How to Use Indistinguishability Obfuscation: Deniable Encryption, and More

Amit Sahai
UCLA
sahai@cs.ucla.edu

Brent Waters
University of Texas at Austin
bwaters@cs.utexas.edu

Abstract

We introduce a new technique, that we call *punctured programs*, to apply indistinguishability obfuscation towards cryptographic problems. We use this technique to carry out a systematic study of the applicability of indistinguishability obfuscation to a variety of cryptographic goals. Along the way, we resolve the 16-year-old open question of *Deniable Encryption*, posed by Canetti, Dwork, Naor, and Ostrovsky in 1997: In deniable encryption, a sender who is forced to reveal to an adversary both her message and the randomness she used for encrypting it should be able to convincingly provide "fake" randomness that can explain any alternative message that she would like to pretend that she sent. We resolve this question by giving the first construction of deniable encryption that *does not require any pre-planning* by the party that must later issue a denial.

In addition, we show the generality of our punctured programs technique by also constructing a variety of core cryptographic objects from indistinguishability obfuscation and one-way functions (or close variants). In particular we obtain: public key encryption, short "hash-and-sign" selectively secure signatures, chosen-ciphertext secure public key encryption, non-interactive zero knowledge proofs (NIZKs), injective trapdoor functions, and oblivious transfer. These results suggest the possibility of indistinguishability obfuscation becoming a "central hub" for cryptography.

Amit Sahai is supported for this research effort in part from NSF grants 1228984, 1136174, 1118096, 1065276, 0916574 and 0830803, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. The views expressed are those of the author and do not reflect the official policy or position of the National Science Foundation, or the U.S. Government.

Brent Waters is supported for this research effort by NSF CNS-0915361 and CNS-0952692, CNS-1228599, DARPA N11AP20006, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

1 Introduction

In 2001, Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang [BGI+01, BGI+12] initiated the formal study of *program obfuscation*, which aims to make computer programs "unintelligible" while preserving their functionality. They offered two notions of general obfuscation: First, they suggested a quite intuitive notion called *virtual black-box* obfuscation – which asks that an obfuscated program be no more useful than a black box implementing the program. Perhaps unsurprisingly, they showed that this notion of virtual black-box obfuscation would have many immediate and intuitive applications throughout cryptography, including for example, giving a transformation converting a natural private-key encryption scheme into a public-key encryption scheme. Unfortunately, however, they also showed that this notion of general-purpose virtual black-box obfuscation is impossible to achieve. Second, motivated by this impossibility, they also proposed the less intuitive, but potentially realizable, notion of *indistinguishability* obfuscation – which asks only that obfuscations of any two distinct (equal-size) programs that implement *identical* functionalities be computationally indistinguishable from each other. Unfortunately, it has been less clear how useful indistinguishability obfuscation would be. In a recent breakthrough, Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH+13a] proposed the first candidate construction of an efficient indistinguishability obfuscator for general programs (written as boolean circuits), and also showed how to apply indistinguishability obfuscation to achieve powerful new functional encryption schemes for general circuits. We believe the work of [GGH+13a] is likely to lead to significant follow-up research proposing alternative constructions of indistinguishability obfuscators under various other (possibly standard) assumptions, perhaps in a manner similar to research on Fully Homomorphic Encryption [Gen09, BV11, BGV12, Bra12, GSW13].

Despite this significant advance, some of the most basic questions about how to use indistinguishability obfuscation remain open. For example, can we use indistinguishability obfuscation to transform a natural private-key encryption scheme to a public-key encryption scheme, as can be done using virtual black-box obfuscation? In this work, we take a fresh look at the question of how to use indistinguishability obfuscation in general contexts. To this end, we introduce a new technique for applying indistinguishability obfuscation, that we call *punctured programs*. We use this technique to carry out a systematic study of the applicability of indistinguishability obfuscation to a variety of cryptographic goals. Along the way, we resolve the 16-year-old open question of *Deniable Encryption*, posed by Canetti, Dwork, Naor, and Ostrovsky in 1997 [CDNO97]: In deniable encryption, a sender who is forced to reveal to an adversary both her message and the randomness she used for encrypting it should be able to convincingly provide "fake" randomness that can explain any alternative message that she would like to pretend that she sent. We resolve this question by giving the first construction of deniable encryption that *does not require any pre-planning* by the party that must later issue a denial.

Punctured Programs. At the heart of our results is a new technique for applying indistinguishability obfuscation that we call *punctured programs*: At a very high-level, the idea of the technique is to alter a program (which is to be obfuscated) by surgically removing a key element of the program, without which the adversary cannot win the security game it must play, but in a way that does not alter the functionality of the program. This is best illustrated by means of an example:

Consider the problem of transforming a natural private-key encryption scheme into a public-key encryption scheme, by means of obfuscation. This idea was first proposed by Diffie and Hellman in their original paper on public-key cryptography [DH76]. For example, consider the following simple and natural private-key encryption scheme: the secret key is the key K to a pseudo-random function (PRF). To encrypt a message m, the sender picks a random string r, and outputs $(r, \mathrm{PRF}(K, r) \oplus m)$.

How can we use indistinguishability obfuscation to convert this encryption scheme into a public-key scheme? Intuitively speaking, security requires that an adversary who is given a challenge ciphertext $c^* = (r^*, e^*)$ encrypting some message m^*, should not be able to recover m^*. As a warmup, consider the following (flawed) solution idea: Let a public key simply be an indistinguishability obfuscation of the encryption function itself, namely the function:

$$f_K(r, m) = \big(r, \mathrm{PRF}(K, r) \oplus m\big)$$

The central idea behind our approach is to observe that if it were possible to build the function f_K using a "punctured" PRF key which gave no information about $\mathrm{PRF}(K, r^*)$, but correctly defined the PRF at all other strings $r \neq r^*$, then even if the adversary *knew* this punctured PRF secret key, it would not be able to break the security of the challenge ciphertext. In fact, it is possible to construct such punctured PRF secret keys, for the original GGM construction of PRFs [GGM84]. This was recently observed independently by [BW13, BGI13, KPTZ13].

All that remains is to argue that we can replace the obfuscation of the original function $f_K(r, m) = (r, \mathrm{PRF}(K, r) \oplus m)$ with the obfuscation of the punctured function, in a manner that is indistinguishable to the adversary. However, this is in fact false: Indeed, there is a trivial attack on this suggestion because given the challenge ciphertext (r^*, e^*), the adversary can simply query the obfuscated program on the input $(r^*, 0)$ and thereby learn $\mathrm{PRF}(K, r^*)$ and recover the message m^* from the challenge ciphertext. Thus, we can never hope to swap the obfuscation of the original function with the punctured function, because the adversary can *always* tell the difference. Indeed, an indistinguishability obfuscator guarantees indistinguishable obfuscated programs only when given two *functionally equivalent* input programs. However, as we just saw, the punctured program is clearly not equivalent to the original program, precisely because of the puncturing.

This brings us to the second main idea behind the punctured programs technique, which is crucial to hide where the puncturing takes place and allow the use of indistinguishability obfuscation: The idea is to find a way to "move" the puncturing to a location that is *never* functionally accessed by the program. Let us illustrate by returning to the example of building a public-key encryption scheme from a natural private-key encryption scheme. Consider a pseudo-random generator PRG that maps λ bits to 2λ bits, where λ is a security parameter. Now consider the following modified private-key encryption function:

$$f_K(r, m) = \Big(\mathrm{PRG}(r), \mathrm{PRF}\Big(K, \mathrm{PRG}(r)\Big) \oplus m\Big)$$

As before, let the public key be an indistinguishability obfuscation of this function. Now, however, a challenge ciphertext will look like $c^* = (\mathrm{PRG}(r^*), \mathrm{PRF}(K, \mathrm{PRG}(r^*)) \oplus m^*)$. Since a PRG is keyless, however, the adversary cannot distinguish the original security game in which the challenge ciphertext was created as above, and a hybrid experiment where the challenge ciphertext is created with a freshly chosen random $\tilde{r} \in \{0,1\}^{2\lambda}$, as follows: $c^* = (\tilde{r}, \mathrm{PRF}(K, \tilde{r}) \oplus m^*)$. However, now we observe that with overwhelming probability, $\tilde{r}$ is outside the image of the PRG. Thus, crucially, the encryption function f_K never makes use of the PRF evaluated at $\tilde{r}$.

Thus, indistinguishability obfuscation guarantees that an obfuscation of an alternative program that uses a punctured PRF key that carves out $\tilde{r}$ is indistinguishable from the obfuscation of the original program, because these two programs are functionally equivalent. Now, due to the puncturing, the adversary simply does not have enough information to distinguish $\mathrm{PRF}(K, \tilde{r})$ from random, and thus security follows.

This example illustrates the core ideas behind the punctured programs technique. We use this technique to use indistinguishability obfuscation to build several cryptographic primitives, including short signatures, CCA-secure encryption, and perfect non-interactive zero-knowledge arguments for NP. Most notably, we use this technique to solve the open question of deniable encryption.

On the need for additional assumptions. Before moving on to deniable encryption, we note that our construction of public-key encryption above required not only indistinguishability obfuscation, but also the assumption that one-way functions exist. It is natural to wonder if this extra assumption of one-way functions is needed. We observe that this indeed seems necessary: If P=NP, then certainly one-way functions would not exist and public-key encryption would be impossible; however, if P=NP then indistinguishability obfuscators *would* exist for all circuits. This follows immediately from the fact that computing the lexicographically first equivalent circuit to an input circuit is in the polynomial hierarchy (specifically in $P^{\Sigma_3^P}$). Thus, it should not be possible to construct interesting cryptographic functionalities from indistinguishability obfuscators alone without proving P$\neq$NP along the way.

Deniable Encryption. In 1997, Canetti, Dwork, Naor, and Ostrovsky [CDNO97] proposed the intriguing problem of *deniable encryption*: In deniable (public-key) encryption, there is a public key used for encryption by an honest sender, to produce a ciphertext that is made public. However, at a later time, an adversary may approach the sender and force her to reveal the message that she encrypted together with the randomness that she used to encrypt the message. In deniable encryption, we would like to provide the sender with a "faking" algorithm that allows her to produce fake randomness for any message of her choice, that appears indistinguishable from the true randomness to any adversary (that does not possess the secret decryption key). Note that any deniable encryption scheme must incur at least some negligible decryption error. Unfortunately, all known proposed algorithms for deniable encryption allow an attacker to gain a noticeable $1/n$ advantage in distinguishing real randomness from fake randomness, where roughly speaking n is the length of the ciphertext [CDNO97]. In fact, over the last 16 years, high-profile papers have claimed solutions to this problem, most notably the recent work of [DF11a], but these have all been shown to be incorrect [DF11b].

We propose a completely new approach to deniable encryption based on obfuscation. The essential idea behind our approach is the following: The public key will consist of an two obfuscated programs.

- The first obfuscated program $\mathsf{Encrypt}$ is used for encryption of ciphertexts. This program takes as input a message m and a long random string u. Before encrypting, it first checks for a special "hidden sparse trigger". It checks to see if the random coins are a special encoding from a sparse space. To do this it checks if the claimed randomness u is *itself* an encoding of some valid ciphertext c and the message m, under a (different internal) encoding scheme described below. If so, it outputs this ciphertext c. Otherwise, it uses u to encrypt m in an (essentially) normal way to produce a ciphertext c under a standard encryption E_{PKE}.
- The second program is an obfuscation of an $\mathsf{Explain}$ algorithm. The explanation algorithm takes as input a ciphertext c, a message m, and some randomness. This explanation algorithm doesn't care what c is "really" encrypting. It simply outputs an encoding of (c, m). Thus, the explanation mode can be used (by anyone) to create fake randomness that can explain any ciphertext as any desired message.

Thus, our approach uses two algorithms in tandem. The $\mathsf{Encrypt}$ algorithm will produce encryptions of some message m. When run with coins u chosen truly at random, it will almost always correctly encrypt m, since the "hidden sparse triggers" form a negligible fraction of possible u values.

The $\mathsf{Explain}$ algorithm takes in a ciphertext-message pair (c, m) and will produce an encoding u' that serves as a hidden trigger for the $\mathsf{Encrypt}$ program. We use the terminology "ciphertext" to refer to the ciphertext, produced by $\mathsf{Encrypt}$, that will be sent to the receiver. We use "encoding" or "hidden trigger encoding" to refer to a value output by the $\mathsf{Explain}$ program. The intended semantics are that if $\mathsf{Encrypt}$ program discovers this hidden trigger, it is to immediately output the ciphertext c that is hidden in the encoding.

For this approach to work, clearly $\mathsf{Explain}$ will need to produce pseudorandom encodings, so that the adversary cannot immediately distinguish fake randomness from true randomness. Upon closer inspection, we see that the attacker can also take some randomness u^* given by the honest sender, modify it, and provide it as input to the $\mathsf{Encrypt}$ program, and see what happens. Indeed, we see that the encodings produced by $\mathsf{Explain}$ will in fact need to be CCA-secure, since the $\mathsf{Encrypt}$ program itself implements a limited kind of CCA attack on $\mathsf{Explain}$ encodings. For example, if the adversary can take a fake randomness u^*, and modify it slightly so that it still encodes the same ciphertext-message pair (c^*, m^*), then the adversary would most likely learn that the sender is faking. Note that we must show that encodings produced by $\mathsf{Explain}$ are pseudorandom, even when subject to this kind of CCA attack.

Finally, we must make our construction work given only an indistinguishability obfuscator, not a virtual black-box obfuscator. To do this, we again make use of the punctured programs technique with multiple PRFs that have been enhanced to have additional properties: we consider *injective* PRFs, where the function $\mathrm{PRF}(K, \cdot)$ is injective with high probability over the choice of key K; and also what we call *extracting* PRFs, where the distribution $(K, \mathrm{PRF}(K, X))$ is very close to a uniform distribution as long as the distribution X

has sufficient min-entropy. We show how to build injective and extracting puncturable PRFs assuming only one-way functions. Thus, we obtain deniable encryption from indistinguishability obfuscation and one-way functions.

Publicly Deniable Encryption. Our obfuscation-based construction of deniable encryption actually achieves a new and stronger notion that we call *publicly deniable encryption*. Recall that in a traditional deniable encryption scheme, the sender must remember his actual randomness and use it to construct fake randomness for other messages. In a publicly deniable encryption scheme, we do not require a sender to know the randomness that was used to generate any particular ciphertext in order to be able to generate fake randomness for it. Thus, even a third party that had nothing to do with the actual honest generation of a ciphertext can nonetheless produce fake randomness for that ciphertext corresponding to any message of his choice. Note that our construction idea above achieves this, since the explanation mode does not need to see what randomness was originally used to create a particular ciphertext.

The public deniability aspect of our security notion allows us to think of producing fake randomness as a kind of "simulation." With this perspective, we observe that we can separate out the security of publicly deniable encryption into two parts: First, we must prove ordinary indistinguishability (IND-CPA) security of the encryption scheme. In other words, an encryption of m_0 must be indistinguishable from an encryption of m_1 for any two messages m_0, m_1. Second, we have a separate "explanation" security requirement, that says that for any fixed message m, an encryption of m together with m and the true randomness used to create that ciphertext should be indistinguishable from an encryption of m together with the *same* message m but with fake randomness explaining the ciphertext (also as an encryption of m).

Note, however, that these two security properties together imply ordinary deniability, since one can first invoke explanation security to create a hybrid where a fake explanation is offered with respect to the actual message m_b that was encrypted, and then invoke IND-CPA to argue that this must be indistinguishable from a fake explanation offered about a ciphertext that was actually created using m_{1-b}.

