

# Constrained Keys for Invertible Pseudorandom Functions

Dan Boneh, Sam Kim, and David J. Wu

Stanford University  
{dabo,skim13,dwu4}@cs.stanford.edu

## Abstract

A constrained pseudorandom function (PRF) is a secure PRF for which one can generate constrained keys that can only be used to evaluate the PRF on a subset of the domain. Constrained PRFs are used widely, most notably in applications of indistinguishability obfuscation (*iO*). In this paper we show how to constrain an *invertible* PRF (IPF), which is significantly harder. An IPF is a secure *injective* PRF accompanied by an inversion algorithm. A constrained key for an IPF can only be used to evaluate the IPF on a subset  $S$  of the domain, and to invert the IPF on the image of  $S$ . We first define the notion of a constrained IPF and then give two main constructions: one for puncturing an IPF and the other for (single-key) circuit constraints. Both constructions rely on recent work on *private* constrained PRFs. We also show that constrained pseudorandom permutations are impossible under our definition.

## 1 Introduction

Pseudorandom functions (PRFs) [GGM84] and pseudorandom permutations (PRPs) [LR88] have found numerous applications in cryptography, such as encryption, data integrity, user authentication, key derivation, and others. Invertible PRFs are a natural extension that borrows features from both concepts. An invertible PRF (IPF) is an efficiently-computable *injective* function  $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  equipped with an efficient inversion algorithm  $F^{-1} : \mathcal{K} \times \mathcal{Y} \rightarrow \mathcal{X} \cup \{\perp\}$ . The inversion algorithm is required to satisfy the following two properties for all  $k \in \mathcal{K}$ :

- (1)  $F^{-1}(k, F(k, x)) = x$  for all  $x \in \mathcal{X}$ .
- (2)  $F^{-1}(k, y) = \perp$  whenever  $y$  is not in the image of  $f(x) := F(k, x)$ .

We say that an IPF  $F$  is secure if no poly-bounded adversary can distinguish the following two experiments. In one experiment the adversary is given oracles for the function  $f(x) := F(k, x)$  and its inverse  $f^{-1}(x) := F^{-1}(k, x)$ , where  $k$  is randomly chosen in  $\mathcal{K}$ . In the other experiment the adversary is given oracles for a *random* injective function  $g : \mathcal{X} \rightarrow \mathcal{Y}$  and its inverse  $g^{-1} : \mathcal{Y} \rightarrow \mathcal{X} \cup \{\perp\}$ . These two experiments should be indistinguishable. We define this in detail in Section 3. Note that when  $\mathcal{X} = \mathcal{Y}$ , an IPF is the same as a strong pseudorandom permutation [LR88].

IPFs come up naturally in the context of deterministic authenticated encryption (DAE) [RS06], as discussed below. A closely related concept called a *pseudorandom injection* (PRI) [RS06] is similar to an IPF except for some syntactic differences (an IPF is a pseudorandom injection without additional length constraints and with an empty header).

**Constrained PRFs.** In this paper we define and construct constrained IPFs. It is helpful to first review constrained PRFs [BW13, KPTZ13, BGI14]. Recall that a PRF  $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  is said to be a constrained PRF if one can derive constrained keys from the master PRF key  $k$ . A constrained key  $k_g$  is associated with a predicate  $g : \mathcal{X} \rightarrow \{0, 1\}$ , and this  $k_g$  enables one to evaluate  $F(k, x)$  for all  $x \in \mathcal{X}$  where  $g(x) = 1$ , but at no other points of  $\mathcal{X}$ . A constrained PRF is secure if given constrained keys for predicates  $g_1, \dots, g_Q$  of the adversary’s choosing, the adversary cannot distinguish the PRF from a random function at points not covered by the given keys, namely at points  $x$  where  $g_1(x) = \dots = g_Q(x) = 0$ . We review the precise definition in Section 3.1.

Constrained PRFs have found numerous applications in cryptography [BW13, KPTZ13, BGI14]: they imply identity-based key exchange and broadcast encryption, and are a crucial ingredient in many applications of indistinguishability obfuscation (*iO*) [SW14].

The simplest constraint is a *puncturing* constraint, a constraint that enables one to evaluate the function on its entire domain except for one point. For  $x \in \mathcal{X}$  we denote by  $k_x$  a *punctured key* that lets one evaluate the PRF at all points in  $\mathcal{X}$ , except for the punctured point  $x$ . Given the key  $k_x$ , the adversary should be unable to distinguish  $F(k, x)$  from a random element in  $\mathcal{Y}$ . PRFs supporting puncturing constraints can be easily constructed from the tree-based PRF of [GGM84], as discussed in [BW13, KPTZ13, BGI14].

**Constrained IPFs.** Given the wide applicability of constrained PRFs, it is natural to look at constraining other symmetric primitives such as PRPs and, more generally, IPFs. A constrained key  $k_g$  for an IPF enables one to evaluate the IPF at all points  $x \in \mathcal{X}$  for which  $g(x) = 1$ , and invert at all points  $y = F(k, x') \in \mathcal{Y}$  for which  $g(x') = 1$ . Security for a constrained IPF is defined as for a PRF: the adversary is given a number of constrained keys and tries to distinguish the IPF from a random injective function at points not covered by any of the given keys. See Section 3.1 for more details.

We first show in Section 3.3 that constrained PRPs do not exist in our model. However constrained IPFs, where the range can be larger than the domain, can exist. The challenge is to construct them. Surprisingly, constraining an IPF is significantly harder than constraining a PRF, even for simple puncturing constraints. For example, it is not difficult to see that puncturing a Luby-Rackoff cipher by puncturing the underlying PRFs does not work.

In this paper, we present constrained IPFs for both puncturing constraints and for arbitrary circuit constraints. Both constructions make use of a recent primitive called a *private constrained* PRF [BLW17] that can be constructed from the learning with errors (LWE) problem [BKM17, CC17, BTVW17]. Roughly speaking, a private constrained PRF is a constrained PRF where a constrained key  $k_g$  reveals nothing about the constraint  $g$ . Before we describe our constructions, let us first look at an application.

**IPFs and deterministic encryption.** While constrained IPFs are interesting in their own right, they come up naturally in the context of deterministic encryption. IPFs are related to the concept of *deterministic authenticated encryption* (DAE) introduced by Rogaway and Shrimpton [RS06] where encryption is deterministic and does not take a nonce as input. A DAE provides the same security guarantees as (randomized) authenticated encryption, as long as all the messages encrypted under a single key are distinct. Rogaway and Shrimpton show that an IPF whose range is sufficiently larger than its domain is equivalent to a secure DAE. They further require that the length of the IPF output depend only on the length of the input, and this holds for all our constructions. Hence, our

constrained IPFs give the ability to constrain keys in a DAE encryption scheme.

## 1.1 Building Constrained IPFs

In Section 4, we present two constructions for constrained IPFs on a domain  $\mathcal{X} = \{0, 1\}^n$ . Our first construction, a warm-up, only supports puncturing constraints. Our second construction gives a constrained IPF for arbitrary circuit constraints, but is only secure if a single constrained key is released. Here we give the main ideas behind the constructions. Both rely heavily on the recent development of private constrained PRFs. In Section 5, we show how to instantiate our constructions from the LWE assumption. In Section 7, we also show that using  $i\mathcal{O}$ , it is possible to construct a multi-key, circuit-constrained IPF.

**A puncturable IPF.** Let  $F_1: \mathcal{K}_1 \times \mathcal{X} \rightarrow \mathcal{V}$  and  $F_2: \mathcal{K}_2 \times \mathcal{V} \rightarrow \mathcal{X}$  be two secure PRFs. Define the following IPF  $F$  on domain  $\mathcal{X}$  using a key  $k = (k^{(1)}, k^{(2)}) \in \mathcal{K}_1 \times \mathcal{K}_2$ :

$$F((k^{(1)}, k^{(2)}), x) := \left\{ \begin{array}{l} y_1 \leftarrow F_1(k^{(1)}, x) \\ y_2 \leftarrow x \oplus F_2(k^{(2)}, y_1) \\ \text{output } (y_1, y_2) \end{array} \right\} \quad F^{-1}((k^{(1)}, k^{(2)}), (y_1, y_2)) := \left\{ \begin{array}{l} x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2 \\ \text{if } F_1(k^{(1)}, x) \neq y_1 \text{ then } x \leftarrow \perp \\ \text{output } x \end{array} \right\} \quad (1.1)$$

It is not difficult to show that  $F$  is a secure IPF. In fact, one can view this IPF as an instance of a DAE construction called SIV (Synthetic-IV) [RS06].

