importantly for us, the proof will simply be the transcript of the homomorphic computation that takes $\widehat{C}$ to $\widehat{C(x)}$, and verifying it can be done in parallel, i.e. in NC^1. So:

$$\text{On input } (y, \pi)\text{: Check the proof } \pi\text{; If it fails, output } \perp\text{, else output } \mathsf{FHE.Dec}(\mathsf{sk}, y).$$

This "should" work but it is unclear how to prove it secure using the IO guarantee.

Instead, we resort to the two-track trick ala Naor-Yung. In other words, the obfuscation of C is a pair of ciphertexts

$$c_0 = \mathsf{FHE.Enc}(\mathsf{pk}_0, C) \text{ and } c_1 = \mathsf{FHE.Enc}(\mathsf{pk}_1, C)$$

and a program that, on input c_0', c_1', x, π as input, checks the proof, and if the proof verifies, decrypts c_0' using sk_0.

This turns out to do the trick. To see why, we construct a sequence of hybrids.

Hybrid 1. c_0 and c_1 as above and $\mathcal{O}(P_{\mathsf{sk}_0, c_0, c_1})$.

Hybrid 2. Change c_1 to be an encryption of C_1 under sk_1. OK by semantic security since sk_1 is never used in these hybrids.

Hybrid 3. Change the program to be an obfuscation of $\mathcal{O}(P_{\mathsf{sk}_1,c_0,c_1})$ which decrypts using sk_1. These two programs are functionally equivalent by the perfect correctness of FHE evaluation and the functional equivalence of C_0 and C_1. Thus, IO guarantees that hybrids 2 and 3 are computationally indistinguishable.

Hybrid 4. Change c_0 to be an encryption of C_1 under sk_0. OK by semantic security since sk_0 is never used in these hybrids.

Hybrid 5. Once again, change the program to be an obfuscation of $\mathcal{O}(P_{\mathsf{sk}_0,c_0,c_1})$ which decrypts using sk_0.

The distribution in hybrid 5 is that of obfuscating program C_1. This finishes the proof.

4.2 (Simulation-Sound) NIZK Proofs from NIZK Proofs

Achieving Statistical Simulation-Soundness. We use the following standard compiler from any NIZK into one that is statistically simulation-sound. The CRS consists of the original CRS (of an underlying NIZK system) together with a commitment to 0, i.e. $c = \mathsf{Com}(0^n; r)$ where 0^n is a special string that cannot be a plausible NP statement. The prover, given (x, w) shows that either $x \in L$ or $\exists r$ such that $c = \mathsf{Com}(x; r)$. Soundness follows from that of the underlying NIZK together with the fact that c is not a commitment of any plausible NP statement. The zero knowledge simulator sets c to be a commitment of x^* and uses the corresponding randomness to generate a simulated proof. Even given such a proof, there do not exist accepting proofs of any $x' \notin R_L$, $x' \neq x^*$, by the statistical soundness of the underlying NIZK system and the perfect binding of the commitment scheme.

NIZK proofs in turn can be constructed from several algebraic assumptions including quadratic residuosity, LWE, factoring, decisional Diffie-Hellman and more.

4.3 Functional Encryption from IO

First, we define functional encryption. Informally, functional encryption allows the release of "partial" decryption keys, which allows one to reveal specified functions f of the underlying plain-text.

Definition 3 (Functional Encryption)**.** *A functional encryption scheme is given by a tuple of PPT machines* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *satisfying the following conditions:*

Correctness. *For all polynomial-size circuits f and inputs x,*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, \mathsf{Enc}(\mathsf{mpk}, x)) = f(x)\right] = 1,$$

where $(\mathsf{mpk}, msk) \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(msk, f)$.

Indistinguishable Security (Adaptive). *For every (stateful) PPT adversary $\mathcal{A}$, there is a negligible function $\mu(\cdot)$ and polynomially-bounded $m_1(\cdot)$ and $m_2(\cdot)$ such that the adversary has advantage at most $\mu(\lambda)$ in the experiment $\left\{\mathsf{Exp}_b^{\mathcal{A}}\left(1^\lambda\right)\right\}_{b\in\{0,1\}}$ defined as follows:*

1. *The challenger runs* $(\mathsf{mpk}, msk) \leftarrow \mathsf{Setup}(1^\lambda)$ *and sends* mpk *to* $\mathcal{A}$.
2. *Adversary $\mathcal{A}$ sequentially and adaptively chooses circuits f_i to send to the challenger, and the challenger sends back* $\mathsf{sk}_{f_i} \leftarrow \mathsf{KeyGen}(msk, f_i)$ *for each* $i \in [m_1(\lambda)]$.
3. *Adversary $\mathcal{A}$ sends two inputs x_0, x_1 to the challenger such that $f_i(x_0) = f_i(x_1)$ for all $i \in [m_1(\lambda)]$.*
4. *The challenger samples* $c \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_b)$ *and sends c to $\mathcal{A}$.*

5. *Adversary $\mathcal{A}$ sequentially and adaptively chooses more circuits g_i to send to the challenger, and the challenger sends back $\mathsf{sk}_{g_i} \leftarrow \mathsf{KeyGen}(msk, g_i)$ for all $i \in [m_2(\lambda)]$, as long as $g_i(x_0) = g_i(x_1)$ for all $i \in [m_2(\lambda)]$.*
6. *Adversary $\mathcal{A}$ concludes by sending some bit $b' \in \{0,1\}$ to the challenger. This bit b' is defined as the output of the experiment.*

We define the advantage *of $\mathcal{A}$ to be*

$$\left|\Pr\left[\mathsf{Exp}_0^{\mathcal{A}}\left(1^\lambda\right) = 1\right] - \Pr\left[\mathsf{Exp}_1^{\mathcal{A}}\left(1^\lambda\right) = 1\right]\right|,$$

which we require to be at most $\mu(\lambda)$.