Universal Deniable Encryption. We also consider another new notion that we call *universal deniable encryption*. In universal deniable encryption, we imagine that there are several different public key infrastructures (PKIs) already set up for a variety of legacy public-key encryption schemes (perhaps such as IND-CPA-secure systems based on RSA or El-Gamal). However, we would like to enable users to deniably encrypt messages, without requiring all parties to rekey their public keys. We can do so as long as there is a trusted party (that cannot be coerced into revealing its randomness by the adversary) that is able to publish a single additional public parameter. (The trusted party plays no further role.) Users are now instructed to encrypt all messages using a combination of the public parameters and the recipient's legacy public key. A recipient will still be able to decrypt the message using its original secret key and decryption algorithm. If an adversary now approaches a user and forces the user to reveal what message and randomness it used (knowing that this user would have used the new standard encryption mechanism to encrypt), the user can provide convincing fake randomness to the adversary to explain any message of his choice.

Technical overview of Deniable Encryption Scheme. We now provide a brief overview of how our main publicly deniable encryption scheme works (this overview does not refer to universal deniable encryption). The technical heart of our construction is the $\mathsf{Explain}$ encoding scheme, and the argument that encodings produced by this encoding scheme on input (c^*, m^*) are indistinguishable from true randomness used to produce c^* when encrypting m^* using E_{PKE}, even when the adversary is given access to the obfuscated $\mathsf{Encrypt}$ and $\mathsf{Explain}$ programs. The construction and argument crucially relies on the punctured programs technique.

We think of the randomness u used by $\mathsf{Encrypt}$ as consisting of two strings $u[1]$ and $u[2]$. To create a fake explanation that a ciphertext c is an encryption of a bit m, $\mathsf{Explain}$ will use additional randomness r to produce an encoding as follows: Set $\alpha = \mathrm{PRF}_2(K_2, (m, c, \mathrm{PRG}(r)))$. Let $\beta = \mathrm{PRF}_3(K_3, \alpha) \oplus (m, c, \mathrm{PRG}(r))$. The encoding e will be $e = (\alpha, \beta)$. Here, it is important that PRF_2 is an injective puncturable PRF with a large co-domain, and that PRF_3 is a puncturable PRF.

Consider a ciphertext $c^* = E_{\mathrm{PKE}}(m^*, u^*)$, and a particular fake explanation encoding $e^* = (\alpha^*, \beta^*)$ for (c^*, m^*), that was generated using randomness r^*. We will show that e^* is "pseudorandom" by repeated application of the punctured programs technique, by which eventually we will replace e^* with a random string (such that it always yields a particular ciphertext c^* when applied on input m^* in $\mathsf{Encrypt}$). Recall that the key technical difficulty in applying the punctured programs technique is to ensure that when puncturing a program – cutting some potential computation out of the program – we must make sure that the *functionality* of the program is not affected, so that we may apply indistinguishability obfuscation. The essential way we do this with respect to the $\mathsf{Explain}$ procedure described above is:

1. First, we replace $\mathrm{PRG}(r^*)$ with a totally random value $\tilde{r}$, so that we can argue that $\mathrm{PRF}_2(K_2, \cdot)$ will never be invoked on $(m^*, c^*, \tilde{r})$ in the functionality of $\mathsf{Explain}$. This lets us puncture out $(m^*, c^*, \tilde{r})$ from the PRF key K_2.

2. Having punctured out $(m^*, c^*, \tilde{r})$, we can then replace α^* with a completely random string, because of the punctured PRF key. By the sparseness of the image of $\mathrm{PRF}_2(K_2, \cdot)$, because it is injective, this means that we can then safely puncture α^* from the PRF key K_3 without disturbing the functionality of $\mathsf{Explain}$.

3. Finally, then, we are able to replace β^* with a random value.

Above we sketched the changes to $\mathsf{Explain}$, but we note that additional (even greater) changes will need to be made to $\mathsf{Encrypt}$, because this procedure must maintain the invariant that $\mathsf{Encrypt}(m^*; u^*) = \mathsf{Encrypt}(m^*; e^*) = c^*$ throughout all our changes. This proceeds in a manner similar to what is described above, but is somewhat more technical, and we crucially use the injectivity of $\mathrm{PRF}(K_2, \cdot)$ beyond just the sparseness of its image.

Hidden Sparse Triggers We briefly recap the main properties of our hidden sparse trigger concept as we believe that it will have applications elsewhere in cryptography. The highlighted properties are:

- Sparseness. The trigger values must live in a sparse subset of some larger set. For our application, if the triggers were not sparse the deniable encryption system would not be correct.

- Oblivious Sampling. It must be possible to sample obliviously from the full set. In our construction, this is simply choosing a string uniformly at random.

- Indistinguishability under malleability attacks. It should be hard to distinguish a value chosen randomly from the larger set from the sparse hidden trigger set. This must be true even given a program that detects and utilizes the hidden trigger.

- Publicly Generated Triggers. It should be possible for anyone to generate hidden triggers.

Core primitives from indistinguishability obfuscation. We show the generality of our punctured programs technique by also constructing a variety of core cryptographic objects from indistinguishability obfuscation and one-way functions (or close variants). In particular, we obtain:

- Public-key encryption from indistinguishability obfuscation and one-way functions, as already sketched in the introduction above.

- Short "hash-and-sign" selectively secure signatures from indistinguishability obfuscation and one-way functions. These signatures are secure in a model where the adversary must declare the message m^* on which a forgery will take place at the start, but then can adaptively query any number of messages distinct from m^* on which it can receive signatures.

- An adaptively chosen-ciphertext secure (CCA2-secure) public-key encryption scheme from indistinguishability obfuscation and one-way functions.

- An adaptively chosen-ciphertext secure (CCA2-secure) key encapsulation mechanism (KEM) from indistinguishability obfuscation and one-way functions. This construction is remarkably simple.
- Perfect non-interactive zero knowledge (NIZK) arguments for all NP languages in the common reference string (CRS) model from indistinguishability obfuscation and one-way functions.
- Injective trapdoor functions from indistinguishability obfuscation and injective one-way functions.
- Oblivious Transfer protocols from indistinguishability obfuscation and one-way functions.

Taken altogether these results (along with the functional encryption result of GGHRSW [GGH+13a]) cover a large swathe of cryptography. This feeds into our research vision of Indistinguishability Obfuscation becoming a central hub of cryptography. In an ideal future "most" cryptographic primitives will be shown to be realizable from $i\mathcal{O}$ and one way functions.

2 Publicly Deniable Encryption

We now give our definition of publicly deniable encryption. We show that our notion of publicly deniable encryption implies the notion of deniable encryption of [CDNO97] (without any "plan-ahead" requirement).

Definition 1 (Publicly Deniable Encryption)**.** A *publicly deniable encryption scheme* over a message space $\mathcal{M} = \mathcal{M}_\lambda$ consists of four algorithms $\mathsf{Setup}\ \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Explain}$.

- $\mathsf{Setup}(1^\lambda)$ is a polynomial time algorithm that takes as input the unary representation of the security parameter λ and outputs a public key PK and a secret key SK.
- $\mathsf{Encrypt}(\mathrm{PK}, m \in \mathcal{M}; u)$ is a polynomial time algorithm that takes as input the public key PK a message m and uses random coins u. (We call out the random coins explicitly for this algorithm). It outputs a ciphertext c.
- $\mathsf{Decrypt}(\mathrm{SK}, c)$ is a polynomial time algorithm that takes as input a secret key SK and ciphertext c, and outputs a message m.
- $\mathsf{Explain}(\mathrm{PK}, c, m; r)$ is a polynomial time algorithm that takes as input a public key PK, a ciphertext c, and a message m, and outputs and a string e, that is the same size as the randomness u taken by $\mathsf{Encrypt}$ above.

We say that a publicly deniable encryption scheme is correct if for all messages $m \in \mathcal{M}$:

$$\Pr[\ (\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Setup}(1^\lambda);\ \ \mathsf{Decrypt}(\mathrm{SK}, \mathsf{Encrypt}(\mathrm{PK}, m; u)) \neq m\ ] = \mathrm{negl}(\lambda)$$

where $\mathrm{negl}(\cdot)$ is a negligible function.

Remark 1. The $\mathsf{Explain}$ algorithm does not directly factor into the correctness definition.

2.1 Security

Any publicly deniable encryption system must meet two security requirements both given as game-based definitions. The first is Indistinguishability under Chosen Plaintext Attack (IND-CPA). This is exactly the same security game [GM84] as in standard public key encryption schemes. (We include a description here for self-containment.)

The second is a new security property called indistinguishability of *explanation*. Informally speaking, if one considers a message m, an encryption c of m and the coins, u used in encryption, then indistinguishability of explanation states that no attacker can distinguish between (m, c, u) and $(m, c, \mathsf{Explain}(c, m; r))$. We now provide both security definitions formally:

Indistinguishability under Chosen Plainext Attack.

We describe the IND-CPA game as a multi-phase game between an attacker $\mathcal{A}$ and a challenger.

Setup: The challenger runs $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives PK to $\mathcal{A}$.

Challenge: $\mathcal{A}$ submits two messages $m_0, m_1 \in \mathcal{M}$, and is given $c^* = \mathsf{Encrypt}(\mathrm{PK}, m_b; u)$ for random coins u.

Guess $\mathcal{A}$ then outputs a bit b' in $\{0, 1\}$.

We define the advantage of an algorithm $\mathcal{A}$ in this game to be

$$\mathsf{Adv}_{\mathcal{A}} = \Pr[b' = b] - \frac{1}{2}$$

Definition 2. An publicly deniable scheme is IND-CPA secure if for all poly-time $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ is negligible.

Indistinguishability of Explanation.

We describe the Indistinguishability of Explanation security game as a multi-phased game between an $\mathcal{A}$ and a challenger.

Setup: The challenger runs $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives PK to $\mathcal{A}$.

Challenge: $\mathcal{A}$ submits *a single* message $m \in \mathcal{M}$. The challenger creates $c^* = \mathsf{Encrypt}(\mathrm{PK}, m; u)$ for random u and it computes $e = \mathsf{Explain}(\mathrm{PK}, c^*; r)$ for random r. The challenger finally flips a coin $b \in \{0, 1\}$. If $b = 0$ it outputs (c^*, u). If $b = 1$ it outputs (c^*, e).

Guess $\mathcal{A}$ then outputs a bit b' in $\{0, 1\}$.

We define the advantage of an algorithm $\mathcal{A}$ in this game to be

$$\mathsf{Adv}_{\mathcal{A}} = \Pr[b' = b] - \frac{1}{2}$$

Definition 3. An publicly deniable scheme has Indistinguishability of Explanation if for all poly-time $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ is negligible.

Remark 2. We remark that a publicly deniable encryption scheme for one bit messages immediately implies a publicly deniable encryption scheme for multi-bit messages of any polynomial length n bits. One can simply create a ciphertext for n bit message as a sequence of n single bit encryptions (i.e. "bit-by-bit encryption"). The $\mathsf{Explain}$ algorithm for the n bit scheme simply calls the single bit explain algorithm for each of the n ciphertext components. Security for both IND-CPA and Indistinguishability of Explanation follows via the usual one bit to many bit hybrid argument.

2.2 Implication for Deniable Encryption

We now show that a publicly deniable encryption is also a deniable encryption as defined by CDNO [CDNO97]. We begin by briefly recalling the CDNO notion of (sender) deniable encryption. We simplify the exposition to consider encryption as a standard one pass algorithm that takes in the public key and outputs a ciphertext.

A (standard) deniable encryption scheme has four algorithms: $\mathsf{Setup}_{\mathsf{DE}}, \mathsf{Encrypt}_{\mathsf{DE}}, \mathsf{Decrypt}_{\mathsf{DE}}, \mathsf{Fake}_{\mathsf{DE}}$.

- $\mathsf{Setup}_{\mathsf{DE}}(1^\lambda)$ is a polynomial time algorithm that takes as input the unary representation of the security parameter λ and outputs a public key PK and a secret key SK.

- $\mathsf{Encrypt}_{\mathsf{DE}}(\mathrm{PK}, m \in \mathcal{M}; u)$ is a polynomial time algorithm that takes as input the public key PK a message m and uses random coins u. (We call out the random coins explicitly for this algorithm). It outputs a ciphertext c.
- $\mathsf{Decrypt}_{\mathsf{DE}}(\mathrm{SK}, c)$ is a polynomial time algorithm that takes as input a secret key SK and ciphertext c, and outputs a message m.
- $\mathsf{Fake}_{\mathsf{DE}}(\mathrm{PK}, m, u, c, m'; r)$ is a polynomial time algorithm that takes as input a public key PK, an "original" message m, bitstring u, ciphertext c, and a "faked" message m', and outputs and a string e, that is the same size as the randomness u taken by $\mathsf{Encrypt}$ above.

We can see that the algorithms essentially align with the publicly deniable algorithms. The main exception is that the $\mathsf{Fake}_{\mathsf{DE}}$ algorithm takes in the original randomness used for encryption as well as the original message. Whereas the $\mathsf{Explain}$ algorithm of publicly deniable encryption only requires the transmitted ciphertext and new message (and none of the other history.)

Security in standard deniable encryption consists of two notions. The first is IND-CPA as given above for publicly deniable encryption. The second is a deniability game given here.

Setup: The challenger runs $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Setup}_{\mathsf{DE}}(1^\lambda)$ and gives PK to $\mathcal{A}$.

Challenge: $\mathcal{A}$ submits *two* message $m_0, m_1 \in \mathcal{M}$. The challenger flips a random coin b. If $b = 0$, it creates $c^* = \mathsf{Encrypt}_{\mathsf{DE}}(\mathrm{PK}, m_0; u)$ for random u. It outputs (c^*, u).

If $b = 1$ it creates $c^* = \mathsf{Encrypt}_{\mathsf{DE}}(\mathrm{PK}, m_0; u)$ and it and it computes $e = \mathsf{Explain}(\mathrm{PK}, m_1, u, c^*, m_0; r)$ for random r. It outputs (c^*, e).

Guess $\mathcal{A}$ then outputs a bit b' in $\{0, 1\}$.

We define the advantage of an algorithm $\mathcal{A}$ in this game to be

$$\mathsf{Adv}_{\mathcal{A}} = \Pr[b' = b] - \frac{1}{2}$$

Definition 4. A deniable encryption scheme is Deniable if for all poly-time $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ is negligible.

The Implication Now we can show that publicly deniable encryption implies standard deniable encryption. (We use the subscript of PDE to denote the original deniable encryption system and DE to denote the constructed deniable encryption system.) The first three algorithms are set to be exactly the same. $\mathsf{Setup}_{\mathsf{DE}} = \mathsf{Setup}_{\mathsf{PDE}}, \mathsf{Encrypt}_{\mathsf{DE}} = \mathsf{Encrypt}_{\mathsf{PDE}}, \mathsf{Decrypt}_{\mathsf{DE}} = \mathsf{Decrypt}_{\mathsf{PDE}}$. The faking algorithm is $\mathsf{Fake}_{\mathsf{DE}}(\mathrm{PK}, m, u, c, m'; r) = \mathsf{Explain}(\mathrm{PK}, c, m')$. In essence, the Explain algorithm is simply used to fake and just doesn't bother to use the original randomness or message.

We now sketch by a simple hybrid argument that a deniable encryption scheme built from publicly deniable one (in the manner outlined above) is secure in the deniability game.

Let Hybrid:1, be the experiment where $(c^* = \mathsf{Encrypt}_{\mathsf{DE}}(\mathrm{PK}, m_0; u), u)$ is output as an encryption of m_0 with the real coins. Now let Hybrid:2 be the experiment where $(CT^* = \mathsf{Encrypt}_{\mathsf{DE}}(\mathrm{PK}, m_0; u), e = \mathsf{Explain}_{\mathsf{PDE}}(\mathrm{PK}, c^*, m_0; r))$. By definition of the Indistinguishability of Explanation, Hybrid:1 is indistinguishabile from Hybrid:2.