The question is how to securely puncture  $F$ . As a first attempt, suppose  $F_1$  is a puncturable PRF, say constructed from the tree-based GGM construction [GGM84]. To puncture the IPF  $F$  at a point  $x \in \mathcal{X}$ , one can puncture  $F_1$  at  $x$  to obtain the IPF punctured key  $k_x := (k_x^{(1)}, k^{(2)})$ . This key  $k_x$  prevents the evaluation  $F$  at the point  $x$ , as required. However, this is completely insecure. To see why, observe that given  $k_x$ , the adversary can easily distinguish  $F(k, x)$  from a random pair in  $\mathcal{V} \times \mathcal{X}$ : given a challenge value  $(y_1, y_2)$  for  $F(k, x)$ , the adversary can simply test if  $x = F_2(k^{(2)}, y_1) \oplus y_2$ . This will be satisfied by  $F(k, x)$ , but is unlikely to be satisfied by a random pair in  $\mathcal{V} \times \mathcal{X}$ .

To properly puncture  $F$  at  $x$  we must puncture  $F_1$  at  $x$  and puncture  $F_2$  at  $y_1 := F_1(k^{(1)}, x)$ . The punctured key for  $F$  is then  $k_x := (k_x^{(1)}, k_{y_1}^{(2)})$ . Here, it is vital that the punctured key  $k_{y_1}^{(2)}$  reveal nothing about the punctured point  $y_1$ . Otherwise, it is again easy to distinguish  $F(k, x) = (y_1, y_2)$  from a random pair in  $\mathcal{V} \times \mathcal{X}$  using the exposed information about  $y_1$ . To ensure that  $y_1$  is hidden, we must use a *privately puncturable* PRF for  $F_2$ . Currently the best constructions for a privately puncturable PRF rely on the LWE assumption [BKM17, CC17, BTVW17]. It is not known how to construct a privately puncturable PRF from one-way functions. We show in Theorem 4.3 that with this setup, the puncturable IPF in Eq. (1.1) is secure.

**A constrained IPF for circuit constraints.** Next we generalize Eq. (1.1) to support an arbitrary circuit constraint  $g$ . As a first step we can constrain  $k^{(1)}$  to  $g$  so that the IPF constrained key is  $k_g := (k_g^{(1)}, k^{(2)})$ . We can use for  $F_1$  any of the candidate circuit-constrained PRFs [BW13, BV15].

As before, this is insecure: for security we must also constrain  $F_2$ . However we immediately run into a problem. Following the blueprint in Eq. (1.1) we must puncture  $F_2$  at all points  $F_1(k^{(1)}, x)$  where  $g(x) = 0$ . However, because the size of this set can be super-polynomial, we would need to constrain  $F_2$  to a set containing super-polynomially-many pseudorandom points. The difficulty is

that  $F_2$  cannot efficiently test if an input  $v \in \mathcal{V}$  satisfies  $v = F_1(k^{(1)}, x)$  with  $g(x) = 0$ . Because  $F_1$  is not invertible, this cannot be done even given  $k^{(1)}$ .

We solve this problem by replacing  $F_1(k^{(1)}, x)$  with a CCA-secure public-key encryption  $\text{PKE.Encrypt}(\text{pk}, x; r_x)$ , where the randomness  $r_x = F_1(k^{(1)}, x)$  is derived from  $F_1$  and  $\text{pk}$  is the public key. In this case, the input to  $F_2$  is a ciphertext  $\text{ct}$  that encrypts the point  $x$ . The output of the IPF is the pair  $(\text{ct}, F_2(k^{(2)}, \text{ct}) \oplus x)$ . When constraining  $F_2$ , we embed the secret decryption key  $\text{sk}$  for the public-key encryption scheme in the constrained key. Then, on an input ciphertext  $\text{ct}$ , the constraint function first decrypts  $\text{ct}$  (using  $\text{sk}$ ) to obtain a value  $x \in \mathcal{X}$ , and then checks if  $g(x) = 1$ . Because knowledge of  $\text{sk}$  allows one to invert on *all* points, it is *critical* that the constrained key hides  $\text{sk}$ . Here, we rely on a strong simulation-based notion of constraint privacy [BKM17, CC17]. In Theorem 4.7, we show that as long as the underlying PKE scheme is CCA-secure and  $F_2$  is a (single-key) *private* constrained PRF, then the resulting scheme is a (single-key) secure circuit-constrained IPF.

By design, our circuit-constrained IPF provides two ways to invert: the “honest” method where on input  $(\text{ct}, y_2)$ , the evaluator uses the PRF key  $k^{(2)}$  to compute a (candidate) preimage  $x \leftarrow F_2(k^{(2)}, \text{ct}) \oplus y_2$ , and the “trapdoor” method where an evaluator who holds the decryption key for the public-key encryption scheme simply decrypts  $\text{ct}$  to recover the (candidate) preimage  $x$ . The inversion trapdoor plays an important role in the security analysis of our circuit-constrained IPF because it enables the reduction algorithm to properly simulate the inversion oracle queries in the IPF security game. We refer to Appendix B for the full details.

Theorems 4.3 and 4.7 state that our puncturable IPF and circuit-constrained IPF are secure assuming the security (and privacy) of the underlying constrained PRFs (and in the latter case, CCA-security of the public-key encryption scheme). While it may seem that security of the IPF should directly follow from security of the underlying puncturable (or constrained) PRFs, several complications arise in the security analysis because we give the adversary access to an IPF inversion oracle in the security game. As a result, our security analysis requires a more intricate hybrid argument where we appeal to the security of the underlying constrained PRFs multiple times. We provide the complete proofs in Appendices A and B.

**A multi-key constrained IPFs from  $i\mathcal{O}$ .** In Section 7, we also show that an indistinguishability obfuscation of the puncturable IPF from Eq. (1.1) gives a multi-key circuit-constrained IPF. This construction parallels the Boneh-Zhandry construction of multi-key circuit-constrained PRFs from standard puncturable PRFs and indistinguishability obfuscation [BZ14].

**Supporting key-delegation.** Several constrained PRF constructions support a mechanism called key-delegation [BW13, CRV14, DDM17], where the holder of a constrained PRF key can further constrain the key. For instance, the holder of a constrained key  $k_f$  for a function  $f$  can further constrain the key to a function of the form  $f \wedge g$  where  $(f \wedge g)(x) = 1$  if and only if  $f(x) = g(x) = 1$ . In Section 6, we describe how our circuit-constrained IPF can be extended to support key-delegation.

**Open problems.** Our circuit-constrained IPF from LWE is secure only if a *single* constrained key is issued. In Section 6, we show how to modify our construction to support giving out a pre-determined number of keys, provided that each successive key adds a further constraint on the previous key (i.e., via key delegation). Is there an IPF that supports multiple constrained keys for an arbitrary set of circuit constraints (and does not rely on strong assumptions such as  $i\mathcal{O}$  or

multilinear maps)? A positive answer would also give a circuit-constrained PRF that supports multiple keys, which is currently an open problem.

Our circuit-constrained IPF relies on the LWE assumption. Can we build constrained IPFs from one-way functions? For example, the tree-based PRF of [GGM84] gives a prefix-constrained PRF from one-way functions. Can we build a prefix-constrained IPF from one-way functions?

## 1.2 Related Work

Authenticated encryption was first formalized over a sequence of works [BN00, BR00, KY00, Rog02, RBB03]. Deterministic authenticated encryption, and the notion of a pseudorandom injection, were introduced in [RS06]. These notions have been further studied in [IY09a, IY09b]. Our circuit-constrained IPF relies on derandomizing a public-key encryption scheme. Similar techniques have been used in the context of constructing deterministic public-key encryption [BBO07, BFOR08, BFO08, FOR12]. Note however that an IPF is a *secret-key* primitive, so in our setting, the randomness used for encryption can be derived using a PRF on the message rather than as a publicly-computable function on the input. This critical difference eliminates the need to make entropic assumptions on the inputs.

Since the introduction of constrained PRFs in [BW13, BGI14, KPTZ13], numerous works have studied constraining other cryptographic primitives such as verifiable random functions (VRFs) [CRV14, Fuc14, DDM17] and signatures [BGI14, BF14]. Other works have focused on constructing adaptively-secure constrained PRFs [Hof14, FKPR14, HKW15] and constrained PRFs on unconstrained inputs [DKW16, DDM17].

## 2 Preliminaries

For a positive integer  $n$ , we write  $[n]$  to denote the set  $\{1, 2, \dots, n\}$ . For a distribution  $\mathcal{D}$ , we write  $x \leftarrow \mathcal{D}$  to denote that  $x$  is sampled from  $\mathcal{D}$ ; for a finite set  $S$ , we write  $x \stackrel{\mathcal{R}}{\leftarrow} S$  to denote that  $x$  is sampled uniformly from  $S$ . Throughout this work, we write  $\lambda$  for the security parameter. We say a function  $f(\lambda)$  is negligible in  $\lambda$  if  $f(\lambda) = o(1/\lambda^c)$  for all  $c \in \mathbb{N}$ . We denote this by writing  $f(\lambda) = \text{negl}(\lambda)$ . We say that an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We write  $\text{poly}(\lambda)$  to denote a quantity that is bounded by some polynomial in  $\lambda$ . We say that an event occurs with overwhelming probability if its complement occurs with negligible probability, and that it occurs with noticeable probability if its complement occurs with non-negligible probability. We say that two families of distributions  $\mathcal{D}_1$  and  $\mathcal{D}_2$  are computationally indistinguishable if no efficient algorithm can distinguish between  $\mathcal{D}_1$  and  $\mathcal{D}_2$ , except with negligible probability. We say that  $\mathcal{D}_1$  and  $\mathcal{D}_2$  are statistically indistinguishable if the statistical distance between  $\mathcal{D}_1$  and  $\mathcal{D}_2$  is negligible.