One may define a weaker notion of security called selective *(indistinguishable) security, where the adversary's challenge inputs are declared at the very beginning of the experiment (see e.g. [?] in comparing selective and adaptive security). Also note that a "master decryption key" can be released by obtaining the decryption key for the identity function.*

A first idea for a construction of functional encryption from IO is to let the functional key of a circuit f (represented as a string) be a digital signature of f using the master secret key msk as the signing key. An encryption of a message m is the obfuscation of the program $P_{m,\mathsf{mpk}}$ that takes as input (f, σ) and checks if σ is a valid signature of f under the verification key mpk. If yes, the program outputs $f(m)$; otherwise, the program outputs $\perp$. While this is a natural idea, it is unclear how to make it work using IO. The reason, roughly speaking, is that IO is useful when one can perhaps "puncture the verification key" in some way to ensure that some f's do not have any valid signature.

Open Problem 4. *Can the above idea go through, perhaps by constructing an appropriate puncturable signature scheme?*

While one may be able to operationalize the above idea, we take a different route here. Assume the existence of statistically simulation-sound NIZK proof system and a CPA encryption scheme. The construction works as follows.

1. The setup algorithm generates two public keys $\mathsf{PK}_0, \mathsf{PK}_1$ and the matching private keys $\mathsf{SK}_0, \mathsf{SK}_1$ of a CPA-secure encryption scheme, as well as a common (or structured) random string CRS of a statistically simulation-sound NIZK proof system. The setup is then

$$\mathsf{MPK} = (\mathsf{PK}_0, \mathsf{PK}_1, \mathsf{CRS}) \text{ and } \mathsf{MSK} = (\mathsf{SK}_0, \mathsf{SK}_1).$$

2. The encryption algorithm, given x, encrypts x with both public keys and attaches a NIZK proof that the two ciphertexts encrypt the same message.

$$C_b \leftarrow \mathsf{Enc}(\mathsf{PK}_b, x; r_b) \text{ and } \pi \leftarrow \mathsf{P}(\mathsf{CRS}, C_0, C_1, x, r_0, r_1)$$

3. The key generation algorithm, given MSK and a circuit f, writes down a program $\Pi_{\mathsf{CRS},\mathsf{SK}_0,f}$ that takes as input (C_0, C_1, π).

$\Pi_{\mathsf{CRS},\mathsf{SK}_0,f}(C_0, C_1, \pi)$ **does the following:**

- Verify π as a proof that C_0 and C_1 encrypt the same message (using CRS).
- If valid, output $f(\mathsf{Dec}(\mathsf{SK}_0, C_0))$.
- Else, output $\perp$.

We then define

$$\mathsf{SK}_f = \mathcal{O}(\Pi_{\mathsf{CRS},\mathsf{SK}_0,f}).$$

4. The decryption algorithm simply feeds the ciphertext into the secret key SK_f which is the obfuscated program.

The (selective security) proof proceeds via the following sequence of hybrids.

Hybrid 0. The adversary declares a challenge (x^*, y^*). Construct C_0^* and C_1^* by encrypting x^* under PK_0 and PK_1 respectively. Let CRS be the real CRS. Produce a real proof π.

Hybrid 1. Replace the common reference string with the simulated CRS^*. Produce a simulated proof π^*. This is indistinguishable by the ZK property of the proof system.

Hybrid 2. Replace C_1^* to be the encryption of y^*. The proof is still simulated. This is indistinguishable by CPA-security since SK_1 is never used.

Hybrid 3. Replace the functional keys with ones that decrypt using SK_1. This is OK because of the IO guarantee and because the circuits that decrypt using SK_0 and SK_1 are identical in functionality. Here, we crucially use the fact that the NIZK proof system is statistically simulation-sound, and that $f(x^*) = f(y^*)$ for every function f for which the adversary obtains the functional key.

Hybrid 4. Replace C_0^* to be the encryption of y^*. The proof is still simulated. This is indistinguishable by CPA-security since SK_0 is never used.

Hybrid 5. Replace the functional keys with ones that decrypt using SK_0. This is OK because of the same argument as in Hybrid 2 vs 3.

Hybrid 6. Replace the common reference string with the real CRS and the proof with the real proof. This is indistinguishable by the ZK property of the proof system.

Hybrid 6 is the real game where y^* is encrypted instead of x^* in Hybrid 0. Since hybrids 0 and 6 are indistinguishable, we are done.