Now let Hybrid:3 be the experiment where the challenger outputs $(c^* = \mathsf{Encrypt}_{\mathsf{DE}}(\mathrm{PK}, m_1; u),\ e = \mathsf{Explain}_{\mathsf{PDE}}(\mathrm{PK}, c^*, m_0; r)) = \mathsf{Fake}_{\mathsf{DE}}(\mathrm{PK}, m_1, u, c, m_0; r)$. Any attacker that can distinguish between Hybrid:1, Hybrid:2 can break IND-CPA of the scheme. A reduction algorithm $\mathcal{B}$ will simply pass the IND-CPA challenge ciphertext c^* of the attacker along with computing itself $e = \mathsf{Explain}_{\mathsf{PDE}}(\mathrm{PK}, c^*, m_0; r))$. It then simply relays the attacker's answer to the IND-CPA game. The simplicity of this argument derives from the fact that the $\mathsf{Explain}_{\mathsf{PDE}}$ algorithm *does not use* any of the coins used to create the original ciphertext.

We conclude by observing that Hybrid:1 and Hybrid:3 correspond respectively to where $b = 0$ and $b = 1$ in the CDNO game. Thus, a publicly deniable encryption scheme gives one in the standard sense.

Advantages of Publicly Deniable Encryption The abstraction of Publicly Deniable Encryption has two advantages. First, public deniability can be a useful property in practice. For example, consider a party was expected to reveal coins for a ciphertext, but had lost them. Using the public deniability Explain algorithm, it could make them up just knowing the transmitted ciphertext.

Second, in proving our construction of Section 4 the we found that proving indistinguishability of explanation is significantly simpler than directly proving deniability. [1] The primary reason stems from the fact that the indistinguishability of explanation only requires a single message in its security game. Our abstraction separates the (in)ability to distinguish between encryptions of different messages from the (in)ability to distinguish between legitimate and explained randomness. [2]

3 Preliminaries

In this section, we define indistinguishability obfuscation, and variants of pseudo-random functions (PRFs) that we will make use of. All the variants of PRFs that we consider will be constructed from one-way functions.

Indistinguishability Obfuscation. The definition below is from [GGH+13a]; there it is called a "family-indistinguishable obfuscator", however they show that this notion follows immediately from their standard definition of indistinguishability obfuscator using a non-uniform argument.

Definition 5 (Indistinguishability Obfuscator ($i\mathcal{O}$))**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

- For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1$$

- For any (not necessarily uniform) PPT adversaries $Samp$, D, there exists a negligible function α such that the following holds: if $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)] > 1 - \alpha(\lambda)$, then we have:

$$\Big| \Pr\left[D(\sigma, i\mathcal{O}(\lambda, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right] - \Pr\left[D(\sigma, i\mathcal{O}(\lambda, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right] \Big| \leq \alpha(\lambda)$$

In this paper, we will make use of such indistinguishability obfuscators for all polynomial-size circuits:

Definition 6 (Indistinguishability Obfuscator for $P/poly$)**.** A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* for $P/poly$ if the following holds: Let $\mathcal{C}_\lambda$ be the class of circuits of size at most λ. Then $i\mathcal{O}$ is an indistinguishability obfuscator for the class $\{\mathcal{C}_\lambda\}$.

Such indistinguishability obfuscators for all polynomial-size circuits were constructed under novel algebraic hardness assumptions in [GGH+13a].

PRF variants. We first consider some simple types of constrained PRFs [BW13, BGI13, KPTZ13], where a PRF is only defined on a subset of the usual input space. We focus on *puncturable* PRFs, which are PRFs that can be defined on all bit strings of a certain length, except for any polynomial-size set of inputs:

Definition 7. A *puncturable* family of PRFs F mapping is given by a triple of Turing Machines Key_F, Puncture_F, and Eval_F, and a pair of computable functions $n(\cdot)$ and $m(\cdot)$, satisfying the following conditions:

[1] In an earlier internal draft we proved deniability directly; however, the proof was more complex and required an additional property on the constrained PRFs that was not obtainable from one way functions.

[2] We shall see that even if an attacker is given the encryption systems secret key, it cannot win in the explanation game.

- **[Pseudorandom at punctured points]** For every PPT adversary (A_1, A_2) such that $A_1(1^\lambda)$ outputs a set $S \subseteq \{0,1\}^{n(\lambda)}$ and state σ, consider an experiment where $K \leftarrow \mathrm{Key}_F(1^\lambda)$ and $K_S = \mathrm{Puncture}_F(K, S)$. Then we have

$$\left| \Pr\left[A_2(\sigma, K_S, S, \mathsf{Eval}_F(K, S)) = 1\right] - \Pr\left[A_2(\sigma, K_S, S, U_{m(\lambda)\cdot|S|}) = 1\right] \right| = negl(\lambda)$$

  where $\mathsf{Eval}_F(K, S)$ denotes the concatenation of $\mathsf{Eval}_F(K, x_1)), \ldots, \mathsf{Eval}_F(K, x_k))$ where $S = \{x_1, \ldots, x_k\}$ is the enumeration of the elements of S in lexicographic order, $negl(\cdot)$ is a negligible function, and U_ℓ denotes the uniform distribution over ℓ bits.

For ease of notation, we write $F(K, x)$ to represent $\mathsf{Eval}_F(K, x)$. We also represent the punctured key $\mathrm{Puncture}_F(K, S)$ by $K(S)$.

The GGM tree-based construction of PRFs [GGM84] from one-way functions are easily seen to yield puncturable PRFs, as recently observed by [BW13, BGI13, KPTZ13]. Thus we have:

Theorem 1. [GGM84, BW13, BGI13, KPTZ13] If one-way functions exist, then for all efficiently computable functions $n(\lambda)$ and $m(\lambda)$, there exists a puncturable PRF family that maps $n(\lambda)$ bits to $m(\lambda)$ bits.

Next we consider families of PRFs that are with high probability injective:

Definition 8. A *statistically injective* (puncturable) PRF family with failure probability $\epsilon(\cdot)$ is a family of (puncturable) PRFs F such that with probability $1 - \epsilon(\lambda)$ over the random choice of key $K \leftarrow \mathrm{Key}_F(1^\lambda)$, we have that $F(K, \cdot)$ is injective.

If the failure probability function $\epsilon(\cdot)$ is not specified, then $\epsilon(\cdot)$ is a negligible function.

Theorem 2. If one-way functions exist, then for all efficiently computable functions $n(\lambda)$, $m(\lambda)$, and $e(\lambda)$ such that $m(\lambda) \geq 2n(\lambda) + e(\lambda)$, there exists a puncturable statistically injective PRF family with failure probability $2^{-e(\lambda)}$ that maps $n(\lambda)$ bits to $m(\lambda)$ bits.

Proof. For ease of notation, we suppress the dependence of n, m, e on λ. Let $\mathcal{H}$ be a family of 2-universal hash functions mapping n bits to m bits. Let F be a family of puncturable PRFs mapping n bits to m bits, which exists by Theorem 1.

Consider the family F' defined as follows: The key space for F' consists of a key K for F and a hash function h chosen from $\mathcal{H}$. We define $F'((K, h), x) = F(K, x) \oplus h(x)$.

Observe that if F were a truly random function, then $F(x) \oplus h(x)$ would still be a truly random function for an independent random choice of h (even if h were public). Thus, because F is a PRF, we have that F' is also a PRF. Similarly, because F is puncturable, so is F': one can puncture the key (K, h) on a set S by simply puncturing K on the same set S.

All that remains is to show that F' is statistically injective. Consider $x \neq x' \in \{0,1\}^n$. Fix any K. By pairwise independence, $\Pr_h[h(x) = h(x') \oplus F(K, x) \oplus F(K, x')] = 2^{-m}$. Taking a union bound over all distinct pairs (x, x') gives us that:

$$\Pr_h[\exists x \neq x' : F'((K, h), x) = F'((K, h), x')] \leq 2^{-(m-2n)}$$

Averaging over the choice of K finishes the proof, by the choice of $m \geq 2n + e$. ∎

Finally, we consider PRFs that are also (strong) extractors over their inputs:

Definition 9. An *extracting* (puncturable) PRF family with error $\epsilon(\cdot)$ for min-entropy $k(\cdot)$ is a family of (puncturable) PRFs F mapping $n(\lambda)$ bits to $m(\lambda)$ bits such that for all λ, if X is any distribution over $n(\lambda)$ bits with min-entropy greater than $k(\lambda)$, then the statistical distance between $(K \leftarrow \mathrm{Key}_F(1^\lambda), F(K, X))$ and $(K \leftarrow \mathrm{Key}_F(1^\lambda), U_{m(\lambda)})$ is at most $\epsilon(\lambda)$, where U_ℓ denotes the uniform distribution over ℓ-bit strings.

Theorem 3. If one-way functions exist, then for all efficiently computable functions $n(\lambda)$, $m(\lambda)$, $k(\lambda)$, and $e(\lambda)$ such that $n(\lambda) \geq k(\lambda) \geq m(\lambda) + 2e(\lambda) + 2$, there exists an extracting puncturable PRF family that maps $n(\lambda)$ bits to $m(\lambda)$ bits with error $2^{-e(\lambda)}$ for min-entropy $k(\lambda)$.

Proof. For ease of notation, we suppress the dependence of n, m, k, e on λ. By Theorem 2, let F be a family of puncturable statistically injective PRFs mapping n bits to $2n + e + 1$ bits, with error $2^{-(e+1)}$. Let $\mathcal{H}$ be a family of 2-universal hash functions mapping $2n + e + 1$ bits to m bits.

Consider the family F' defined as follows: The key space for F' consists of a key K for F and a hash function h chosen from $\mathcal{H}$. We define $F'((K, h), x) = h(F(K, x))$.

Observe that if F were a truly random function, then $h(F(x))$ would still be a truly random function for an independent random choice of h (even if h were public), by the Leftover Hash Lemma. Thus, because F is a PRF, we have that F' is also a PRF. Similarly, because F is puncturable, so is F': one can puncture the key (K, h) on a set S by simply puncturing K on the same set S.

Now suppose we have any distribution X over n bits with min-entropy at least $k \geq m + 2e + 2$. Fix any key K such that $F(K, \cdot)$ is injective. Then the Leftover Hash Lemma implies that that the statistical distance between $(h \leftarrow \mathcal{H}, h(F(K, X)))$ and $(h \leftarrow \mathcal{H}, U_m)$ is at most $2^{-(e+1)}$.

Since the probability that K yields a non-injective $F(K, \cdot)$ is also at most $2^{-(e+1)}$, by a union bound, this implies that that statistical distance between $((K, h), F'((K, h), X))$ and $((K, h), U_m)$ is at most 2^{-e}, as desired. ∎

4 Our Deniable Encryption System

Let λ be a security parameter. Below, we assume that the $\mathsf{Encrypt}_{PKE}(\mathrm{PK}, \cdot; \cdot)$ algorithm accepts a one-bit message m and randomness of length $\ell_e \geq 2$, and outputs ciphertexts of length ℓ_c. For ease of notation, we suppress the dependence of these lengths on λ.

Our Encrypt program will take a one-bit message m, and randomness $u = (u[1], u[2])$ of length $\ell_1 + \ell_2$, where $|u[1]| = \ell_1$ and $|u[2]| = \ell_2$. We will set $\ell_1 = 5\lambda + 2\ell_c + \ell_e$, and $\ell_2 = 2\lambda + \ell_c + 1$.

We use a PRG that maps $\{0,1\}^\lambda$ to $\{0,1\}^{2\lambda}$. We also make use of three different PRF variants in our construction:

- An puncturable extracting PRF $F_1(K_1, \cdot)$ that accepts inputs of length $\ell_1 + \ell_2 + 1$, and output strings of length ℓ_e. It is extracting when the input min-entropy is greater than $\ell_e + 2\lambda + 4$, with error less than $2^{-(\lambda+1)}$. Observe that $n + \ell_1 + \ell_2 \geq \ell_e + 2\lambda + 4$, and thus if one-way functions exist, then such puncturable extracting PRFs exist by Theorem 3.
- A puncturable statistically injective PRF $F_2(K_2, \cdot)$ that accepts inputs of length $2\lambda + \ell_c + 1$, and outputs strings of length ℓ_1. Observe that $\ell_1 \geq 2 \cdot (2\lambda + \ell_c + 1) + \lambda$, and thus if one-way functions exist, such puncturable statistically injective PRFs exist by Theorem 2.
- A puncturable PRF $F_3(K_3, \cdot)$ that accepts inputs of length ℓ_1 and outputs strings of length ℓ_2. If one-way functions exist, then such a puncturable PRF exists by Theorem 1.

The public key consist of indistinguishability obfuscation of the following two programs:

4.1 Proofs of Security

We will use the following simple lemma in the proofs of security below.

Lemma 1. Except with negligible probability over the choice of key K_2, the following two statements hold:

1. For any fixed $u[1] = \alpha$, there can exist at most one pair (m, β) such that the input m with randomness $u = (\alpha, \beta)$ will cause the Step 1 check of the Encrypt Program to be satisfied.
2. There are at most $2^{2\lambda+\ell_c+1}$ values for randomness u that can cause the Step 1 check of the Encrypt Program to be satisfied.

Encrypt

Constants: Public key PK and keys K_1, K_2, and K_3.
Input: Message m, randomness $u = (u[1], u[2])$.

1. If $F_3(K_3, u[1]) \oplus u[2] = (m', c', r')$ for (proper length) strings c', r', m', and $m' = m$, and $u[1] = F_2(K_2, (m', c', r'))$, then output $c = c'$ and end.
2. Else let $x = F_1(K_1, (m, u))$. Output $c = \mathsf{Encrypt}_{PKE}(\mathrm{PK}, m; x)$.

Figure 1: Program Encrypt

Explain

Constants: keys K_2 and K_3.
Input: Message m, ciphertext c, randomness $r \in \{0,1\}^\lambda$.

1. Set $\alpha = F_2(K_2, (m, c, \mathrm{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, c, \mathrm{PRG}(r))$. Output $e = (\alpha, \beta)$.

Figure 2: Program Explain

Proof. We begin by conditioning on the event that key K_2 is chosen so that $F_2(K_2, \cdot)$ is injective, which happens with all but negligible probability by the definition of statistically injective PRF.

We prove both statements simultaneously. The Step 1 check includes verifying that $u[1] = F_2(K_2, (m', c', r'))$ for some strings m', c', r'. Observe that there are at most $2^{2\lambda+\ell_c+1}$ such strings. Furthermore, by the injectivity of $F_2(K_2, \cdot)$, this can only be satisfied with respect to a unique input string m'. However, the Step 1 check also verifies that $u[2] = F_3(K_3, u[1]) \oplus (m', c', r')$. Thus, for a given $u[1]$ value, there can be at most one such string $u[2]$ that causes the check to pass. ∎

4.1.1 IND-CPA

First, we prove semantic security (IND-CPA) of our encryption scheme:

Proof. For ease of reference, we first write out the IND-CPA security game in full, specialized to our construction.
IND-CPA Game. The attacker must guess the bit g with non-negligible advantage over random guessing.

1. Choose keys K_1, K_2, K_3 at random.
2. Let $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Setup}_{PKE}(1^\lambda)$.
3. Let $P_{\mathrm{enc}} = i\mathcal{O}(\mathrm{Encrypt})$. Let $P_{\mathrm{explain}} = i\mathcal{O}(\mathrm{Explain})$.
4. Flip coin $g \in \{0,1\}$.
5. Select $u \in \{0,1\}^{\ell_1+\ell_2}$ at random.
6. Let $c = P_{\mathrm{enc}}(g; u)$.
7. Output (μ, c).