**Function families.** For two sets  $\mathcal{X}, \mathcal{Y}$ , we write  $\text{Funcs}[\mathcal{X}, \mathcal{Y}]$  to denote the set of functions from  $\mathcal{X}$  to  $\mathcal{Y}$ . We write  $\text{InjFuncs}[\mathcal{X}, \mathcal{Y}]$  to denote the set of *injective* functions from  $\mathcal{X}$  to  $\mathcal{Y}$ . For an injective function  $f \in \text{InjFuncs}[\mathcal{X}, \mathcal{Y}]$ , we denote by  $f^{-1} : \mathcal{Y} \rightarrow \mathcal{X} \cup \{\perp\}$  the function where  $f^{-1}(y) = x$  if  $y = f(x)$ , and  $\perp$  otherwise. We sometimes refer to  $f^{-1}$  as the (generalized) inverse of  $f$ . When the domain and range are the same, the set  $\text{InjFuncs}[\mathcal{X}, \mathcal{X}]$  is precisely the set of permutations on  $\mathcal{X}$ .

## 2.1 CCA-Secure Public-Key Encryption

A PKE scheme consists of three algorithms  $\text{PKE} = (\text{PKE.Setup}, \text{PKE.Encrypt}, \text{PKE.Decrypt})$  over a message space  $\mathcal{M}$  and a ciphertext space  $\mathcal{T}$  with the following properties:

- $\text{PKE.Setup}(1^\lambda) \rightarrow (\text{pk}, \text{sk})$ : On input the security parameter  $\lambda$ , the setup algorithm generates a public key  $\text{pk}$  and a secret key  $\text{sk}$ .
- $\text{PKE.Encrypt}(\text{pk}, m) \rightarrow \text{ct}$ : On input a public key  $\text{pk}$  and a message  $m \in \mathcal{M}$ , the encryption algorithm returns a ciphertext  $\text{ct} \in \mathcal{T}$ .
- $\text{PKE.Decrypt}(\text{sk}, \text{ct}) \rightarrow m$ : On input a secret key  $\text{sk}$  and a ciphertext  $\text{ct} \in \mathcal{T}$ , the decryption algorithm outputs a message  $m \in \mathcal{M} \cup \{\perp\}$ .

We say that a PKE scheme is (perfectly) correct if for all keys  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Setup}(1^\lambda)$ , and for all messages  $m \in \mathcal{M}$ , we have that

$$\Pr[\text{PKE.Decrypt}(\text{sk}, \text{PKE.Encrypt}(\text{pk}, m)) = m] = 1.$$

**Definition 2.1** (CCA-Security [NY90, RS91]). Let  $\text{PKE} = (\text{PKE.Setup}, \text{PKE.Encrypt}, \text{PKE.Decrypt})$  be a PKE scheme with message space  $\mathcal{M}$  and ciphertext space  $\mathcal{T}$ , and let  $\mathcal{A}$  be an efficient adversary. For a security parameter  $\lambda$  and a bit  $b \in \{0, 1\}$ , we define the CCA-security experiment  $\text{Expt}_{\mathcal{A}, \text{PKE}}^{(\text{CCA})}(\lambda, b)$  as follows. The challenger first samples  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Setup}(1^\lambda)$ . The adversary can then issue decryption oracle queries and up to one challenge oracle query.<sup>1</sup> Depending on the bit  $b \in \{0, 1\}$ , the challenger responds to each query as follows:

- **Decryption oracle.** On input a ciphertext  $\text{ct} \in \mathcal{T}$ , the challenger responds with the decryption  $m \leftarrow \text{PKE.Decrypt}(\text{sk}, \text{ct})$ .
- **Challenge oracle.** On input two messages  $m_0, m_1 \in \mathcal{M}$ , the challenger responds with the ciphertext  $\text{ct}^* \leftarrow \text{PKE.Encrypt}(\text{pk}, m_b)$ .

At the end of the experiment, the adversary  $\mathcal{A}$  outputs a bit  $b' \in \{0, 1\}$  which is the output of the experiment. An adversary  $\mathcal{A}$  is admissible if  $\mathcal{A}$  does not submit the ciphertext  $\text{ct}^*$  it received from the challenge oracle to the decryption oracle. We say that PKE is secure against chosen-ciphertext attacks (CCA-secure) if for all efficient and admissible adversaries  $\mathcal{A}$ ,

$$\left| \Pr[\text{Expt}_{\mathcal{A}, \text{PKE}}^{(\text{CCA})}(\lambda, 0) = 1] - \Pr[\text{Expt}_{\mathcal{A}, \text{PKE}}^{(\text{CCA})}(\lambda, 1) = 1] \right| = \text{negl}(\lambda).$$

The following statistical property of CCA-secure (more generally, any semantically-secure) PKE scheme will be useful in our analysis.

**Lemma 2.2.** *Let  $\text{PKE} = (\text{PKE.Setup}, \text{PKE.Encrypt}, \text{PKE.Decrypt})$  be a CCA-secure PKE scheme with message space  $\mathcal{M}$  and ciphertext space  $\mathcal{T}$ . Then, for all messages  $m \in \mathcal{M}$  and all strings  $\text{ct} \in \mathcal{T}$ , if  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Setup}(1^\lambda)$  then*

$$\Pr[\text{PKE.Encrypt}(\text{pk}, m) = \text{ct}] = \text{negl}(\lambda),$$

where the probability is taken over the randomness in  $\text{PKE.Setup}$  and  $\text{PKE.Encrypt}$ .

<sup>1</sup>In the public-key setting, security against adversaries that make a single challenge query implies security against adversaries that make multiple challenge queries (via a standard hybrid argument).

### 3 Invertible PRFs

In this section, we introduce the notion of an invertible pseudorandom function (IPF). We then extend our notions to that of a *constrained* IPF. We begin by recalling the definition of a pseudorandom function (PRF) [GGM84].

**Definition 3.1** (Pseudorandom Function [GGM84]). A pseudorandom function (PRF) with key-space  $\mathcal{K}$ , domain  $\mathcal{X}$ , and range  $\mathcal{Y}$  is a function  $F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  that can be computed by a deterministic polynomial-time algorithm. A PRF can also include a setup algorithm  $F.\text{Setup}(1^\lambda)$  that on input the security parameter  $\lambda$ , outputs a key  $k \in \mathcal{K}$ . A function  $F$  is a secure PRF if for all efficient adversaries  $\mathcal{A}$ ,

$$\left| \Pr \left[ k \leftarrow F.\text{Setup}(1^\lambda) : \mathcal{A}^{F(k, \cdot)}(1^\lambda) = 1 \right] - \Pr \left[ R \stackrel{\mathcal{R}}{\leftarrow} \text{Funs}[\mathcal{X}, \mathcal{Y}] : \mathcal{A}^{R(\cdot)}(1^\lambda) = 1 \right] \right| = \text{negl}(\lambda).$$

An invertible pseudorandom function (IPF) is an injective PRF whose inverse function can be computed efficiently (given the secret key). This requirement that the inverse be efficiently computable is the key distinguishing factor between IPFs and injective PRFs. For instance, injective PRFs can be constructed by composing a sufficiently-expanding PRF with a pairwise-independent hash function. However, it is unclear how to invert such a PRF. We now give the definition of an IPF:

**Definition 3.2** (Invertible Pseudorandom Functions). An invertible pseudorandom function (IPF) with key-space  $\mathcal{K}$ , domain  $\mathcal{X}$ , and range  $\mathcal{Y}$  consists of two functions  $F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  and  $F^{-1}: \mathcal{K} \times \mathcal{Y} \rightarrow \mathcal{X} \cup \{\perp\}$ . An IPF can also include a setup algorithm  $F.\text{Setup}(1^\lambda)$  that on input the security parameter  $\lambda$ , outputs a key  $k \in \mathcal{K}$ . The functions  $F$  and  $F^{-1}$  satisfy the following properties:

- Both  $F$  and  $F^{-1}$  can be computed by deterministic polynomial-time algorithms.
- For all security parameters  $\lambda$  and all keys  $k$  output by  $F.\text{Setup}(1^\lambda)$ , the function  $F(k, \cdot)$  is an injective function from  $\mathcal{X}$  to  $\mathcal{Y}$ . Moreover, the function  $F^{-1}(k, \cdot)$  is the (generalized) inverse of  $F(k, \cdot)$ .

**Definition 3.3** (Pseudorandomness). An IPF  $F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  is secure if for all efficient adversaries  $\mathcal{A}$ ,

$$\left| \Pr \left[ k \leftarrow F.\text{Setup}(1^\lambda) : \mathcal{A}^{F(k, \cdot), F^{-1}(k, \cdot)}(1^\lambda) \right] - \Pr \left[ R \stackrel{\mathcal{R}}{\leftarrow} \text{InjFuns}[\mathcal{X}, \mathcal{Y}] : \mathcal{A}^{R(\cdot), R^{-1}(\cdot)}(1^\lambda) \right] \right| = \text{negl}(\lambda).$$

**Remark 3.4** (Strong vs. Weak Pseudorandomness). The pseudorandomness requirement for an IPF (Definition 3.3) requires that the outputs of an IPF be indistinguishable from random against adversaries that can query the IPF in both the forward direction as well as the backward direction. We can also consider a weaker notion of pseudorandomness where the adversary is given access to an evaluation oracle  $F(k, \cdot)$ , but not an inversion oracle  $F^{-1}(k, \cdot)$ . Motivated by the applications we have in mind, in this work, we focus exclusively on building IPFs satisfying the *strong* notion of pseudorandomness from Definition 3.3, where the adversary can evaluate the IPF in both directions.

#### 3.1 Constrained PRFs and IPFs

We next review the notion of a constrained PRF [BW13, KPTZ13, BGI14] and then extend these definitions to constrained IPFs.

**Definition 3.5** (Constrained PRF [BW13, KPTZ13, BGI14]). A PRF  $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  is said to be constrained with respect to a predicate family  $\mathcal{F} = \{f : \mathcal{X} \rightarrow \{0, 1\}\}$  if there are two additional algorithms ( $F.\text{Constrain}, F.\text{Eval}$ ) with the following properties:

- $F.\text{Constrain}(k, f) \rightarrow k_f$ : On input a PRF key  $k \in \mathcal{K}$  and a function  $f \in \mathcal{F}$ , the constraining algorithm outputs a constrained key  $k_f$ .
- $F.\text{Eval}(k_f, x) \rightarrow y$ : On input a constrained key  $k_f$  and a point  $x \in \mathcal{X}$ , the evaluation algorithm outputs a value  $y \in \mathcal{Y}$ .

We say that a constrained PRF is correct for a function family  $\mathcal{F}$  if for all  $k \leftarrow F.\text{Setup}(1^\lambda)$ , every function  $f \in \mathcal{F}$ , and every input  $x \in \mathcal{X}$  where  $f(x) = 1$ , we have that

$$F.\text{Eval}(F.\text{Constrain}(k, f), x) = F(k, x).$$

**Definition 3.6** (Constrained PRF Security Experiment). Let  $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  be a constrained PRF with respect to a function family  $\mathcal{F}$ , and let  $\mathcal{A}$  be an efficient adversary. In the constrained PRF security experiment  $\text{Expt}_{\mathcal{A}, F}^{(\text{PRF})}(\lambda, b)$  (parameterized by a security parameter  $\lambda$  and a bit  $b \in \{0, 1\}$ ), the challenger begins by sampling a key  $k \leftarrow F.\text{Setup}(1^\lambda)$  and a function  $g \stackrel{\mathcal{R}}{\leftarrow} \text{Funcs}[\mathcal{X}, \mathcal{Y}]$ . The adversary is allowed to make constrain, evaluation, and challenge oracle queries. Depending on the value of the bit  $b \in \{0, 1\}$ , the challenger responds to each oracle query as follows:

- **Constrain oracle.** On input a function  $f \in \mathcal{F}$ , the challenger responds with a constrained key  $k_f \leftarrow F.\text{Constrain}(k, f)$ .
- **Evaluation oracle.** On input a point  $x \in \mathcal{X}$ , the challenger returns  $y = F(k, x)$ .
- **Challenge oracle.** On input a point  $x \in \mathcal{X}$ , the challenger returns  $y = F(k, x)$  to  $\mathcal{A}$  if  $b = 0$  and  $y = g(x)$  if  $b = 1$ .

Finally, at the end of the experiment, the adversary  $\mathcal{A}$  outputs a bit  $b' \in \{0, 1\}$  which is also the output of the experiment.

**Definition 3.7** (Constrained PRF Security). Let  $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  be a constrained PRF for a function family  $\mathcal{F}$ . We say that an adversary  $\mathcal{A}$  is admissible for the constrained PRF security experiment (Definition 3.6) if the following conditions hold:

- For all constrain queries  $f \in \mathcal{F}$  and challenge queries  $x^* \in \mathcal{X}$  the adversary makes,  $f(x^*) = 0$ .
- For all evaluation queries  $x \in \mathcal{X}$  and challenge queries  $x^* \in \mathcal{X}$  the adversary makes,  $x \neq x^*$ .

We say that  $F$  is a secure constrained PRF if for all efficient and admissible adversaries  $\mathcal{A}$ ,

$$\left| \Pr[\text{Expt}_{\mathcal{A}, F}^{(\text{PRF})}(\lambda, 0) = 1] - \Pr[\text{Expt}_{\mathcal{A}, F}^{(\text{PRF})}(\lambda, 1) = 1] \right| = \text{negl}(\lambda).$$

Without loss of generality, we restrict the adversary to make at most one challenge query in the constrained PRF security experiment.<sup>2</sup>

<sup>2</sup>As noted in [BW13], a standard hybrid argument shows that security against adversaries making a single challenge query implies security against adversaries making multiple challenge queries.



**Remark 3.8** (Selective vs. Adaptive Security). The constrained PRF security game (Definition 3.6) allows the adversary to *adaptively* choose the challenge point after making constrain and evaluation queries. We can also define a *selective* notion of security where the adversary must commit to its challenge query at the beginning of the security game (before it starts making queries). Using a standard technique called *complexity leveraging* [BB04], selective security implies adaptive security at the expense of a super-polynomial loss in the security reduction. For instance, this is the technique used in [BW13] in the context of constrained PRFs.

**Remark 3.9** (Single-Key Security). Brakerski and Vaikuntanathan [BV15] considered another relaxation of Definition 3.7 where in the constrained PRF security game (Definition 3.6), the adversary is restricted to making a single query to the constrain oracle. In the single-key setting, we can consider the notion of *selective-function* security, where the adversary must commit to its constrain oracle query at the beginning of the security experiment. Thus, in this setting, there are two different notions of selectivity: the usual notion where the adversary commits to the challenge point (Remark 3.8) and selective-function security where the adversary commits to the function. Many of the lattice-based (single-key) constrained PRF constructions [BV15, BKM17, CC17, BTVW17] are selectively secure in the choice of the constraint function, but adaptively secure in the choice of the challenge point.

**Definition 3.10** (Constrained IPF). An IPF  $(F, F^{-1})$  with key-space  $\mathcal{K}$ , domain  $\mathcal{X}$ , and range  $\mathcal{Y}$  is said to be constrained with respect to a function family  $\mathcal{F} = \{f : \mathcal{X} \rightarrow \{0, 1\}\}$  if there are three additional algorithms  $(F.Constrain, F.Eval, F.Eval^{-1})$  with the following properties:

- $F.Constrain(k, f) \rightarrow k_f$ : On input a PRF key  $k \in \mathcal{K}$  and a function  $f \in \mathcal{F}$ , the constraining algorithm outputs a constrained key  $k_f$ .
- $F.Eval(k_f, x) \rightarrow y$ : On input a constrained key  $k_f$  and a value  $x \in \mathcal{X}$ , the evaluation algorithm outputs a value  $y \in \mathcal{Y}$ .
- $F.Eval^{-1}(k_f, y) \rightarrow x$ : On input a constrained key  $k_f$  and a value  $y \in \mathcal{Y}$ , the evaluation algorithm outputs a value  $x \in \mathcal{X} \cup \{\perp\}$ .

We say that a constrained IPF is correct for a function family  $\mathcal{F}$  if for all keys  $k \leftarrow F.Setup(1^\lambda)$ , every function  $f \in \mathcal{F}$ , and  $k_f \leftarrow F.Constrain(k, f)$ , the following two properties hold:

- For all inputs  $x \in \mathcal{X}$  where  $f(x) = 1$ ,  $F.Eval(k_f, x) = F(k, x)$ .
- For all inputs  $y \in \mathcal{Y}$  where there exists  $x \in \mathcal{X}$  such that  $F(k, x) = y$  and  $f(x) = 1$ , then  $F.Eval^{-1}(k_f, y) = F^{-1}(k, y)$ .

**Definition 3.11** (Constrained IPF Security Experiment). Let  $(F, F^{-1})$  be an IPF with key-space  $\mathcal{K}$ , domain  $\mathcal{X}$ , range  $\mathcal{Y}$ , and constrained with respect to a function family  $\mathcal{F}$ . Let  $\mathcal{A}$  be an efficient adversary. The constrained IPF security experiment  $\text{Expt}_{\mathcal{A}, F}^{(IPF)}(\lambda, b)$  is defined exactly as the constrained PRF security experiment  $\text{Expt}_{\mathcal{A}, F}^{(PRF)}(\lambda, b)$  (except with the IPF in place of the PRF), where in addition to the constrain, evaluation, and challenge oracles, the adversary is also given access to an inversion oracle:

- **Inversion oracle.** On input a point  $y \in \mathcal{Y}$ , the challenger returns  $F^{-1}(k, y)$ .