Hybrid 1. In our first Hybrid, we replace Step 7:

$$\text{Let } c = P_{\mathrm{enc}}(g; u).$$

with

$$\text{Let } x = F_1(K_1, (g, u)). \text{ Let } c = \mathsf{Encrypt}_{PKE}(\mathrm{PK}, g; x)$$

In other words, we omit the Step 1 check within Program Encrypt. By Lemma 1, we have that with overwhelming probability, the Step 1 check in Program Encrypt will not be satisfied when preparing the ciphertext c using true randomness u. Thus, this Hybrid is statistically close to the original experiment.

Hybrid 2. In our next Hybrid, we eliminate the choice of randomness u in Step 6, and again replace Step 7 with:

$$\text{Select } x \in \{0,1\}^{\ell_e} \text{ at random. Let } c = \mathsf{Encrypt}_{PKE}(\mathrm{PK}, g; x)$$

The statistical closeness of Hybrid 2 from Hybrid 1 follows from the extracting PRF property of F_1, as we now detail: Observe that the selection of u at random in Hybrid 1 contained min-entropy exceeding $\ell_e + 2(\lambda + 1)$ by our choice of parameters. Therefore, if we denote a uniform distribution over strings of length ℓ by U_ℓ, then the definition of extracting PRF guarantees that $(K_1 \leftarrow U_{|K_1|}, F_1(K_1, (g, U_{\ell_1+\ell_2})))$ and $(K_1 \leftarrow U_{|K_1|}, U_{\ell_e})$ are $2^{-(\lambda+1)}$ statistically close for any fixed bit g. Thus, these are within $2^{-\lambda}$ statistical distance for both $g = 0$ and $g = 1$.

Finally, we finish the proof by showing that if the adversary can succeed in Hybrid 2, then there exists an IND-CPA adversary defeating $\mathsf{Encrypt}_{PKE}$ itself.

Note that the output of Hybrid 2 no longer makes use of the indistinguishability obfuscator at all, and indeed it is exactly the same as the IND-CPA game for $\mathsf{Encrypt}_{PKE}$. Thus, if the adversary could guess g with non-negligible advantage in Hybrid 2, our adversary can guess g with (the same) non-negligible advantage in the IND-CPA game for $\mathsf{Encrypt}_{PKE}$. ■

4.1.2 Explainability

We now turn to the explainability game. This proof proceeds through several hybrids. To maintain ease of verification for the reader, we present a full description of each hybrid experiment, each one given on a separate page. The change between the presented hybrid and the previous hybrid will be denoted in red underlined font. The hybrids are chosen so that the indistinguishability of each successive hybrid experiment follows in a relatively straightforward manner.

Proof. Recall the explainability game. In Hybrid 0, below, we simply unwrap the explainability game, and eliminate the Step 1 check from the Encrypt program when preparing the initial encryption of the fixed challenge message m^*. Hybrid 0 is statistically close to the original explainability game by Lemma 1.

Hybrid 0. Fix a one-bit message m^*. The attacker must distinguish between $g = 0$ and $g = 1$ in the experiment below.

1. Choose K_1, K_2, K_3 at random.
2. Let $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Setup}_{PKE}$
3. Select u^* at random. Select r^* at random.
4. Let $x^* = F_1(K_1, (m^*, u^*))$. Let $c = \mathsf{Encrypt}_{PKE}(\mathrm{PK}, m^*; x^*)$.
5. Set $\alpha^* = F_2(K_2, (m^*, c^*, \mathrm{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (m^*, c^*, \mathrm{PRG}(r^*))$. Set $e^* = (\alpha^*, \beta^*)$.
6. Let $P_{\text{enc}} = i\mathcal{O}(\text{Encrypt})$. Let $P_{\text{explain}} = i\mathcal{O}(\text{Explain})$.
7. Flip coin $g \in \{0, 1\}$.
8. If $g = 0$, output $(P_{\text{enc}}, P_{\text{explain}}, u^*, c^*)$. If $g = 1$, output $(P_{\text{enc}}, P_{\text{explain}}, e^*, c^*)$.

Encrypt

Constants: Public key PK and keys K_1, K_2, and K_3.
Input: Message m, randomness $u = (u[1], u[2])$.

1. If $F_3(K_3, u[1]) \oplus u[2] = (m', c', r')$ for (proper length) strings c', r', m', and $m' = m$, and $u[1] = F_2(K_2, (m', c', r'))$, then output $c = c'$ and end.
2. Else let $x = F_1(K_1, (m, u))$. Output $c = \mathsf{Encrypt}_{PKE}(\mathrm{PK}, m; x)$.

Figure 3: Program Encrypt

Explain

Constants: keys K_2 and K_3.
Input: Message m, ciphertext c, randomness $r \in \{0,1\}^\lambda$.

1. Set $\alpha = F_2(K_2, (m, c, \text{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, c, \text{PRG}(r))$.
   Output $e = (\alpha, \beta)$.

Figure 4: Program Explain

Hybrid 1. In this hybrid, we modify the Encrypt program as follows: First, we add constants m^*, u^*, e^*, c^* to the program. Then, we add an "if" statement at the start that outputs c^* if the input is either (m^*, u^*) or (m^*, e^*), as this is exactly what the original Encrypt program would do by our choice of u^*, e^*. Because this "if" statement is in place, we know that F_1 cannot be evaluated at either (m^*, u^*) or (m^*, e^*) within the program, and therefore we can safely puncture key K_1 at these two positions.

By construction, the new Encrypt program is functionally equivalent to the original Encrypt program. Therefore the indistinguishability of Hybrid 0 and Hybrid 1 follows by the security of $i\mathcal{O}$.

Note: implicitly, all "if" statements that are added to programs with multiple checks are written in lexicographic order; that is, if $u^* < e^*$ in lexicographic order, the statement is written as "If $(m, u) = (m^*, u^*)$ or (m^*, e^*)", otherwise it is written as "If $(m, u) = (m^*, e^*)$ or (m^*, u^*)".

Explainability Game.

1. Choose K_1, K_2, K_3 at random.
2. Let $(\text{PK}, \text{SK}) \leftarrow \mathsf{Setup}_{PKE}$
3. Select u^* at random. Select r^* at random.
4. Let $x^* = F_1(K_1, (m^*, u^*))$. Let $c^* = \mathsf{Encrypt}_{PKE}(\text{PK}, m^*; x^*)$.
5. Set $\alpha^* = F_2(K_2, (m^*, c^*, \text{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (m^*, c^*, \text{PRG}(r^*))$.
   Set $e^* = (\alpha^*, \beta^*)$.
6. Let $P_{\text{enc}} = i\mathcal{O}(\text{Encrypt})$. Let $P_{\text{explain}} = i\mathcal{O}(\text{Explain})$.
7. Flip coin $g \in \{0, 1\}$.
8. If $g = 0$, output $(P_{\text{enc}}, P_{\text{explain}}, u^*, c^*)$. If $g = 1$, output $(P_{\text{enc}}, P_{\text{explain}}, e^*, c^*)$.

Encrypt

Constants: m^*, u^*, e^*, c^*; Public key PK and PRF keys $K_1(\{(m^*, u^*), (m^*, e^*)\})$, K_2, and K_3.
Input: Message m, randomness $u = (u[1], u[2])$.

1. If $(m, u) = (m^*, u^*)$ or (m^*, e^*), output c^* and stop.
2. If $F_3(K_3, u[1]) \oplus u[2] = (m', c', r')$ for (proper length) strings c', r', m',
   and $m' = m$,
   and $u[1] = F_2(K_2, (m', c', r'))$, then output $c = c'$ and end.
3. Else let $x = F_1(K_1, (m, u))$. Output $c = \mathsf{Encrypt}_{PKE}(\text{PK}, m; x)$.

Figure 5: Program Encrypt

Explain

Constants: PRF keys K_2, and K_3.
Input: Message m, ciphertext c, randomness r.

1. Set $\alpha = F_2(K_2, (m, c, \text{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, c, \text{PRG}(r))$.
   Output $e = (\alpha, \beta)$.

Figure 6: Program Explain

Hybrid 2. In this hybrid, the value x^* is chosen randomly instead of as the output of $F_1(K_1, (m^*, u^*))$. The indistinguishability of Hybrid 2 from Hybrid 1 follows immediately from the pseudorandomness property of the punctured PRF F_1.

Explainability Game.

1. Choose K_1, K_2, K_3 at random.
2. Let $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Setup}_{PKE}$
3. Select u^* at random. Select r^* at random.
4. Let x^* be chosen randomly. Let $c^* = \mathsf{Encrypt}_{PKE}(\mathrm{PK}, m^*; x^*)$.
5. Set $\alpha^* = F_2(K_2, (m^*, c^*, \mathrm{PRG}(r^*)))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (m^*, c^*, \mathrm{PRG}(r^*))$. Set $e^* = (\alpha^*, \beta^*)$.
6. Let $P_{\mathrm{enc}} = i\mathcal{O}(\mathrm{Encrypt})$. Let $P_{\mathrm{explain}} = i\mathcal{O}(\mathrm{Explain})$.
7. Flip coin $g \in \{0, 1\}$.
8. If $g = 0$, output $(P_{\mathrm{enc}}, P_{\mathrm{explain}}, u^*, c^*)$. If $g = 1$, output $(P_{\mathrm{enc}}, P_{\mathrm{explain}}, e^*, c^*)$.

Encrypt

Constants: m^*, u^*, e^*, c^*; Public key PK and PRF keys $K_1(\{(m^*, u^*), (m^*, e^*)\})$, K_2, and K_3.
Input: Message m, randomness $u = (u[1], u[2])$.

1. If $(m, u) = (m^*, u^*)$ or (m^*, e^*), output c^* and stop.
2. If $F_3(K_3, u[1]) \oplus u[2] = (m', c', r')$ for (proper length) strings c', r', m', and $m' = m$, and $u[1] = F_2(K_2, (m', c', r'))$, then output $c = c'$ and end.
3. Else let $x = F_1(K_1, (m, u))$. Output $c = \mathsf{Encrypt}_{PKE}(\mathrm{PK}, m; x)$.

Figure 7: Program Encrypt

Explain

Constants: PRF keys K_2, and K_3.
Input: Message m, ciphertext c, randomness r.

1. Set $\alpha = F_2(K_2, (m, c, \mathrm{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, c, \mathrm{PRG}(r))$. Output $e = (\alpha, \beta)$.

Figure 8: Program Explain

Hybrid 3. In this hybrid, instead of picking r^* at random and applying a PRG to it, a value $\tilde{r}$ is chosen at random from the codomain of the PRG. The indistinguishability of Hybrid 2 and Hybrid 3 follows immediately from the security of the PRG.

Explainability Game.

1. Choose K_1, K_2, K_3 at random.
2. Let $(\text{PK}, \text{SK}) \leftarrow \mathsf{Setup}_{PKE}$
3. Select u^* at random. Select $\tilde{r}$ at random.
4. Let x^* be chosen randomly. Let $c^* = \mathsf{Encrypt}_{PKE}(\text{PK}, m^*; x^*)$.
5. Set $\alpha^* = F_2(K_2, (m^*, c^*, \tilde{r}))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (m^*, c^*, \tilde{r})$.
   Set $e^* = (\alpha^*, \beta^*)$.
6. Let $P_{\text{enc}} = i\mathcal{O}(\text{Encrypt})$. Let $P_{\text{explain}} = i\mathcal{O}(\text{Explain})$.
7. Flip coin $g \in \{0, 1\}$.
8. If $g = 0$, output $(P_{\text{enc}}, P_{\text{explain}}, u^*, c^*)$. If $g = 1$, output $(P_{\text{enc}}, P_{\text{explain}}, e^*, c^*)$.

Encrypt

Constants: m^*, u^*, e^*, c^*; Public key PK and PRF keys $K_1(\{(m^*, u^*), (m^*, e^*)\})$, K_2, and K_3.
Input: Message m, randomness $u = (u[1], u[2])$.

1. If $(m, u) = (m^*, u^*)$ or (m^*, e^*), output c^* and stop.
2. If $F_3(K_3, u[1]) \oplus u[2] = (m', c', r')$ for (proper length) strings c', r', m',
   and $m' = m$,
   and $u[1] = F_2(K_2, (m', c', r'))$, then output $c = c'$ and end.
3. Else let $x = F_1(K_1, (m, u))$. Output $c = \mathsf{Encrypt}_{PKE}(\text{PK}, m; x)$.

Figure 9: Program Encrypt

Explain

Constants: PRF keys K_2, and K_3.
Input: Message m, ciphertext c, randomness r.

1. Set $\alpha = F_2(K_2, (m, c, \text{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, c, \text{PRG}(r))$.
   Output $e = (\alpha, \beta)$.

Figure 10: Program Explain

Hybrid 4. In this hybrid, the Encrypt and Explain programs are modified as shown below. In Lemma 2, (proven below after all hybrids are given), we argue that except with negligible probability over choice of constants, these modifications do not alter the functionality of either program.

Thus, the indistinguishability of Hybrid 3 and Hybrid 4 follows from the $i\mathcal{O}$ security property.

Explainability Game.

1. Choose K_1, K_2, K_3 at random.
2. Let $(\text{PK}, \text{SK}) \leftarrow \mathsf{Setup}_{PKE}$
3. Select u^* at random. Select $\tilde{r}$ at random.
4. Let x^* be chosen randomly. Let $c^* = \mathsf{Encrypt}_{PKE}(\text{PK}, m^*; x^*)$.
5. Set $\alpha^* = F_2(K_2, (m^*, c^*, \tilde{r}))$. Let $\beta^* = F_3(K_3, \alpha^*) \oplus (m^*, c^*, \tilde{r})$.
   Set $e^* = (\alpha^*, \beta^*)$.
6. Let $P_{\text{enc}} = i\mathcal{O}(\text{Encrypt})$. Let $P_{\text{explain}} = i\mathcal{O}(\text{Explain})$.
7. Flip coin $g \in \{0, 1\}$.
8. If $g = 0$, output $(P_{\text{enc}}, P_{\text{explain}}, u^*, c^*)$. If $g = 1$, output $(P_{\text{enc}}, P_{\text{explain}}, e^*, c^*)$.

Encrypt

Constants: m^*, u^*, e^*, c^*; Public key PK and PRF keys $K_1(\{(m^*, u^*), (m^*, e^*)\})$, K_2, and $K_3(\{u^*[1], e^*[1]\})$.
Input: Message m, randomness $u = (u[1], u[2])$.

1. If $(m, u) = (m^*, u^*)$ or (m^*, e^*), output c^* and stop.
2. If $u[1] = e^*[1]$ or $u[1] = u^*[1]$, then skip this step.
   If $F_3(K_3, u[1]) \oplus u[2] = (m', c', r')$ for (proper length) strings c', r', m',
   and $m' = m$,
   and $u[1] = F_2(K_2, (m', c', r'))$, then output $c = c'$ and end.
3. Else let $x = F_1(K_1, (m, u))$. Output $c = \mathsf{Encrypt}_{PKE}(\text{PK}, m; x)$.

Figure 11: Program Encrypt

Explain

Constants: PRF keys K_2, and $K_3(\{u^*[1], e^*[1]\})$.
Input: Message m, ciphertext c, randomness r.

1. Set $\alpha = F_2(K_2, (m, c, \text{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, c, \text{PRG}(r))$.
   Output $e = (\alpha, \beta)$.