At the end of the experiment, the adversary  $\mathcal{A}$  outputs a bit  $b' \in \{0, 1\}$ , which is the output of the experiment.

**Definition 3.12** (Constrained IPF Security). Let  $(F, F^{-1})$  be an IPF with key-space  $\mathcal{K}$ , domain  $\mathcal{X}$ , range  $\mathcal{Y}$ , and constrained with respect to a function family  $\mathcal{F}$ . We say that an adversary  $\mathcal{A}$  is admissible for the constrained IPF security experiment (Definition 3.11) if the following conditions hold:

- For all constrain queries  $f \in \mathcal{F}$  and challenge queries  $x^* \in \mathcal{X}$  the adversary makes,  $f(x^*) = 0$ .
- For all evaluation queries  $x \in \mathcal{X}$  and challenge queries  $x^* \in \mathcal{X}$  the adversary makes,  $x \neq x^*$ .
- For all inversion queries  $y \in \mathcal{Y}$  the adversary makes,  $y \notin \mathcal{Y}^*$ , where  $\mathcal{Y}^*$  is the set of responses to the adversary's challenge oracle queries from the challenger.

We say that  $F$  is a secure constrained IPF if for all efficient and admissible adversaries  $\mathcal{A}$ ,

$$\left| \Pr[\text{Expt}_{\mathcal{A}, F}^{(\text{IPF})}(\lambda, 0) = 1] - \Pr[\text{Expt}_{\mathcal{A}, F}^{(\text{IPF})}(\lambda, 1) = 1] \right| = \text{negl}(\lambda).$$

As in Definition 3.7, we restrict the adversary to making at most one challenge query in the constrained IPF security experiment.

**Remark 3.13** (Selective vs. Adaptive Security for IPFs). As with constrained PRFs, we can define a notion of selective security for IPFs, where the adversary commits to its challenge query at the beginning of the constrained IPF security experiment (Remark 3.8). Similarly, we can consider a single-key variant of the security game, where the adversary makes a single constrain oracle query. In this case, we can also define the corresponding notion of selective-function security (Remark 3.9).

**Puncturable PRFs and IPFs.** An important subclass of constrained PRFs is the class of punctured PRFs [BW13, KPTZ13, BGI14]. A punctured PRF over a domain  $\mathcal{X}$  is a PRF constrained with respect to the family of point functions:  $\mathcal{F} = \{f_{x^*} : \mathcal{X} \rightarrow \{0, 1\} \mid x^* \in \mathcal{X}\}$ , where  $f_{x^*}(x) = 1$  for all  $x \neq x^*$  and  $f_{x^*}(x^*) = 0$ . For notational convenience, when working with a puncturable PRF  $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ , we replace the  $F.\text{Constrain}$  algorithm with the  $F.\text{Puncture}$  algorithm that takes as input a PRF key  $k$  and a point  $x^* \in \mathcal{X}$  and outputs a punctured key  $k_{x^*}$  (a key constrained to the point function  $f_{x^*}$ ). We extend these notions accordingly to puncturable IPFs.

## 3.2 Private Constrained PRFs

One of the key primitives we will need to build constrained IPFs is a *private* constrained PRF [BLW17]. A *private* constrained PRF is a constrained PRF with the additional property that the constrained keys hide the underlying constraining function. Boneh et al. [BLW17] showed how to construct private constrained PRFs for all circuits using indistinguishability obfuscation. Recently, a number of works have shown how to construct private constrained PRFs for puncturing constraints [BKM17],  $\text{NC}^1$  constraints [CC17], and general circuit constraints [BTVW17] from standard lattice assumptions. We now review the simulation-based notion of privacy considered in [BKM17, CC17].

**Definition 3.14** (Single-Key Constraint Privacy [BKM17, CC17]). Let  $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  be a constrained PRF with respect to a function family  $\mathcal{F}$ . We say that  $F$  is a single-key, selectively-private constrained PRF for  $\mathcal{F}$  if for all efficient adversaries  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ , there exists a stateful simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  such that the following two distributions are computationally indistinguishable:

**Experiment**  $\text{Real}_{\mathcal{A},F}(\lambda)$ :

- $(f, \text{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$
- $k \leftarrow \text{F.Setup}(1^\lambda)$
- $k_f \leftarrow \text{F.Constrain}(k, f)$
- $b \leftarrow \mathcal{A}^{\text{F}(k, \cdot)}(k_f, \text{st}_{\mathcal{A}})$
- Output  $b$

**Experiment**  $\text{Ideal}_{\mathcal{A},S,F}(\lambda)$ :

- $(f, \text{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$
- $(k_f, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$
- $b \leftarrow \mathcal{A}^{\mathcal{O}_{\text{Eval}}(\cdot)}(k_f, \text{st}_{\mathcal{A}})$ , where the ideal evaluation oracle  $\mathcal{O}_{\text{Eval}}(\cdot)$  takes as input a point  $x \in \mathcal{X}$ , computes  $(y, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(x, f(x), \text{st}_{\mathcal{S}})$ , and returns  $y$
- Output  $b$

Observe that the simulator  $(S_1, S_2)$  in the ideal experiment is not given the function  $f$  as input. Nevertheless, the simulator can simulate  $k_f$  as in the real experiment. This implies that the adversary learns nothing about  $f$  from  $k_f$  beyond the value of  $f$  at points  $x \in \mathcal{X}$  where the adversary asks for  $\text{F}(k, x)$ . Leaking this minimal information about  $f$  is unavoidable.

### 3.3 Special Cases: PRPs and Constrained PRPs

Invertible pseudorandom functions can be viewed as a generalization of pseudorandom permutations (PRPs) where we allow the range of the function to be larger than its domain. A PRP is an IPF where the domain and range are identical. Our definitions for constrained IPFs can be similarly adapted to the setting of constrained PRPs. In this section, we make several observations on the (non)-existence of constrained PRPs, as well as discuss some possible relaxations of the security requirements to circumvent the impossibility results. We first show that constrained PRPs on small (polynomial-size) domains do not exist. Next, we show that puncturable PRPs do not exist, even in the case where the domain is super-polynomial. Our argument here can be extended to derive a lower bound on the size of the range of any IPF that supports puncturing constraints (or more generally, any constraint that enables evaluation on all but a polynomial number of points).

**Remark 3.15** (Small-Domain Constrained PRPs are Insecure). No constrained PRP over a polynomial-size domain can be secure under the standard pseudorandomness definition of Definition 3.12. This follows from the fact that a PRP is easily distinguishable from a PRF when the domain is small—given even a single input-output pair  $(x^*, y^*)$  for the PRP, the adversary already learns something about the values of the PRP at any point  $x \neq x^*$  (namely, the value of the PRP at  $x$  cannot be  $y^*$ ). Thus, the adversary can distinguish the real output of the PRP at  $x \neq x^*$  (which cannot be  $y^*$ ) from a uniformly random value (which can be  $y^*$  with noticeable probability when the domain is small).

**Lemma 3.16** (Puncturable PRPs are Insecure). *Let  $\text{F} : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{X}$  be a puncturable PRP with key-space  $\mathcal{K}$  and domain  $\mathcal{X}$ . Then,  $\text{F}$  cannot be secure in the sense of Definition 3.12.*

*Proof.* By Remark 3.15, it suffices to consider the case where  $1/|\mathcal{X}| = \text{negl}(\lambda)$ . In addition, we assume that for any fixed  $x^* \in \mathcal{X}$ , if  $k \leftarrow \text{F.Setup}(1^\lambda)$  and  $k_{x^*} \leftarrow \text{F.Puncture}(k, x^*)$ , then with overwhelming probability,  $\text{F.Eval}(k_{x^*}, x^*) \neq \text{F}(k, x^*)$ . Otherwise, there is a trivial adversary that breaks the puncturing security game. We now construct the following adversary for the punctured PRP security game:

1. First,  $\mathcal{A}$  makes a puncturing query for an (arbitrary) point  $x^* \in \mathcal{X}$ . It receives from the challenger a punctured key  $k_{x^*}$  and a challenge value  $y^*$ .

2. Output 1 if  $F.\text{Eval}(k_{x^*}, F.\text{Eval}^{-1}(k_{x^*}, y^*)) = y^*$  and 0 otherwise.

To complete the analysis, we compute the probability that  $\mathcal{A}$  outputs 1:

- Suppose  $y^* = F(k, x^*)$ . By correctness of  $F$ , on all inputs  $x \neq x^*$ ,  $F.\text{Eval}(k_{x^*}, x) = F(k, x)$ . Since  $F(k, \cdot)$  is a permutation, for all  $x \neq x^*$ ,  $F(k, x) \neq F(k, x^*) = y^*$ . Finally, we have that with overwhelming probability,  $F.\text{Eval}(k_{x^*}, x^*) \neq F(k, x^*) = y^*$ . Thus, with overwhelming probability, on all inputs  $x \in \mathcal{X}$ ,  $F.\text{Eval}(k_{x^*}, x) \neq y^*$ , and  $\mathcal{A}$  outputs 0.
- Suppose  $y^*$  is uniformly random over  $\mathcal{Y}$ . With overwhelming probability (since  $|\mathcal{X}|$  is super-polynomial),  $y^* \neq F(k, x^*)$ . Then, by correctness of  $F$ ,  $F.\text{Eval}(k_{x^*}, F.\text{Eval}^{-1}(k_{x^*}, y^*)) = y^*$ , and  $\mathcal{A}$  outputs 1 with overwhelming probability.

We conclude that  $\mathcal{A}$  is able to break puncturing security of  $F$  with probability close to 1.  $\square$

**Remark 3.17** (Polynomially-Puncturable PRPs are Insecure). Lemma 3.16 can be easily extended to show that that no PRP constrained with respect to a function family  $\mathcal{F} = \{f : \mathcal{X} \rightarrow \{0, 1\}\}$  is secure under Definition 3.12, if for all  $f \in \mathcal{F}$ , the number of points  $x \in \mathcal{X}$  for which  $f(x) = 0$  is polynomially-bounded.

**Remark 3.18** (Puncturable IPFs Must be Expanding). The attack given in the proof of Lemma 3.16 also applies when  $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  is a puncturable IPF with a small range—in particular, if  $|\mathcal{Y}| \leq |\mathcal{X}| \cdot \text{poly}(\lambda)$ . When this is the case, the distinguishing advantage of the attack in the proof of Lemma 3.16 is negligibly close to  $|\mathcal{X}| / |\mathcal{Y}| = 1/\text{poly}(\lambda)$ . Therefore, any constrained IPF that admits a puncturing constraint, or more generally, a constraint that permits evaluation of the IPF on all but  $\text{poly}(\lambda)$  points in the domain (Remark 3.17), must have a range that is larger than the domain by at least a super-polynomial factor.

**Remark 3.19** (Weaker Security Relations). The impossibility results in Remarks 3.15 and 3.17 only apply when we require that the IPF value at a punctured point appears pseudorandom given the constrained key. One way to circumvent the lower bound is to consider a weaker security notion where we just require the IPF value at a punctured point to be *unpredictable* rather than pseudorandom (given the constrained key). In other words, no efficient adversary should be able to predict  $F(k, x)$  given a constrained key  $k_f$  that does not allow evaluation at  $x$ . While the weaker security properties are potentially satisfiable, they may not be sufficient for specific applications.

## 4 Constructing Constrained IPFs

We now turn to constructing constrained IPFs and give two main constructions in this section. Our main constructions use private constrained (non-invertible) PRFs as the primary tool. As a warm-up, we first construct a puncturable IPF from a privately puncturable PRF in Section 4.1. We then show how the basic IPF construction can be extended to obtain a (single-key) circuit-constrained IPF in Section 4.2. In Section 7, we also show that an indistinguishability obfuscation of the basic puncturable IPF gives a *multi-key* circuit-constrained IPF.

## 4.1 Warm-up: Puncturable IPF from Private Puncturable PRFs

We begin by showing how to construct a puncturable IPF on a domain  $\mathcal{X}$  from a privately puncturable PRF on  $\mathcal{X}$ . We describe the construction and then show in Theorems 4.2 and 4.3 that it is a secure puncturable IPF.

**Construction 4.1.** Fix a domain  $\mathcal{X} = \{0, 1\}^n$  where  $n = n(\lambda)$ . Let  $F_1: \mathcal{K}_1 \times \mathcal{X} \rightarrow \mathcal{V}$  be an injective puncturable PRF with key-space  $\mathcal{K}_1$  and range  $\mathcal{V}$ . Let  $F_2: \mathcal{K}_2 \times \mathcal{V} \rightarrow \mathcal{X}$  be a privately puncturable PRF with key-space  $\mathcal{K}_2$ . The puncturable IPF  $F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  with key-space  $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2$ , domain  $\mathcal{X}$ , and range  $\mathcal{Y} = \mathcal{V} \times \mathcal{X}$  is defined as follows:

- The IPF key is a pair of keys  $k = (k^{(1)}, k^{(2)}) \in \mathcal{K}_1 \times \mathcal{K}_2$  for the puncturable PRFs  $F_1$  and  $F_2$ .
- On input  $k = (k^{(1)}, k^{(2)}) \in \mathcal{K}_1 \times \mathcal{K}_2 = \mathcal{K}$ , and  $x \in \mathcal{X}$  the IPF is defined as the pair

$$F((k^{(1)}, k^{(2)}), x) := \left( F_1(k^{(1)}, x), x \oplus F_2(k^{(2)}, F_1(k^{(1)}, x)) \right).$$

- On input  $k = (k^{(1)}, k^{(2)}) \in \mathcal{K}_1 \times \mathcal{K}_2 = \mathcal{K}$ , and  $y = (y_1, y_2) \in \mathcal{V} \times \mathcal{X} = \mathcal{Y}$ , the inversion algorithm  $F^{-1}(k, y)$  first computes  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$  and outputs

$$F^{-1}(k, (y_1, y_2)) := \begin{cases} x & \text{if } y_1 = F_1(k^{(1)}, x) \\ \perp & \text{otherwise.} \end{cases}$$

Next, we define the setup and constraining algorithms for  $(F, F^{-1})$ .

- **F.Setup**( $1^\lambda$ ): On input the security parameter  $\lambda$ , the setup algorithm samples two puncturable PRF keys  $k^{(1)} \leftarrow F_1.\text{Setup}(1^\lambda)$  and  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$ . The setup algorithm outputs the IPF key  $k = (k^{(1)}, k^{(2)})$ .
- **F.Puncture**( $k, x^*$ ): On input the IPF key  $k = (k^{(1)}, k^{(2)})$  and a point  $x^* \in \mathcal{X}$  to be punctured, the puncturing algorithm first computes  $v^* \leftarrow F_1(k^{(1)}, x^*)$ . It then generates two punctured keys  $k_{x^*}^{(1)} \leftarrow F_1.\text{Puncture}(k^{(1)}, x^*)$  and  $k_{v^*}^{(2)} \leftarrow F_2.\text{Puncture}(k^{(2)}, v^*)$  and returns  $k_{x^*} = (k_{x^*}^{(1)}, k_{v^*}^{(2)})$ .
- **F.Eval**( $k_{x^*}, x$ ): On input the punctured key  $k_{x^*} = (k_{x^*}^{(1)}, k_{v^*}^{(2)})$  and a point  $x \in \mathcal{X}$ , the evaluation algorithm first computes  $y_1 \leftarrow F_1.\text{Eval}(k_{x^*}^{(1)}, x)$  and returns  $y = (y_1, F_2.\text{Eval}(k_{v^*}^{(2)}, y_1) \oplus x)$ .
- **F.Eval**<sup>-1</sup>( $k_{x^*}, y$ ): On input the punctured key  $k_{x^*} = (k_{x^*}^{(1)}, k_{v^*}^{(2)})$ , and  $y = (y_1, y_2) \in \mathcal{V} \times \mathcal{X} = \mathcal{Y}$ , the inversion algorithm first computes  $x \leftarrow F_2.\text{Eval}(k_{v^*}^{(2)}, y_1) \oplus y_2$ . It returns  $x$  if  $F_1.\text{Eval}(k_{x^*}^{(1)}, x) = y_1$  and  $\perp$  otherwise.

We now state our correctness and security theorems, but defer their formal proofs to Appendix A.

**Theorem 4.2.** *Suppose  $F_1$  is an injective puncturable PRF and  $F_2$  is a puncturable PRF. Then, the IPF  $(F, F^{-1})$  from Construction 4.1 is correct.*

**Theorem 4.3.** *Suppose  $F_1$  is a selectively-secure puncturable PRF,  $F_2$  is a selectively-secure, privately puncturable PRF, and  $|\mathcal{X}| / |\mathcal{V}| = \text{negl}(\lambda)$ . Then  $(F, F^{-1})$  from Construction 4.1 is a selectively-secure puncturable IPF.*

**Remark 4.4** (Adaptive Security). Theorem 4.3 shows that if the underlying puncturable PRFs in Construction 4.1 are selectively secure, then the resulting IPF is selectively secure. We note that if we instantiate the underlying PRFs with an adaptively-secure (privately) puncturable PRF (for instance, the construction due to Canetti and Chen [CC17]), then the resulting IPF can also be shown to be adaptively secure (following a similar argument as that used in the proof of Theorem 4.3).

## 4.2 Circuit-Constrained IPF from Private Circuit-Constrained PRFs

In this section, we show how to extend our puncturable IPF construction from Section 4.1 to obtain a (single-key) constrained IPF for arbitrary circuit constraints. Our security analysis for our circuit-constrained IPF construction relies critically on the assumption that one of the underlying PRFs is a circuit-constrained PRF satisfying a strong simulation-based notion of privacy (Definition 3.14). Canetti and Chen [CC17] previously showed that even a 2-key private constrained PRF satisfying this simulation-based notion of privacy implies virtual black-box (VBB) obfuscation for the same underlying circuit class. Since VBB obfuscation for all circuits is impossible in the standard model [BGI<sup>+</sup>01], our construction is instantiatable only in the single-key setting, and thus, we present our construction in the single-key setting.