Figure 12: Program Explain

Hybrid 5. In this hybrid, the value $e^*[2]$, denoted β^*, is chosen at random instead of being chosen as $\beta^* = F_3(K_3, \alpha^*) \oplus (m^*, c^*, \tilde{r})$. The indistinguishability of Hybrid 4 and Hybrid 5 follows immediately from the pseudorandomness property for the puncturable PRF F_3.

Explainability Game.

1. Choose K_1, K_2, K_3 at random.
2. Let $(\text{PK}, \text{SK}) \leftarrow \mathsf{Setup}_{PKE}$
3. Select u^* at random. Select $\tilde{r}$ at random.
4. Let x^* be chosen randomly. Let $c^* = \mathsf{Encrypt}_{PKE}(\text{PK}, m^*; x^*)$.
5. Set $\alpha^* = F_2(K_2, (m^*, c^*, \tilde{r}))$. Let β^* be random. Set $e^* = (\alpha^*, \beta^*)$.
6. Let $P_{\text{enc}} = i\mathcal{O}(\text{Encrypt})$. Let $P_{\text{explain}} = i\mathcal{O}(\text{Explain})$.
7. Flip coin $g \in \{0, 1\}$.
8. If $g = 0$, output $(P_{\text{enc}}, P_{\text{explain}}, u^*, c^*)$. If $g = 1$, output $(P_{\text{enc}}, P_{\text{explain}}, e^*, c^*)$.

Encrypt

Constants: m^*, u^*, e^*, c^*; Public key PK and PRF keys $K_1(\{(m^*, u^*), (m^*, e^*)\})$, K_2, and $K_3(\{u^*[1], e^*[1]\})$.
Input: Message m, randomness $u = (u[1], u[2])$.

1. If $(m, u) = (m^*, u^*)$ or (m^*, e^*), output c^* and stop.
2. If $u[1] = e^*[1]$ or $u[1] = u^*[1]$, then skip this step.
   If $F_3(K_3, u[1]) \oplus u[2] = (m', c', r')$ for (proper length) strings c', r', m',
   and $m' = m$,
   and $u[1] = F_2(K_2, (m', c', r'))$, then output $c = c'$ and end.
3. Else let $x = F_1(K_1, (m, u))$. Output $c = \mathsf{Encrypt}_{PKE}(\text{PK}, m; x)$.

Figure 13: Program Encrypt

Explain

Constants: PRF keys K_2, and $K_3(\{u^*[1], e^*[1]\})$.
Input: Message m, ciphertext c, randomness r.

1. Set $\alpha = F_2(K_2, (m, c, \text{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, c, \text{PRG}(r))$.
   Output $e = (\alpha, \beta)$.

Figure 14: Program Explain

Hybrid 6. In this hybrid, first we modify the Encrypt program to add a condition to the Step 2 check to determine if the decrypted $(m', c', r') = (m^*, c^*, \tilde{r})$, and if so, to skip this check. This new check does not change the functionality of the program, because $e^*[1] = F_2(K_2, (m^*, c^*, \tilde{r}))$, and therefore the check could not pass if $(m', c', r') = (m^*, c^*, \tilde{r})$, since Step 2 is skipped entirely if $u[1] = e^*[1]$. Then, both the Encrypt and Explain programs are modified to have keys K_2 punctured at the point $(m^*, c^*, \tilde{r})$. This puncturing does not change the functionality of the Encrypt program because of the new "if" condition just implemented. With high probability over the choice of $\tilde{r}$, it is true that $\tilde{r}$ is not in the image of the PRG, and therefore this puncturing also does not change the functionality of the Explain program.

Thus, the indistinguishability of Hybrid 5 and Hybrid 6 follows from the $i\mathcal{O}$ security property.

Explainability Game.

1. Choose K_1, K_2, K_3 at random.
2. Let $(\text{PK}, \text{SK}) \leftarrow \mathsf{Setup}_{PKE}$
3. Select u^* at random. Select $\tilde{r}$ at random.
4. Let x^* be chosen randomly. Let $c^* = \mathsf{Encrypt}_{PKE}(\text{PK}, m^*; x^*)$.
5. Set $\alpha^* = F_2(K_2, (m^*, c^*, \tilde{r}))$. Let β^* be random. Set $e^* = (\alpha^*, \beta^*)$.
6. Let $P_{\text{enc}} = i\mathcal{O}(\text{Encrypt})$. Let $P_{\text{explain}} = i\mathcal{O}(\text{Explain})$.
7. Flip coin $g \in \{0, 1\}$.
8. If $g = 0$, output $(P_{\text{enc}}, P_{\text{explain}}, u^*, c^*)$. If $g = 1$, output $(P_{\text{enc}}, P_{\text{explain}}, e^*, c^*)$.

Encrypt

Constants: $m^*, u^*, e^*, c^*, \tilde{r}$; Public key PK and PRF keys $K_1(\{(m^*, u^*), (m^*, e^*)\})$, $K_2(\{(m^*, c^*, \tilde{r})\})$, and $K_3(\{u^*[1], e^*[1]\})$.
Input: Message m, randomness $u = (u[1], u[2])$.

1. If $(m, u) = (m^*, u^*)$ or (m^*, e^*), output c^* and stop.
2. If $u[1] = e^*[1]$ or $u[1] = u^*[1]$, then skip this step.
   If $F_3(K_3, u[1]) \oplus u[2] = (m', c', r')$ for (proper length) strings c', r', m',
   and $m' = m$,
   and $(m', c', r') \neq (m^*, c^*, \tilde{r})$, then also check if $u[1] = F_2(K_2, (m', c', r'))$, then output $c = c'$ and end.
3. Else let $x = F_1(K_1, (m, u))$. Output $c = \mathsf{Encrypt}_{PKE}(\text{PK}, m; x)$.

Figure 15: Program Encrypt

Explain

Constants: PRF keys $K_2((m^*, c^*, \tilde{r}))$, and $K_3(\{u^*[1], e^*[1]\})$.
Input: Message m, ciphertext c, randomness r.

1. Set $\alpha = F_2(K_2, (m, c, \text{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, c, \text{PRG}(r))$.
   Output $e = (\alpha, \beta)$.

Figure 16: Program Explain

Hybrid 7. In our final hybrid, we modify $e^*[1]$, denoted α^*, to be randomly chosen, instead of being set as $\alpha^* = F_2(K_2, (m^*, c^*, \tilde{r}))$. The indistinguishability of Hybrid 6 and Hybrid 7 follows immediately from the pseudorandomness property of the puncturable PRF F_2.

Explainability Game.

1. Choose K_1, K_2, K_3 at random.
2. Let $(\text{PK}, \text{SK}) \leftarrow \mathsf{Setup}_{PKE}$
3. Select u^* at random. Select $\tilde{r}$ at random.
4. Let x^* be chosen randomly. Let $c^* = \mathsf{Encrypt}_{PKE}(\text{PK}, m^*; x^*)$.
5. Let α^* be random. Let β^* be random. Set $e^* = (\alpha^*, \beta^*)$.
6. Let $P_{\text{enc}} = i\mathcal{O}(\text{Encrypt})$. Let $P_{\text{explain}} = i\mathcal{O}(\text{Explain})$.
7. Flip coin $g \in \{0, 1\}$.
8. If $g = 0$, output $(P_{\text{enc}}, P_{\text{explain}}, u^*, c^*)$. If $g = 1$, output $(P_{\text{enc}}, P_{\text{explain}}, e^*, c^*)$.

Encrypt

Constants: $m^*, u^*, e^*, c^*, \tilde{r}$; Public key PK and PRF keys $K_1(\{(m^*, u^*), (m^*, e^*)\})$, $K_2(\{(m^*, c^*, \tilde{r})\})$, and $K_3(\{u^*[1], e^*[1]\})$.
Input: Message m, randomness $u = (u[1], u[2])$.

1. If $(m, u) = (m^*, u^*)$ or (m^*, e^*), output c^* and stop.
2. If $u[1] = e^*[1]$ or $u[1] = u^*[1]$, then skip this step.
   If $F_3(K_3, u[1]) \oplus u[2] = (m', c', r')$ for (proper length) strings c', r', m',
   and $m' = m$,
   and $(m', c', r') \neq (m^*, c^*, \tilde{r})$, then also check if $u[1] = F_2(K_2, (m', c', r'))$, then output $c = c'$ and end.
3. Else let $x = F_1(K_1, (m, u))$. Output $c = \mathsf{Encrypt}_{PKE}(\text{PK}, m; x)$.

Figure 17: Program Encrypt

Explain

Constants: PRF keys $K_2(\{(m^*, c^*, \tilde{r})\})$, and $K_3(\{u^*[1], e^*[1]\})$.
Input: Message m, ciphertext c, randomness r.

1. Set $\alpha = F_2(K_2, (m, c, \text{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, c, \text{PRG}(r))$.
   Output $e = (\alpha, \beta)$.

Figure 18: Program Explain

Final security argument. Looking at Hybrid 7, we observe that the variables e^*, u^* are now both independently uniformly random strings, and they are treated entirely symmetrically. (Recall that the "if" statements above have the conditions written in lexicographic order, so they do not reveal any asymmetry between e^* and u^*.) Thus, the distributions output by this Hybrid for $g = 0$ and $g = 1$ are identical, and therefore even an unbounded adversary could have no advantage in distinguishing them. ∎

The proof above made use of the following lemma for arguing that the programs obfuscated by the indistinguishability obfuscator in Hybrid 4 are equivalent to the programs in Hybrid 5.

Lemma 2. Except with negligible probability over the choice of $u^*[1]$ and $e^*[1]$, the Encrypt and Explain programs in Hybrid 4 are equivalent to the Encrypt and Explain programs in Hybrid 3.

Proof. We consider each change to the programs below.

First, an "if" statement is added to Step 2 of the Encrypt program, to skip the Step 2 check if either $u[1] = e^*[1]$ or $u[1] = u^*[1]$. To see why this change does not affect the functionality of the program, let us consider each case in turn. Observe that by Lemma 1, if $u[1] = e^*[1]$, then the only way the Step 2 check can be satisfied is if $m = m^*$ and $u[2] = e^*[2]$. But this case is already handled in Step 1, therefore skipping Step 2 if $u[1] = e^*[1]$ does not affect functionality. On the other hand, recall that $u^*[1]$ is chosen at random, and therefore the probability that $u^*[1]$ would be in the image of $F_2(K_2, \cdot)$ is negligible, therefore with high probability over the choice of constant $u^*[1]$, the Step 2 check cannot be satisfied if $u[1] = u^*[1]$. Therefore, the addition of this "if" statement does not alter the functionality of the Encrypt program.

Also, in this hybrid, the key K_3 is punctured at $\{u^*[1], e^*[1]\}$ in both the Encrypt and Explain programs. The new "if" statement above ensures that $F_3(K_3, \cdot)$ is never called at these values in the Encrypt program. Recall that the Explain program only calls $F_3(K_3, \cdot)$ on values computed as $F_2(K_2, (m, c, \mathrm{PRG}(r)))$ for some bit m and strings c and r. Furthermore, F_2 is statistically injective with a very sparse image set, by our choice of parameters. Since $u^*[1]$ is randomly chosen, it is very unlikely to be in the image of $F_2(K_2, \cdot)$. Since $e^*[1]$ is chosen based on a random $\tilde{r}$ value instead of a PRG output, it is very unlikely to correspond to $F_2(K_2, (m, c, \mathrm{PRG}(r)))$ for any (m, c, r). Thus, these values are not called by the Explain program, except with negligible probability over the choice of these constants $u^*[1]$ and $e^*[1]$. ■

4.2 Universal Deniable Encryption

In some applications of deniable encryption, we might want to deniably encrypt to a receiver that has established a public key for a standard encryption system such as ElGamal or RSA. Moreover, the receiver might be oblivious of a sender's desire to deniably encrypt or disinterested in investing any additional resources to enable it. For example, we could imagine a user that wishes to make their encrypted SSL transmissions to a webservice deniable.

In this section we sketch an extension to our construction that we call universal deniable encryption. In a universal deniable there is a one time setup process by a trusted party. (We assume that this trusted party will never be forced to disclose its randomness.) The party runs an algorithm $\mathsf{UniversalSetup}(1^\lambda, B)$ that takes as input a security parameter and a positive integer B which serves as a bound (to be described momentarily). We let $U_B(\cdot)$ denote a universal circuit that accepts any a family of circuits that can be described by B bits. For simplicity we assume all encryptions are single bit messages.

The UniversalSetup algorithm creates obfuscations of the program Universal Encrypt of Figure 19 and universal explain of Figurefig:universal-deniable-explain. The pair of these are output as a common reference string.

Suppose that we want to deniably encrypt to a user that uses a public key PK that belongs to an encryption system with encrypt algorithm $\mathsf{Encrypt}(\mathrm{PK}, m; x)$, which takes in the three inputs of a public key, a message and a randomness. If we "hardwire" in the public key, this defines a program $p(m; x)$, that takes in two inputs. A single bit message and randomness x. To encrypt we can simply call the obfuscation of program Universal Encrypt with inputs p, m and random u. The obfuscated program will use a universal circuit U_B to run the program p (as long as p falls in the circuit class). The Universal Explain program works in an analogous way for deniability. The receiver can decrypt the message without even being aware that the ciphertext was generated in such a manner.

This construction highlights an important feature of our original construction. Namely, that in our approach the setup of a deniable encryption system can be strongly separated from the choosing of a public/private key pair for the underlying semantically secure public-key encryption. In particular, knowledge of the secret key is not required. The universal deniable encryption system takes this one step further, and allows for the public encryption system to be defined as a parameter of the obfuscated programs, as opposed to being built into the obfuscated programs.

Universal Encrypt

Constants: Public key PK and keys K_1, K_2, and K_3.
Input: Message m, encryption program p, randomness $u = (u[1], u[2])$.

1. If $F_3(K_3, u[1]) \oplus u[2] = (m', p', c', r')$ for (proper length) strings c', r', m', p', and $m' = m$, $p' = p$
   and $u[1] = F_2(K_2, (m', p, c', r'))$, then output $c = c'$ and end.
2. Else let $x = F_1(K_1, (m, p, u))$. Output $c = U_B(p, m; x)$.

Figure 19: Program Universal Encrypt

Universal Explain

Constants: keys K_2 and K_3.
Input: Message m, encryption program p, ciphertext c, randomness $r \in \{0,1\}^\lambda$.

1. Set $\alpha = F_2(K_2, (m, p, c, \mathrm{PRG}(r)))$. Let $\beta = F_3(K_3, \alpha) \oplus (m, p, c, \mathrm{PRG}(r))$.
   Output $e = (\alpha, \beta)$.

Figure 20: Program Universal Explain

5 Core Primitives from Indistinguishability Obfuscation

In this section we show how to build up core cryptographic primitives from Indistinguishable Obfuscation. The primitives we build are: public key encryption, short "hash-and-sign" selectively secure signatures, chosen-ciphertext secure public key encryption, non-Interactive zero knowledge proofs (NIZKs), injective trapdoor functions, and oblivious transfer. We show that each of these can be built from Indistinguishability Obfuscation and (using known implications [GGM84, HILL99]) standard one way functions, with the exception of trapdoor functions which requires an injective one way function.

Interestingly, the use of punctured programmability emerges as a repeated theme in building each of these. In addition, techniques used to realize one primitive will often have applications in another. For example, our NIZK system uses an obfuscated program that takes as input an instance and witness and outputs a signature on the instance if the witness relation holds. Internally, it leverages the structure of our signature scheme.

For compactness we refer the reader to the literature (e.g. [Gol01]) for the definitions of these core primitives. For each primitive we will present our construction and proof of security. In the proof of security we will give a detailed description of a sequence of hybrid arguments. Once the hybrids are established arguing indistinguishability between them is fairly straightforward and we sketch these arguments.