**Construction 4.5.** Fix a domain  $\mathcal{X} = \{0, 1\}^n$  where  $n = n(\lambda)$ . Our circuit-constrained IPF construction for  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ) relies on several primitives:

- Let  $\text{PKE} = (\text{PKE.Setup}, \text{PKE.Encrypt}, \text{PKE.Decrypt})$  be a PKE scheme with message space  $\mathcal{X}$ , ciphertext space  $\mathcal{T}$ , and whose decryption function can be computed in  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ). Let  $\mathcal{PK}$  and  $\mathcal{SK}$  denote the space of public keys and the space of secret keys, respectively, for PKE. Let  $\mathcal{V}$  denote the space from which the randomness for encryption is sampled.
- Let  $F_1 : \mathcal{K}_1 \times \mathcal{X} \rightarrow \mathcal{V}$  be a circuit-constrained PRF for  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ).
- Let  $F_2 : \mathcal{K}_2 \times \mathcal{T} \rightarrow \mathcal{X}$  be a private circuit-constrained PRF for  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ).<sup>3</sup>

The constrained IPF  $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  with key-space  $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2 \times \mathcal{PK} \times \mathcal{SK}$ , domain  $\mathcal{X}$ , and range  $\mathcal{Y} \subseteq \mathcal{T} \times \mathcal{X}$  is defined as follows:

- The IPF key consists of two PRF keys  $(k^{(1)}, k^{(2)}) \in \mathcal{K}_1 \times \mathcal{K}_2$  for  $F_1$  and  $F_2$ , respectively, and a public/secret key-pair  $(\text{pk}, \text{sk}) \in \mathcal{PK} \times \mathcal{SK}$  for the public-key encryption scheme PKE.
- On input a key  $k = (k^{(1)}, k^{(2)}, \text{pk}, \text{sk}) \in \mathcal{K}$ , and  $x \in \mathcal{X}$ , the IPF  $F(k, x)$  computes randomness  $r_x \leftarrow F_1(k^{(1)}, x)$ , a ciphertext  $\text{ct} \leftarrow \text{PKE.Encrypt}(\text{pk}, x; r_x)$ , and outputs

$$F(k, x) := \left( \text{ct}, F_2(k^{(2)}, \text{ct}) \oplus x \right).$$

Note that the public key  $\text{pk}$  can also be included as part of the public parameters for the IPF.

<sup>3</sup>To simplify the presentation, we implicitly assume that the PRFs  $F_1$  and  $F_2$  support general circuit constraints (i.e.,  $\text{NC}^1$  constraints or  $\text{P/poly}$  constraints). However, we can also instantiate our construction using private constrained PRFs for weaker constraint classes, provided that the constraint class is expressive enough to include the decryption algorithm for a CCA-secure public-key encryption scheme (see Remark 4.8).

- On input a key  $k = (k^{(1)}, k^{(2)}, \text{pk}, \text{sk}) \in \mathcal{K}$ , and  $(y_1, y_2) \in \mathcal{Y}$ , the inversion function  $F^{-1}(k, (y_1, y_2))$  first computes  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$  and  $r_x \leftarrow F_1(k^{(1)}, x)$ . Finally, it outputs

$$F^{-1}(k, (y_1, y_2)) := \begin{cases} x & \text{if } y_1 = \text{PKE.Encrypt}(\text{pk}, x; r_x) \\ \perp & \text{otherwise.} \end{cases}$$

- The range of the IPF  $\mathcal{Y}$  is defined to be  $\mathcal{T}' \times \mathcal{X}$  where  $\mathcal{T}' = \{\text{PKE.Encrypt}(\text{pk}, x; r)\}_{x \in \mathcal{X}, r \in \mathcal{V}}$  is the subset of ciphertexts that correspond to a valid encryption of some message under the public key  $\text{pk}$ .

Next, we define the setup and constraining algorithms for  $(F, F^{-1})$ .

- $F.\text{Setup}(1^\lambda)$ : On input the security parameter  $\lambda$ , the setup algorithm samples two PRF keys  $k^{(1)} \leftarrow F_1.\text{Setup}(1^\lambda)$ ,  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$ , and a public/secret key-pair for the PKE scheme:  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Setup}(1^\lambda)$ . It outputs the IPF key  $k = (k^{(1)}, k^{(2)}, \text{pk}, \text{sk})$ .
- $F.\text{Constrain}(k, f)$ : On input the IPF key  $k = (k^{(1)}, k^{(2)}, \text{pk}, \text{sk})$  and a constraint function  $f \in \mathcal{F}$ , the algorithm first constrains  $k_f^{(1)} \leftarrow F_1.\text{Constrain}(k^{(1)}, f)$ . Then, it defines the function  $F_{\text{sk}, f} : \mathcal{T} \rightarrow \{0, 1\}$  as follows:

$$F_{\text{sk}, f}(\text{ct}) := \begin{cases} 1 & \text{if } \text{PKE.Decrypt}(\text{sk}, \text{ct}) \neq \perp \text{ and } f(\text{PKE.Decrypt}(\text{sk}, \text{ct})) = 1 \\ 0 & \text{otherwise.} \end{cases} \quad (4.1)$$

The constrain algorithm constrains the key  $k^{(2)}$  to  $F_{\text{sk}, f}$  and obtains  $k_F^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, F_{\text{sk}, f})$ . It then defines and returns the constrained key  $k_f = (k_f^{(1)}, k_F^{(2)}, \text{pk})$ . Note that if  $\text{PKE.Decrypt}(\text{sk}, \cdot)$  can be computed in  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ), then the function  $F_{\text{sk}, f}$  can also be computed in  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ).

- $F.\text{Eval}(k_f, x)$ : On input the constrained key  $k_f = (k_f^{(1)}, k_F^{(2)}, \text{pk})$ , and a point  $x \in \mathcal{X}$ , the algorithm first computes  $r_x \leftarrow F_1.\text{Eval}(k_f^{(1)}, x)$ . Then, it encrypts  $\text{ct} \leftarrow \text{PKE.Encrypt}(\text{pk}, x; r_x)$  and returns the tuple  $y = (\text{ct}, F_2.\text{Eval}(k_F^{(2)}, \text{ct}) \oplus x)$ .
- $F.\text{Eval}^{-1}(k_f, y)$ : On input the constrained key  $k_f = (k_f^{(1)}, k_F^{(2)}, \text{pk})$ , and a point  $y = (y_1, y_2) \in \mathcal{Y}$ , the algorithm first computes  $x \leftarrow F_2.\text{Eval}(k_F^{(2)}, y_1) \oplus y_2$ . Then, it computes  $r_x \leftarrow F_1.\text{Eval}(k_f^{(1)}, x)$  and  $\text{ct} \leftarrow \text{PKE.Encrypt}(\text{pk}, x; r_x)$ . If  $y_1 = \text{ct}$ , then the algorithm returns  $x$ . Otherwise, it returns  $\perp$ .

We now state our correctness and security theorems, but defer their formal proofs to Appendix B.

**Theorem 4.6.** *Suppose PKE is a perfectly correct PKE scheme, and  $F_1, F_2$  are circuit-constrained PRFs for  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ). Then, the IPF  $(F, F^{-1})$  from Construction 4.5 is a circuit-constrained IPF for  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ).*

**Theorem 4.7.** *Suppose PKE is a perfectly correct, CCA-secure public-key encryption scheme,  $F_1$  is a single-key selective-function-secure circuit-constrained PRF for  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ), and  $F_2$  is a single-key, selective-function-secure private circuit-constrained PRF for  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ). Then,  $(F, F^{-1})$  from Construction 4.5 is a single-key, selective-function-secure circuit-constrained IPF for  $\text{NC}^1$  (resp.,  $\text{P/poly}$ ).*

**Remark 4.8** (Weaker Constraint Classes). While Construction 4.5 gives a circuit-constrained IPF from private circuit-constrained PRFs, the same construction also applies for building constrained PRFs that support a weaker class of constraints. Specifically, given a private constrained PRF for some constraint family  $\mathcal{F}$ , if  $\mathcal{F}$  is expressive enough to support the decryption operation of a CCA-secure PKE scheme (composed with the constraining function), then the constrained PRF for  $\mathcal{F}$  can be leveraged to construct an IPF for the family  $\mathcal{F}$  (via Construction 4.5).

## 5 Concrete Instantiations of Constrained IPFs

In this section, we describe how to concretely instantiate Constructions 4.1 and 4.5 using existing lattice-based private constrained PRFs [BKM17, CC17, BTVW17] to obtain puncturable IPFs and circuit-constrained IPFs (for both  $\text{NC}^1$  and  $\text{P/poly}$ ), respectively, from standard lattice assumptions.