5.1 Public Key Encryption

We note that the implication that public-key encryption follows from indistinguishability obfuscation and one-way functions already follows from claims in the literature: Indistinguishability obfuscation implies Witness Encryption [GGH$^+$13a]; Witness Encryption and one-way functions imply public-key encryption [GGH$^+$13b]. However, that chain of implications, if followed, would result in a public-key encryption scheme where ciphertexts are themselves obfuscated programs. Our approach, on the other hand, yields public-key encryption where public keys are obfuscated programs, and ciphertexts are short.

Let PRG be a pseudo random generator that maps $\{0,1\}^\lambda$ to $\{0,1\}^{2\lambda}$. Let F be a puncturable PRF that takes inputs of 2λ bits and outputs ℓ bits. We describe our encryption algorithm which encrypts for the message space $\mathcal{M} = \{0,1\}^\ell$.

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm first chooses a puncturable PRF key K for F. Next, it creates an obfuscation of the program PKE Encrypt of Figure 21. The size of the program is padded to be the

maximum of itself and PKE Encrypt* of Figure 22. The public key, PK, is the obfuscated program. The secret key SK is K.

- $\mathsf{Encrypt}(\mathrm{PK}, m \in \mathcal{M})$: The encryption algorithm chooses a random value $r \in \{0,1\}^\lambda$ and runs the obfuscated program of PK on inputs m, r.

- $\mathsf{Decrypt}(\mathrm{SK}, c = (c_1, c_2))$ The decryption algorithm outputs $m' = F(K, c_1) \oplus c_2$.

PKE Encrypt

Constants: Punctured PRF key K.
Input: Message $m \in \{0,1\}^\ell$, randomness $r \in \{0,1\}^\lambda$.

1. Let $t = PRG(r)$.
2. Output $c = (c_1 = x, c_2 = F(K, t) \oplus m)$.

Figure 21: Program PKE Encrypt

PKE Encrypt*

Constants: Punctured PRF key $K(\{t^*\})$.
Input: Message $m \in \{0,1\}^\ell$, randomness $r \in \{0,1\}^\lambda$.

1. Let $t = PRG(r)$.
2. Output $c = (c_1 = x, c_2 = F(K, t) \oplus m)$.

Figure 22: Program PKE Encrypt*

Theorem 4. If our obfuscation scheme is indistingishuably secure, PRG is a secure pseudo random generator, and F is a secure punctured PRF, then our PKE scheme is IND-CPA secure.

Proof. We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original IND-CPA security game. We prove that the attacker's advantage must be the negligibly close between each successive hybrid and that the attacker has zero advantage in the final experiment.

- Hyb_0 : In the first hybrid the following game is played.
  1. $r^* \in \{0,1\}^\lambda$ is chosen at random and $t^* = \mathrm{PRG}(r^*)$.
  2. K is chosen as a key for the puncturable PRF.
  3. The public key given out is an obfuscation of the program PKE Encrypt.
  4. The attacker receives PK and then gives $m_0, m_1 \in \{0,1\}^\ell$ to the challenger.
  5. The challenge ciphertext is $c^* = (c_1^* = t^*, c_2^* = F(K, t^*) \oplus m_b)$ where $b \in \{0,1\}$ is chosen randomly.
- Hyb_1 : Is the same as Hyb_0 with the exception that t^* is chosen randomly in $\{0,1\}^{2\lambda}$. Note that r^* is no longer in the attacker's view and does not need to be generated.
- Hyb_2 : Is the same as Hyb_1 except that the public key is created as an obfuscation of the program PKE Encrypt* of Figure 22. Note that with all but negligible probability t^* is not in the image of the PRG.
- Hyb_3 : Is the same as Hyb_2 except the challenge ciphertext is given as $(c_1^* = t^*, c_2^* = z^*)$ for random z^*.

We first argue that the advantage of any poly-time attacker in guessing the bit b in Hyb_1 must be negligibly close to the attacker's advantage in Hyb_0. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the security of the pseudorandom generator. $\mathcal{B}$ runs as the challenger of the IND-CPA security game and takes in a PRG challenge a and sets $t^* = a$. It then runs the rest of the experiment. If a is the output of a PRG, then we are in Hyb_0. If a is chosen as a random string, then we are in Hyb_1. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on PRG security.

Next, we argue that the advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_1 and Hyb_2. We first observe that with all but negligible probability that the input/output behavior of programs PKE Encrypt and PKE Encrypt* are identical when t^* is chosen at random. The reason is that with probability $1 - 1/2^\lambda$, t^* is not in the image on the PRG. Thus, with high probability for all inputs neither program can call on $F(K, t^*)$. Therefore, puncturing t^* out from the key K will not effect input/output behavior. Therefore, if there is a difference in advantage, we can create an algorithm $\mathcal{B}$ that breaks indistinguishability security for obfuscation. $\mathcal{B}$ runs as the challenger, but where t^* is chosen at random. When it is to create the obfuscated program it submits both programs PKE Encrypt and PKE Encrypt* to an $i\mathcal{O}$ challenger. If the $i\mathcal{O}$ challenger chooses the first then we are in Hyb_1. If it chooses the second then we are in Hyb_2. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on $i\mathcal{O}$ security.

We now argue that the advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_2 and Hyb_3. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the *selective* security of the constrained pseudorandom function at the punctured points. $\mathcal{B}$ runs the security game of Hyb_2, except that it gets gets the punctured PRF key $K(\{t^*\})$ and challenge a. It continues to run as in Hyb_2 except that it creates the challenge ciphertext as $(t^*, a \oplus m_b)$. If a is the output of the PRF at point t^*, then we are in Hyb_2. If it was chosen uniformly at random, we are in Hyb_3. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on the constrained PRF security. Note, that we were able to reduce to selective security since t^* is defined to be chosen at random by the challenger and outside the attacker's control and was chosen before the punctured PRF key was received.

Finally, we observe that any attacker's advantage in Hyb_3 must be 0, since it conveys no information about b. Since the advantage of all poly-time attacker's are negligibly close in each successive hybrid, this proves IND-CPA security.

■

5.2 Short Signatures

We can give a "short" or "hash and sign"[3] signature scheme that signs ℓ bit messages and prove it to be *selectively* secure. (One can argue adaptive security via the usual complexity leveraging argument.) Our signatures are essentially the classical PRF MAC. (I.e. A signature on m is $\sigma = F(K, m)$). To verify, we create an obfuscated program that checks the MACs.

The technical challenge is to ensure that the symmetric key type verification material in the obfuscated program cannot be used to forge a new signature. A natural direction is to simply have the obfuscated program evaluate the PRF itself and check for equality. However, in the proof there would be no way to excise out the information about $F(K, m^*)$ (for challenge message m^*) without making the program incapable of verifying for m^*. We resolve this by making the verification compute $f(F(K, m))$ and compare the result for equality to to $f(\sigma)$, where f is a one way function and σ is the claimed signature on m.

Let F be a puncturable PRF that takes inputs of ℓ bits and outputs λ bits. Let $f(\cdot)$ be a one way function. The message space is $\mathcal{M} = \{0,1\}^\ell$.

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm first chooses a puncturable PRF key K for F. Next, it creates an obfuscation of the Verify Signature of Figure 23. The size of the program is padded to be the maximum of itself and Program Verify Signature* of Figure 24. The verification key, VK, is the obfuscated program. The secret key SK is K.

[3]The term "hash and sign" has evolved to roughly mean a signature that does not use a tree-based structure like that of Goldwasser-Micali-Rivest [GMR88].

- $\mathsf{Sign}(\mathrm{SK}, m \in \mathcal{M})$: The signature algorithm outputs $\sigma = F(K, m)$.

- $\mathsf{Verify}(\mathrm{VK}, m, \sigma)$ The verification algorithm runs the obfuscated program VK on inputs m, σ.

Verify Signature

Constants: PRF key K.
Input: Signature message m, σ.

1. Test if $f(\sigma) = f(F(K, m))$. Output accept if true, reject if false.

Figure 23: Program Verify Signature

Verify Signature*

Constants: PRF key $K(\{m^*\})$ and values $m^* \in \{0,1\}^\ell, z^*$.
Input: Signature message m, σ.

1. If $m = m^*$, test if $f(\sigma) = z^*$. Output accept if true, reject otherwise.
2. Else, test if $f(\sigma) = f(F(K, m))$. Output accept if true, reject if false.

Figure 24: Program Verify Signature*

Theorem 5. If our obfuscation scheme is indistingishuably secure, F is a secure punctured PRF, and $f(\cdot)$ is a one way function, then our signature scheme is existentially unforgeable under chosen message attack.

Proof. We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. We prove that a poly-time attacker's advantage must be negligibly close between each successive one. Then, we show that any poly-time attacker in the final experiment that succeeds in forging with non-negligible probability can be used to break one way functions.

- Hyb_0 : In the first hybrid the following game is played.
  1. The attacker selectively gives the challenger the message m^*.
  2. K is chosen as a key for the puncturable PRF.
  3. The public key VK is given out as an obfuscation of the program Verify Signature.
  4. The attacker queries the sign oracle a polynomial number of times on messages $m \neq m^*$. It receives back $F(K, m)$.
  5. The attacker sends a forgery σ^* and wins if $\mathsf{Verify}(m^*, \sigma^*) = 1$.
- Hyb_1 : Is the same as Hyb_0 except we let $z^* = f(F(K, m^*))$ and let VK be the obfuscation of the program Verify Signature* of Figure 24.
- Hyb_2 : Is the same as Hyb_1 except $z^* = f(t)$ for t chosen uniformly at random in $\{0,1\}^\lambda$.

First, we argue that the advantage for any poly-time attacker in forging a signature must be negligibly close in hybrids Hyb_0 and Hyb_1. We first observe that the input/output behavior of programs Verify Signature and Verify Signature* are identical. The only difference is that the first program computes $F(K, m^*)$ before applying the OWF f to it for message input m^*, whereas the second is given $f(F(K, m^*))$ as the constant z^*. Therefore, if there is a difference in advantage we can create an algorithm $\mathcal{B}$ that breaks indistinguishability security for obfuscation. $\mathcal{B}$ runs as the challenger. When it is to create the obfuscated program it submits both programs Verify Signature and Verify Signature* to the $i\mathcal{O}$ challenger. It sets VK to the program returned by the challenger. If the $i\mathcal{O}$ challenger chooses the first, then we are in Hyb_0. If it chooses the second, then we are in Hyb_1. $\mathcal{B}$ will output 1 if the attacker successfully forges. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on $i\mathcal{O}$ security. *We note that for Verify Signature* to be*

well defined, the value m^* must be well defined. By the time the obfuscated program is created, the selective attacker has already declared m^*.

We now argue that the advantage for any poly-time attacker in forging a signature must be negligibly close in hybrids Hyb_1 and Hyb_2. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the *selective* security of the constrained pseudorandom function at the punctured points. $\mathcal{B}$ first gets m^* selectively from the attacker. It submits this to the constrained PRF challenger and receives the punctured PRF key $K(\{m^*\})$ and challenge a. It continues to run the experiment of Hyb_1 except it sets $z^* = f(a)$. If a is the output of the PRF at point m^*, then we are in Hyb_1. If it was chosen uniformly at random, then we are in Hyb_2. $\mathcal{B}$ will output 1 if the attacker successfully forges. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on the constrained PRF security. Here we were able to reduce to selective security since the attacker was defined to be selective.

Finally, if there is an attacker in Hyb_2, we can use it to break the security of the OWF. We build a reduction $\mathcal{B}$ that first takes in m^* selectively and receives y as the challenge for a OWF and sets $z^* = y$. If an attacker successfully forges on m^*, then by definition he has computed a σ such that $f(\sigma) = z^*$. Therefore, if the OWF is secure, no poly-time attacker can forge with non-negligible advantage. Since the advantage of all poly-time attacker's are negligibly close in each successive hybrid, this proves selective security for the signature scheme.

■

Remark 3. If we use an *injective* one way function for $f(\cdot)$, then the scheme is strongly unforgeable.

5.3 Chosen Ciphertext Secure Public Key Encryption (for bit encryption)

Let PRG be a pseudo random generator that maps $\{0,1\}^\lambda$ to $\{0,1\}^{2\lambda}$. Let F_1 be a puncturable PRF that takes inputs of 2λ bits and outputs a single bit. Let F_2 be a punctured PRF that takes as input $2\lambda + 1$ bits and outputs λ bits.

- $\mathsf{Setup}(1^\lambda)$: The setup algorithm first chooses a puncturable PRF key K_1 for F_1 and K_2 for F_2. Next, it creates an obfuscation of the program CCA-PKE Encrypt of Figure 25. The size of the program is padded to be the maximum of itself and CCA-PKE Encrypt* of Figure 26. The public key, PK, is the obfuscated program. The secret key SK is (K_1, K_2).
- $\mathsf{Encrypt}(\mathrm{PK}, m \in \{0,1\})$: The encryption algorithm chooses a random value $r \in \{0,1\}^\lambda$ and runs the obfuscated program of PK on inputs $m \in \{0,1\}$ and $r \in \{0,1\}^\lambda$.
- $\mathsf{Decrypt}(\mathrm{SK}, c = (c_1, c_2, c_3))$ The decryption algorithm first checks if $c_3 = F_2(K_2, c_1|c_2)$. If not, it outputs reject and stops. Otherwise, it outputs $m' = F_1(K_1, c_1) \oplus c_2$.

CCA-PKE Encrypt

Constants: Punctured PRF keys K_1, K_2.
Input: Message $m \in \{0,1\}$, randomness $r \in \{0,1\}^\lambda$.

1. Let $t = PRG(r)$
2. Output $c = (c_1 = t, c_2 = F_1(K_1, t) \oplus m, c_3 = F_2(K_2, c_1|c_2))$

Figure 25: Program CCA-PKE Encrypt

Theorem 6. If our obfuscation scheme is indistingishuably secure, PRG is a secure pseudo random generator, and F_1, F_2 are a secure punctured PRFs, then our encryption scheme is IND-CCA secure.

Proof. We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original IND-CCA security game. We prove that the attacker's advantage must be the same between each successive one and that the attacker has zero advantage in the final experiment.

CCA-PKE Encrypt*

Constants: Punctured PRF keys $K_1(\{t^*\}), K_2(\{t^*|0, t^*|1\})$.
Input: Message $m \in \{0,1\}$, randomness $r \in \{0,1\}^\lambda$.

1. Let $t = PRG(r)$
2. Output $c = (c_1 = t, c_2 = F_1(K_1, t) \oplus m, c_3 = F_2(K_2, c_1|c_2))$

Figure 26: Program CCA-PKE Encrypt*

- Hyb_0 : In the first hybrid the following game is played.
  1. $r^* \in \{0,1\}^\lambda$ is chosen at random and $t^* = \text{PRG}(r^*)$.
  2. K_1, K_2 are chosen as a key for the puncturable PRF.
  3. The public key given out is an obfuscation of the program CCA-PKE Encrypt.
  4. The attacker receives PK.
  5. The attacker makes queries for ciphertexts c to the decryption oracle.
  6. The challenge ciphertext is $c^* = (c_1^* = t^*, c_2^* = F(K, t^*) \oplus b, c_3^* = F_2(K_2, c_1^*|c_2^*))$ and is given to the attacker.
  7. The attacker makes queries for ciphertexts $c \neq c^*$ to the decryption oracle. $b \in \{0,1\}$ is chosen randomly.
- Hyb_1 : Is the same as Hyb_0 with the exception that t^* is chosen randomly in $\{0,1\}^{2\lambda}$. Note that r^* is no longer in the attacker's view and does not need to be generated.
- Hyb_2 : Is the same as Hyb_1 except the challenger rejects any phase 1 decryption queries where $c_1 = c_1^*$.
- Hyb_3 : Is the same as Hyb_2 except that the public key is created as an obfuscation of the program CCA-PKE Encrypt* of Figure 26.
- Hyb_4 : Is the same as Hyb_3 except that the challenger replaces $F_2(K_2, t^*|0)$ and $F_2(K_2, t^*|1)$ with two values chosen uniformly at random for ciphertext queries and creating the challenge ciphertext.
- Hyb_5 : Is the same as Hyb_3 except the challenge ciphertext is given as $(c_1^* = t^*, c_2^* = z^*, c_3^*)$ for random $z^* \in \{0,1\}$.

We first argue that the advantage of any poly-time attacker in guessing the bit b in Hyb_1 must be negligibly close to the attacker's advantage in Hyb_0. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the security of the pseudorandom generator. $\mathcal{B}$ runs as the challenger of the security game and take in a PRG challenge a and sets $t^* = a$. It then runs the rest of the experiment. If a is the output of a PRG, then we are in Hyb_0. If a is chosen as a random string, then we are in Hyb_1. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on PRG security. Note that the reduction algorithm has keys K_1, K_2 to answer all decryption queries.

The advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_1 and Hyb_2. The reason is that t^* is chosen randomly. If an attacker makes q queries, then the chances of $c_1 = t^*$ is at most $q/2^\lambda$.

Next, we argue that the advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_2 and Hyb_3. We first observe that with all but negligible probability that the input/output behavior of programs CCA-PKE Encrypt and CCA-PKE Encrypt* are identical when t^* is chosen at random. The reason is that with probability $1 - 1/2^\lambda$, t^* is not in the image on the PRG and neither program will call $F_1(K_1, t^*)$ nor $F_2(K_2, t^*|0)$, nor $F_2(K_2, t^*|1)$. Therefore puncturing out these values from the key will not make a difference in input/output behavior. Therefore, if there is a difference in advantage we can create an algorithm $\mathcal{B}$ that breaks indistinguishability security for obfuscation. $\mathcal{B}$ runs as the challenger, but where

t^* is chosen at random. When it is to create the obfuscated program it submits both programs CCA-PKE Encrypt and CCA-PKE Encrypt*. It sets PK to be the obfuscated program returned by the $i\mathcal{O}$ challenger. If the $i\mathcal{O}$ challenger chooses the first, then we are in Hyb_2. If it chooses the second, then we are in Hyb_3. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on $i\mathcal{O}$ security. Note that the reduction algorithm has keys K_1, K_2 to answer all decryption queries.

We now argue that the advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_3 and Hyb_4. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the *selective* security of the constrained pseudorandom function at the punctured points. $\mathcal{B}$ runs where it picks t^* randomly first and then gets the punctured PRF key $K_2(\{t^*|0, t^*|1\})$ and challenge a_0, a_1. $\mathcal{B}$ creates runs the experiment as in Hyb_3 except it whenever $F_2(K_2, t^*|0)$ is called it uses a_0 and whenever $F_2(K_2, t^*|1)$ is called it uses a_1. (One of these substations will be used in creating the challenger ciphertext and the other might be used in answering a decryption query.) If a_0, a_1 are the outputs of the PRF at points $t^*|0$ and $t^*|1$, then we are in Hyb_3. If it was chosen uniformly at random, we are in Hyb_4. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on the constrained PRF security. Note, that we were able to reduce to selective security since t^* is defined to be chosen at random by the challenger and outside the attacker's control.

We now argue that the advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_4 and Hyb_5. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the *selective* security of the constrained pseudorandom function at the punctured points. $\mathcal{B}$ runs where it picks t^* randomly first and then gets the punctured PRF key $K_1(\{t^*\})$ and challenge a. It creates the punctured PRF $K_2(\{t^*|0, t^*|1\})$ itself. It continues as in Hyb_4 except it creates the challenge ciphertext as $(c_1^* = t^*, c_2^* = a \oplus m_b, c_3^*)$ (for randomly chosen c_3^*). If a is the output of the PRF at point t^*, then we are in Hyb_4. If it was chosen uniformly at random, then we are in Hyb_5. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on the constrained PRF security. Note, that we were able to reduce to selective security since t^* is defined to be chosen at random by the challenger and outside the attacker's control. *In addition, we note that the punctured keys $K_1(\{t^*\})$ and $K_2(\{t^*|0, t^*|1\})$ are enough to answer all allowed decryption queries in hybrids Hyb_4 and Hyb_5 except when $c_1 = c_1^*, c_2 = 1 - c_2^*$. However, in this case with overwhelming probability c_3 will not be equal to the random value chosen to replace $F_2(K_2, t^*|c_2^* - 1)$, so the reduction algorithm can reject. If $c_1 = c_1^*, c_2 = c_2^*$, then simply reject as the only accepted c_3 value is c_3^*. However, this is the challenge ciphertext, which is not allowed to be queried.*

Finally, we observe that any attacker's advantage in Hyb_5 must be 0, since it conveys no information about b. Since the advantage of all poly-time attacker's are negligibly close in each successive hybrid, this proves IND-CCA security.

■

The above cryptosystem is only designed to encrypt a single bit message. To decrypt multi-bit messages we have several options. First, we could simply apply the known single bit to multi bit transformation of Myers and Shelat [MS09]. It is also interesting to examine how to handle this without applying outside results. A natural option is to run the same system, but just let m be multiple bits (and adjust the PRF domains and ranges accordingly). The biggest issue with this is that when proving security the CCA-PKE Encrypt* program K_2 should have punctured out all inputs of the form $(t^*|y)$ where y is in the message space. This grows exponentially in the length of y. One solution is to use a different form of confined PRFs that succinctly allows the program to evaluate all PRF inputs *except* those with prefix t^*. These forms of PRFs are also realizable via the GGM [GGM84] tree and thus derivable from one way functions. Finally, in the next subsection we will see a very simple CCA-secure Key Encapsulation Mechanism (KEM). This is arguably preferable to these more complicated alternatives listed above.

5.4 Chosen Ciphertext Secure Public Key Encapsulation Mechanism (KEM).

We now show a chosen ciphertext secure key encapsulation mechanism (KEM) which is essentially as simple our the public key encryption scheme. There exist known transformations [CS04] for CCA-KEMs to CCA-

secure encryption.

Let PRG be a pseudo random generator that maps $\{0,1\}^\lambda$ to $\{0,1\}^{2\lambda}$. Let F be a puncturable PRF that takes inputs of 2λ bits and outputs λ bits.

- $\mathsf{KEMSetup}(1^\lambda)$: The setup algorithm first chooses a puncturable PRF key K for F. Next, it creates an obfuscation of the program CCA-KEM Encrypt of Figure 27. The size of the program is padded to be the maximum of itself and CCA-KEM* of Figure 28. The public key, PK, is the obfuscated program. The secret key SK is K.
- $\mathsf{KEMEncrypt}(\mathrm{PK})$: The encryption algorithm chooses a random value $r \in \{0,1\}^\lambda$ and runs the obfuscated program of PK on inputs r. It receives back the ciphertext c and a symmetric key k.
- $\mathsf{KEMDecrypt}(\mathrm{SK}, c)$ The KEM-decrypt algorithm computes the symmetric key $k = F(K, c)$.

CCA-KEM Encrypt

Constants: Punctured PRF keys K.
Input: Randomness $r \in \{0,1\}^\lambda$.

1. Let $t = PRG(r)$
2. Output $c = t, k = F(K, t)$.

Figure 27: Program CCA-KEM Encrypt

CCA-KEM Encrypt*

Constants: Punctured PRF keys $K(\{t^*\})$.
Input: Randomness $r \in \{0,1\}^\lambda$.

1. Let $t = PRG(r)$
2. Output $c = t, k = F(K, t)$.

Figure 28: Program CCA-KEM Encrypt*

Theorem 7. If our obfuscation scheme is indistingishuably secure, PRG is a secure pseudo random generator, and F is a secure punctured PRF, then our key encapsulation mechanism scheme is chosen ciphertext secure.

Proof. We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original IND-CCA security game. We prove that the attacker's advantage must be the same between each successive one and that the attacker has zero advantage in the final experiment.

- Hyb_0 : In the first hybrid the following game is played.
  1. $r^* \in \{0,1\}^\lambda$ is chosen at random and $t^* = \mathrm{PRG}(r^*)$.
  2. K is chosen as a key for the puncturable PRF.
  3. The public key given out is an obfuscation of the program CCA-KEM Encrypt.
  4. The attacker receives PK.
  5. The attacker makes queries for ciphertexts c to the KEM decryption oracle and receives the responses.
  6. The challenger flips a random coin $b \in \{0,1\}$. The challenge ciphertext is $c^* = t^*$. The challenger returns the ciphertext along with k_b, where $k_0 = F(K, t^*)$ and k_1 is chosen at random.
  7. The attacker makes queries for ciphertexts $c \neq c^*$ to the decryption oracle.

- Hyb_1 : Is the same as Hyb_0 with the exception that t^* is chosen randomly in $\{0,1\}^{2\lambda}$. Note that r^* is no longer in the attacker's view and does not need to be generated.
- Hyb_2 : Is the same as Hyb_1 except the challenger rejects any phase 1 decryption queries where $c = c^*$.
- Hyb_3 : Is the same as Hyb_2 except that the public key is created as an obfuscation of the program CCA-PKE Encrypt* of Figure 28.
- Hyb_4 : Is the same as Hyb_3 except that k_0 is chosen uniformly at random.

We first argue that the advantage of any poly-time attacker in guessing the bit b in Hyb_1 must be negligibly close to the attacker's advantage in Hyb_0. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the security of the pseudorandom generator. $\mathcal{B}$ runs as the challenger of the security game and takes in a PRG challenge a and sets $t^* = a$. It then runs the rest of the experiment. If a is the output of a PRG, then we are in Hyb_0. If a is chosen as a random string, then we are in Hyb_1. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on PRG security. All decryption queries can be answered since the reduction knows the key K.

The advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_1 and Hyb_2. The reason is that t^* is chosen randomly. If an attacker makes q queries, then the chances of $c_1 = t^*$ is at most $q/2^\lambda$.

Next, we argue that the advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_2 and Hyb_3. We first observe that with all but negligible probability that the input/output behavior of programs CCA-KEM Encrypt and CCA-KEM Encrypt* are identical when t^* is chosen at random. The reason is that with probability $1 - 1/2^\lambda$, t^* is not in the image of the PRG. Therefore neither program will evaluate $F(K, t^*)$. Thus, puncturing excising t^* out from the key will not make a difference in input/output behavior. Therefore, if there is a difference in advantage we can create an algorithm $\mathcal{B}$ that breaks indistinguishability security for obfuscation. $\mathcal{B}$ runs as the challenger, but where t^* is chosen at random. When it is to create the obfuscated program it submits both programs CCA-KEM Encrypt and CCA-KEM Encrypt*. It set PK to be the obfuscated program returned from the challenger. If the $i\mathcal{O}$ challenger chooses the first then we are in Hyb_2. If it chooses the second then we are in Hyb_3. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on $i\mathcal{O}$ security. All decryption queries can be answered since the reduction knows the key K.

We now argue that the advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_3 and Hyb_4. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the *selective* security of the constrained pseudorandom function at the punctured points. $\mathcal{B}$ runs the experiment where it picks t^* randomly first. Then, it receives the punctured PRF key $K(\{t^*\})$ and challenge a from the constrained PRF challenger. It continues as in Hyb_2 except it creates $k_0 = a$. If ia was chosen uniformly at random, we are in Hyb_4, otherwise we are in hybrid Hyb_3. $\mathcal{B}$ will output 1 if $b' = b$. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on the constrained PRF security. Note, that we were able to reduce to selective security since t^* is defined to be chosen at random by the challenger and outside the attacker's control. All decryption queries for $c \neq c^* = t^*$ can be answered since the reduction knows the key $K(t^*)$. The query $c = c^*$ is disallowed by the security game.

Finally, we observe that any attacker's advantage in Hyb_4 must be 0, since it conveys no information about b given that both k_0 and k_1 are chosen at random. Since the advantage of all poly-time attacker's are negligibly close in each successive hybrid, this proves CCA-KEM security.

■

5.5 Non-Interactive Zero Knowledge Proofs with Perfect Zero Knowledge

We give a Non-Interactive Zero Knowledge Proof (NIZK) system for any NP language with statements up to a bounded size. The system consists of two obfuscated programs. The first is a proving algorithm that takes as input an instance x and witness w. It outputs a signature on x if $R(x, w) = 1$. The signature is of

the form of our short signature scheme. Verification is done by an obfuscation of signature verification and is virtually identical to our short signature verification algorithm.

An interesting feature about our NIZK scheme is that it is the first to have both a deterministic proving algorithm and be perfectly zero knowledge. We observe that these features combined give a sort of deniability property for NIZKs. A prover can originally create a NIZK proof of statement x under (legitimate) witness w_0 to generate proof π.. Later, he can claim that π was a proof given using witness $w_1 \neq w_0$ even if he did not know of the witness w_1 when he first made the proof.

Let F be a puncturable PRF that takes inputs of ℓ bits and outputs λ bits. Let $f(\cdot)$ be a one way function. Let L be a language and $R(\cdot,\cdot)$ be a relation that takes in an instance and a witness. Our system will allow proofs of instances of ℓ bits (for some integer ℓ) and witness of ℓ' bits. The values of bounds given by the values $\ell and \ell'$ can be specified at setup, although we suppress that notation here.

- $\mathsf{NIZKSetup}(1^\lambda)$: The setup algorithm first chooses a puncturable PRF key K for F. Next, it creates an obfuscation of the Verify NIZK of Figure 29. The size of the program is padded to be the maximum of itself and the program Verify NIZK* of Figure 30. In addition, it creates an obfuscation of the program Prove NIZK of Figure 31. The size of the program is padded to be the maximum of itself and the program Prove NIZK* of Figure 32.

  The common reference string CRS consists of the two obfuscated programs.

- $\mathsf{NIZKProve}(\mathrm{CRS}, x, w)$: The NIZK prove algorithm runs the obfuscated program of Prove NIZK from CRS on inputs (x, w). If $R(x, w)$ holds the program returns a proof π.

- $\mathsf{NIZKVerify}(\mathrm{CRS}, x, \pi)$ The verification algorithm runs the obfuscated program of Verify NIZK on inputs x, π and outputs the accept/reject result.

Verify NIZK

Constants: PRF key K.
Input: Instance x, proof π.

1. Test if $f(\pi) = f(F(K, x))$. Output accept if true, reject if false.

Figure 29: Program Verify NIZK

Verify NIZK*

Constants: PRF key $K(\{x^*\}), x^*, z^*$.
Input: Instance x, proof π.

1. If $x = x^*$, test if $f(\pi) = z$. Output accept if true, reject otherwise.
2. Test if $f(\pi) = f(F(K, x))$. Output accept if true, reject if false.

Figure 30: Program Verify NIZK*

Prove NIZK

Constants: PRF key K.
Input: Instance x, witness w.

1. Test if $R(x, w) = 1$. If so output $F(K, x)$. Otherwise, output reject.

Figure 31: Program Prove NIZK

Theorem 8. The scheme is perfectly zero knowledge.

Prove NIZK*

Constants: PRF key $K(\{x^*\})$.
Input: Instance x, witness w.

1. Test if $R(x, w) = 1$. If so output $F(K, x)$. Otherwise, output reject.

Figure 32: Program Prove NIZK*

Proof. Consider a simulator $\mathcal{S}$ that on input x, runs the setup algorithm and outputs the corresponding CRS along with $\pi^* = F(K, x)$. The simulator has the exact same distribution as any the real prover's algorithm for any $x \in L$ and witness w where $R(x, w) = 1$.

■

Theorem 9. If our obfuscation scheme is indistingishuably secure, F is a secure punctured PRF, and $f(\cdot)$ is a one way function, then our NIZK scheme is sound.

Proof. We show standard soundness by showing that for all $x^* \notin L$, no poly-time attacker can create a proof π^* for x^* that accepts with non-negligible probability. We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original soundness security game. We prove that a poly-time attacker's advantage must be negligibly close between each successive one. Then, we show that any poly-time attacker in the final experiment that succeeds with non-negligible probability can be used to break one way functions.

- Hyb_0 : In the first hybrid the following game is played.
  1. K is chosen as a key for the puncturable PRF.
  2. The public key CRS is given out as an obfuscation of the programs Verify NIZK and Prove NIZK.
  3. The attacker sends π^* and wins if $\mathsf{NIZKVerify}(x^*, \pi^*)$ outputs accept.
- Hyb_1 : Is the same as Hyb_0 except we use the program Prove NIZK* of Figure 32 (in place of Prove NIZK).
- Hyb_2 : Is the same as Hyb_0 except we let $z^* = f(F(K, m^*))$ and use the program Verify NIZK* of Figure 30 (in place of Verify NIZK).
- Hyb_3 : Is the same as Hyb_1 except z^* is set to be $f(t)$ where $t \in \{0,1\}^\lambda$ is chosen uniformly at random.

First, we argue that the advantage for any poly-time attacker in forging a NIZK must be negligibly close in hybrids Hyb_0 and Hyb_1. We first observe that w that the input/output behavior of programs Prove NIZK and Prove NIZK* are identical. The only difference is that in Prove NIZK* the PRF at point x^* is punctured out of the constrained PRF key. However, since x^* is not in L, $F(K, x^*)$ will never get called. Therefore, if there is a difference in advantage we can create an algorithm $\mathcal{B}$ that breaks indistinguishability security for obfuscation. $\mathcal{B}$ runs as the challenger. When it is to create the obfuscated program it submits both programs Verify Signature and Verify Signature*. It embeds the resulting obfuscated program into the CRS. If the $i\mathcal{O}$ challenger chooses the first then we are in Hyb_0. If it chooses the second then we are in Hyb_1. $\mathcal{B}$ will output 1 if the attacker successfully forges. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on $i\mathcal{O}$ security.

Second, we argue that the advantage for any poly-time attacker in forging a signature must be negligibly close in hybrids Hyb_1 and Hyb_2. We first observe that that the input/output behavior of programs Verify NIZK and Verify NIZK* are identical. The only difference is that the first program computes $F(K, x^*)$ before applying the OWF f to it when verifying an instance input x^*, whereas the second is given $f(F(K, x^*))$ as the constant z^*. Therefore, if there is a difference in advantage we can create an algorithm $\mathcal{B}$ that breaks indistinguishability security for obfuscation. $\mathcal{B}$ runs as the challenger, but where t^* is chosen at

random. When it is to create the obfuscated program it submits both programs Verify Signature and Verify Signature*. It embeds the resulting obfuscated program into the CRS. If the $i\mathcal{O}$ challenger chooses the first, then we are in Hyb_1. If it chooses the second, then we are in Hyb_2. $\mathcal{B}$ will output 1 if the attacker successfully forges. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on $i\mathcal{O}$ security. *We note that for Verify NIZK* to be well defined, the value x^* must be well defined. Since we are quantifying over all x^*, x^* is known to the reduction.*

We now argue that the advantage for any poly-time attacker in forging a NIZK must be negligibly close in hybrids Hyb_2 and Hyb_3. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the *selective* security of the constrained pseudorandom function at the punctured points. $\mathcal{B}$ first gives x^* to the constrained PRF challenger and gets the punctured PRF key $K(\{x^*\})$ and challenge a. It runs as in Hyb_2 except it sets $z^* = f(a)$ If a is the output of the PRF at point x^*, then we are in Hyb_2. If it was chosen uniformly at random, we are in Hyb_3. $\mathcal{B}$ will output 1 if the attacker is successful. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on the constrained PRF security. Here we were able to reduce to selective security since in standard NIZK soundness the instance x^* is given at the beginning to the reduction.

Finally, if there is an attacker in Hyb_2, we can use it to break the security of the OWF. We build a reduction $\mathcal{B}$ that receives y as the challenge for a OWF and sets $z^* = y$. If an attacker successfully forges a NIZK on x^*, then by definition he has computed a π such that $f(\pi) = z^*$. Therefore, if the OWF is secure, no poly-time attacker can forge with non-negligible advantage. Since the advantage of all poly-time attackers are negligibly close in each successive hybrid, this proves soundness for the NIZK scheme.

■

5.6 Injective Trapdoor Functions

The final primitive that we will build is injective trapdoor functions (TDFs). Once again we will use the ingredients of indistinguishable obfuscation and punctured PRFs. In addition, we will use a public key encryption scheme and an *injective* one way function. Since we showed how to build PKE earlier, the only new assumption we add relative to the other primitives is that our one way function is injective. The actual construction will use the public key encryption system to encrypt the input r. The randomness used for encryption is derived from a punctured PRF applied to r.

Let $f(\cdot)$ be an *injective* one way function, where on inputs of λ bits outputs $\ell = \ell(\lambda)$ bits Let F be a puncturable PRF that takes inputs of ℓ bits and outputs λ bits. We denote $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ as the respective PKE encryption and decryption algorithms. The encryption algorithm will take messages of length $\lambda + 1$ bits and randomness of a λ bit string. We describe our injective trapdoor function that takes inputs of λ bits.

- $\mathsf{TDFSetup}(1^\lambda)$: The setup algorithm first chooses a puncturable PRF key K for F as well as generates a public and secret key pair for a public key encryption scheme where the encryption system's public key is $\tilde{\mathrm{PK}}$ and secret key is $\tilde{\mathrm{SK}}$. Next, it creates an obfuscation of the program TDF of Figure 33. The size of the program is padded to be the maximum of itself and the program TDF* of Figure 34. The TDF public key PK is the obfuscated program. The secret key SK is the secret key of the encryption scheme.

- $\mathsf{TDFEval}(\mathrm{PK}, r)$: The TDF evaluation algorithm takes in the public key and a value $r \in \{0,1\}^\lambda$ and runs the obfuscated program of PK on input r.

- $\mathsf{TDFInvert}(\mathrm{SK}, y)$ The inversion algorithm runs the public key decryption algorithm $\mathsf{Decrypt}$ on y with its secret key and outputs the message (if any) that it recovers.

Theorem 10. If our obfuscation scheme is indistingishuably secure, F is a secure punctured PRF, and $f(\cdot)$ is an injective one way function, then our TDF is hard to invert.

TDF

Constants: Punctured PRF key K.
Input: Value $r \in \{0,1\}^\lambda$.

1. Compute $t = f(r)$.
2. Output $y = \mathsf{Encrypt}(\mathrm{PK}', 1|r; F_K(t))$

Figure 33: TDF

TDF*

Constants: Punctured PRF key $K(\{t^*\})$ and values t^*, y^*.
Input: Value $r \in \{0,1\}^\lambda$.

1. Compute $t = f(r)$.
2. If $t = t^*$, then output $y = y^*$.
3. Else, output $y = \mathsf{Encrypt}(\mathrm{PK}', 1|r; F_K(t))$

Figure 34: TDF*

Proof. We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original signature security game. We prove that a poly-time attacker's advantage must be negligibly close between each successive one. Then, we show that with high probability no inverse exist in the final hybrid experiment, so no attacker can win there.

- Hyb_0 : In the first hybrid the following game is played.
  1. The value r^* is chosen randomly.
  2. The value t^* is computed as $\mathrm{PRG}(r^*)$.
  3. K is chosen as a key for the puncturable PRF.
  4. The value y^* is computed as $\mathsf{Encrypt}(\mathrm{PK}', 1|r^*; F_K(t^*))$.
  5. The public key PK is given out as an obfuscation of the program TDF. PK and y^* are given to the attacker.
  6. The attacker outputs r' and wins is $\mathsf{TDFEval}(\mathrm{PK}, r') = y^*$.
- Hyb_1 : Is the same as Hyb_0 except we use the program TDF* of Figure 34.
- Hyb_2 : Is the same as Hyb_1 except y^* is computed as $y^* = \mathit{Encrypt}(\mathrm{PK}, 1|r^*; z)$ for randomly chosen z.
- Hyb_3 Is the same as Hyb_2 except y^* is computed as an encryption of the all 0's string $0^{\lambda+1}$ as $y^* = \mathsf{Encrypt}(\mathrm{PK}, 0^\lambda; z)$ for randomly chosen z.

First, we argue that the advantage for any poly-time attacker in inverting the TDF must be negligibly close in hybrids Hyb_0 and Hyb_1. We first observe that that the input/output behavior of programs TDF and Prove TDF* are identical. The only difference is that in Prove TDF* the output on $t = t^*$ is hardwired in to be the constant y^*. Whereas the program TDF computes $\mathsf{Encrypt}(\mathrm{PK}', r^*; F_K(t^*))$ on input t^*. *Since the OWF is injective, this will not effect the program behavior on any inputs other than r^*.* Therefore, if there is a difference in advantage we can create an algorithm $\mathcal{B}$ that breaks indistinguishability security for obfuscation. $\mathcal{B}$ runs as the challenger. When it reaches the point of creating the obfuscated program it submits both programs TDF and TDF*. It receives the obfuscated program and sets this to be PK. If the $i\mathcal{O}$ challenger chooses the first, then we are in Hyb_0. If it chooses the second, then we are in Hyb_1. $\mathcal{B}$ will output 1 if the attacker successfully forges. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$

as an attacker on $i\mathcal{O}$ security. *Note, that the program TDF* depends on t^*, y^* being defined. The reduction algorithm can choose them before the step which it receives the obfuscations from the $i\mathcal{O}$ challenger.*

Next, we argue that the advantage for any poly-time attacker in inverting must be negligibly close in hybrids Hyb_1 and Hyb_2. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks the *selective* security of the constrained pseudorandom function at the punctured points. $\mathcal{B}$ first gives t^* to the constrained PRF challenger and gets the punctured PRF key $K(\{t^*\})$ and challenge a. It runs as in Hyb_2 except it creates y^* as $\mathsf{Encrypt}(\mathrm{PK}', 1|r^*; a)$ If a is the output of the PRF at point t^*, then we are in Hyb_1. If a was chosen uniformly at random, then we are in Hyb_2. $\mathcal{B}$ will output 1 if the attacker is successful. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ as an attacker on the constrained PRF security. Here we were able to reduce to selective security since r^* is chosen randomly by the game and outside of the attacker's control.

We argue that the advantage for any poly-time attacker in inverting must be negligibly close in hybrids Hyb_2 and Hyb_3. Otherwise, we can create a reduction algorithm $\mathcal{B}$ that breaks IND-CPA security of the public key encryption scheme. $\mathcal{B}$ runs the reduction as in Hyb_2. When it comes to creating y^*, it submits $m_0 = r^*$ and $m_1 = 0^{\lambda+1}$ to the IND-CPA challenger. It gets back a ciphertext a and sets $y^* = a$. If a is the encryption of r^*, then we are in Hyb_2. If it is the encryption of $0^{\lambda+1}$, then we are in Hyb_2. $\mathcal{B}$ will output 1 if the attacker is successful. Any attacker with different advantages in the hybrids leads to $\mathcal{B}$ on IND-CPA security.

Finally, if there is an attacker in Hyb_3, we can use it to break the security of the injective OWF. We build a reduction $\mathcal{B}$ that receives a as the challenge for a OWF and sets $y^* = a$. We know by definition of the OWR game that $a = f(r')$ for some r'. This will implicitly set $r^* = r'$ where r^* is unknown to the reduction algorithm. Suppose, an attacker successfully inverts y^* with respect to the obfuscated program of TDF*. Then it must be the case that the attacker gave r^*. All other inputs result in an output of an encryption where a 1 is appended to the beginning of the encrypted message, whereas y^* begins with a 0. So by correctness of the PKE, the attacker must have given r^*. However, $f(r^*) = y^* = a$, so the attacker inverted the one way function.

■

5.7 Oblivious Transfer

We briefly sketch a simple 2-round construction for building (semi-honest) Oblivious Transfer (OT) from indistinguishability obfuscation and one-way functions. Semi-honest OT implies malicious secure OT in a black-box manner [Hai08]. This construction does not use the punctured programs technique. Note that OT from indistinguishability obfuscation and one-way functions already follows from claims implicit in the literature: Indistinguishability Obfuscation implies Witness Encryption [GGH$^+$13a], Witness Encryption and one-way functions implies OT [Rud89] (see [Bei11] for a sketch). We sketch the implication here for completeness, based on an idea present in [GIS$^+$10]:

Suppose the receiver has a bit b, the sender has two strings s_0, s_1. In an OT protocol, the sender should learn nothing, and the receiver should learn only s_b. The protocol will proceed as follows:

1. The receiver sends a standard statistically binding commitment $c = \mathrm{com}(b; r)$ to the sender.
2. The sender prepares a program $P_{(c,s_0,s_1)}$, that takes as input (b', r'), and behaves as follows: If $c = \mathrm{com}(b'; r')$, then it outputs $s_{b'}$, otherwise it outputs $\perp$.

   The sender computes $\tilde{P} = i\mathcal{O}(P_{(c,s_0,s_1)})$, and sends it to the receiver.
3. The receiver feeds (b, r) to the program $\tilde{P}$ to recover s_b.

Semi-honest security of this protocol is straightforward: If the sender is corrupted, then security for the receiver follows immediately from the security of the commitment scheme. If the receiver is corrupted, the simulator obtains $s^* = s_b$ from the trusted party, and it prepares a program P' that always outputs s^* if the check passes. It outputs $i\mathcal{O}(P')$ instead. However, note that by statistical binding, once the (semi-honest)

corrupted receiver sends its commitment c, the actual $P_{(c,s_0,s_1)}$ can only output s_b or $\perp$; no other output is possible. Therefore P' and $P_{(c,s_0,s_1)}$ are functionally identical, and security follows from the $i\mathcal{O}$ property.

References

[Bei11] Amos Beimel. Secret-sharing schemes: A survey. In *IWCC*, pages 11–46, 2011.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, pages 868–886, 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO*, 2011.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. *IACR Cryptology ePrint Archive*, 2013:352, 2013.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In *CRYPTO*, pages 90–104, 1997.

[CS04] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, January 2004.

[DF11a] Markus Dürmuth and David Mandell Freeman. Deniable encryption with negligible detection probability: An interactive construction. In *EUROCRYPT*, pages 610–626, 2011.

[DF11b] Markus Dürmuth and David Mandell Freeman. Deniable encryption with negligible detection probability: An interactive construction. *IACR Cryptology ePrint Archive*, 2011:66, 2011.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography, 1976.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH+13a] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. Cryptology ePrint Archive, Report 2013/128, 2013. `http://eprint.iacr.org/`.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Jour. of Computer and System Science*, 28(2):270–299, 1984.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[Gol01] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

[Hai08] Iftach Haitner. Semi-honest to malicious oblivious transfer - the black-box way. In *TCC*, pages 412–426, 2008.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. *IACR Cryptology ePrint Archive*, 2013:379, 2013.

[MS09] Steven Myers and Abhi Shelat. Bit encryption is complete. In *FOCS*, pages 607–616, 2009.

[Rud89] Steven Rudich. Unpublished, 1989.