**Puncturable IPFs from lattices.** To apply Construction 4.1, we require an injective puncturable PRF and a private puncturable PRF. As shown in [SW14], (statistically) injective puncturable PRFs<sup>4</sup> can be built from any one-way function. Next, the recent works of [BKM17, CC17, BTVW17] show how to construct private puncturable PRFs from standard lattice assumptions. Thus, applying Construction 4.1, we obtain puncturable IPFs from standard lattice assumptions. In fact, the construction of Canetti and Chen [CC17] gives an adaptively-secure private puncturable PRF from the (polynomial) hardness of the learning with errors (LWE) problem [Reg05], and so, combining their construction with Theorem 4.3, we obtain an adaptively-secure puncturable IPF from the (polynomial) hardness of LWE with subexponential error rate.

**Circuit-constrained IPFs from lattices.** Starting from (single-key) private circuit-constrained PRFs for  $\text{NC}^1$  [CC17] and  $\text{P/poly}$  [BTVW17], we can leverage Construction 4.5 to obtain (single-key) circuit-constrained IPFs for  $\text{NC}^1$  and  $\text{P/poly}$ , respectively. We give two candidate instantiations based on standard lattice assumptions:

- To construct a circuit-constrained IPF for  $\text{NC}^1$ -constraints, we require a private circuit-constrained PRF for  $\text{NC}^1$  and a CCA-secure public-key encryption scheme with an  $\text{NC}^1$  decryption circuit. We can instantiate the private circuit-constrained PRF for  $\text{NC}^1$  using the construction of Canetti and Chen [CC17]. The CCA-secure encryption scheme with  $\text{NC}^1$  decryption can be instantiated using existing lattice-based CCA-secure PKE schemes [PW08, Pei09, MP12] or by applying the Boneh et al. [BCHK07] transformation to a suitable identity-based encryption (IBE) scheme [GPV08, ABB10a, CHKP10, ABB10b] and a message authentication code (MAC) with verification in  $\text{NC}^1$ , which can be built from lattice-based PRFs [BPR12, BLMR13, BP14]. Putting these pieces together, we obtain a (single-key) circuit-constrained IPF for  $\text{NC}^1$  constraints from standard lattice assumptions.
- To construct a circuit-constrained IPF for  $\text{P/poly}$ , we primarily require a private constrained PRF for  $\text{P/poly}$ . We instantiate the private circuit-constrained PRF using the recent construction of Brakerski et al. [BTVW17], and the CCA-secure public key encryption as above. This yields a secure (single-key) circuit-constrained IPF for general predicates from standard lattice assumptions.

---

<sup>4</sup>A statistically injective puncturable PRF is a puncturable PRF  $F$  where  $F(k, \cdot)$  is injective with overwhelming probability over the choice of coins used for sampling the key  $k \leftarrow F.\text{Setup}(1^\lambda)$ .



**Remark 5.1** (Relaxed Notions of Correctness). Several lattice-based constrained PRF constructions [BV15, BKM17, BTVW17] satisfy a weaker “computational” notion of correctness which roughly states that an efficient adversary with a constrained key  $k_f$  cannot find an input  $x$  where  $\text{F.Eval}(k_f, x) \neq \text{F}(k, x)$ , where  $k$  is the PRF key. If we instantiate Constructions 4.1 and 4.5 with a constrained PRF that satisfies a computational notion of correctness, then the resulting constrained IPF also achieves computational correctness. It is straightforward to modify the correctness analysis (Theorem 4.2 and 4.6) to work under a computational notion of correctness. The security analysis remains unchanged since none of the proofs rely on perfect correctness of the underlying constrained PRFs.

## 6 An Extension: Supporting Delegation

In a delegatable constrained IPF, the holder of a constrained IPF key  $k_f$  for a function  $f$  can further constrain the key to some function  $g$  (i.e., construct a key  $k_{f \wedge g}$  that allows IPF evaluation only on points  $x$  where  $f(x) = g(x) = 1$ ). Many constrained PRF constructions either support or can be modified to support some flavor of key delegation [BW13, CRV14, DDM17]. In this section, we describe (informally) how to extend our constrained IPF construction from Section 4.2 to support key delegation.

**Delegatable constrained PRFs.** A constrained PRF that supports one level of delegation can be generically constructed from any constrained PRF by defining the PRF output to be the xor of the outputs of two constrained PRFs. For instance, we can define a PRF  $F$  as follows:

$$F((k_1, k_2), x) := F_1(k^{(1)}, x) \oplus F_2(k^{(2)}, x),$$

where  $F_1$  and  $F_2$  are constrained PRFs. The master secret key is  $k^{(1)}$  and  $k^{(2)}$ , and the constrained key for a function  $f$  is  $(k_f^{(1)}, k_f^{(2)})$  where  $k^{(1)} \leftarrow F_1.\text{Constrain}(k^{(1)}, f)$ . The holder of the constrained key  $(k_f^{(1)}, k_f^{(2)})$  can further constrain to a function of the form  $f \wedge g$  by computing  $(k_g^{(1)}, k_g^{(2)})$  where  $k_g^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, g)$ . Security of this construction follows by a simple hybrid argument. This general technique can be extended to support any a priori polynomially-bounded delegation depth.

**Delegatable constrained IPFs.** We can define a similar notion of key delegation for constrained IPFs. However, the above method of xoring together the outputs of several constrained IPFs does not directly give a delegatable constrained IPF. In fact, xoring together the outputs of several IPFs may not even give an injective function, let alone an efficiently invertible one. Thus, to support delegation for a constrained IPF, we need a different construction. One method is to use a variant of the xoring trick in conjunction with Construction 4.5. We describe a construction for achieving one level of delegation here. Our construction relies on a CCA-secure public-key encryption scheme PKE, three constrained PRFs  $F_1, F_2, F_3$ , and a constrained IPF IPF. The master secret key consists of keys  $k^{(1)}, k^{(2)}, k^{(3)}$  for  $F_1, F_2$ , and  $F_3$ , respectively, a key  $k^{(\text{IPF})}$  for IPF, and the public/secret

key-pair  $\text{pk}, \text{sk}$  for the PKE scheme. Our delegatable IPF works as follows:

$$\begin{aligned}
F((k^{(1)}, k^{(2)}, k^{(3)}, k^{(\text{IPF})}, \text{pk}, \text{sk}), x) &:= & F^{-1}((k^{(1)}, k^{(2)}, k^{(3)}, k^{(\text{IPF})}, \text{pk}, \text{sk}), (\text{ct}, z)) &:= \\
\left. \begin{array}{l} r \leftarrow F_1(k^{(1)}, x) \oplus F_3(k^{(3)}, x) \\ \text{ct} \leftarrow \text{PKE.Encrypt}(\text{pk}, x; r) \\ z \leftarrow F_2(k^{(2)}, \text{ct}) \oplus \text{IPF}(k^{(\text{IPF})}, x) \\ \text{output } (\text{ct}, z) \end{array} \right\} & & \left. \begin{array}{l} x \leftarrow \text{IPF}^{-1}(k^{(\text{IPF})}, z \oplus F_2(k^{(2)}, \text{ct})) \\ r \leftarrow F_1(k^{(1)}, x) \oplus F_3(k^{(3)}, x) \\ \text{if } \text{ct} \neq \text{PKE.Encrypt}(\text{pk}, x; r) \\ \quad \text{then } x \leftarrow \perp \\ \text{output } x \end{array} \right\}
\end{aligned}$$

To constrain a key  $(k^{(1)}, k^{(2)}, k^{(3)}, k^{(\text{IPF})}, \text{pk}, \text{sk})$  to a function  $f$ , we first constrain the PRF keys  $k^{(1)}, k^{(2)}$  exactly as described in Construction 4.5. In particular, the constrain algorithm computes  $k_f^{(1)} \leftarrow F_1.\text{Constrain}(k^{(1)}, f)$  and  $k_f^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, F_{\text{sk}, f})$ , where  $F_{\text{sk}, f}$  is defined as in Eq. (4.1). The constrained key is the tuple  $k_f = (k_f^{(1)}, k_f^{(2)}, k^{(3)}, k^{(\text{IPF})}, \text{pk})$ . To further constrain (that is, delegate) to a function  $g$ , we constrain  $F_3$  and IPF to  $g$ . In other words, we compute  $k_g^{(3)} \leftarrow F_3.\text{Constrain}(k^{(3)}, g)$  and  $k_g^{(\text{IPF})} \leftarrow \text{IPF}.\text{Constrain}(k^{(\text{IPF})}, g)$ . The constrained key  $k_{f \wedge g}$  for the function  $f \wedge g$  is defined to be  $k_{f \wedge g} := (k_f^{(1)}, k_f^{(2)}, k_g^{(3)}, k_g^{(\text{IPF})}, \text{pk})$ . Security of this construction follows by a similar argument as that used in the proof of Theorem 4.7 (namely, by appealing to security of  $F_1$  and privacy as well as security of  $F_2$ ), in addition to security of  $F_3$  and the underlying IPF. Our construction can be viewed as taking a standard constrained IPF (that does not support key delegation), and constructing a constrained IPF that supports one level of delegation. Iterating this construction multiple times yields an IPF that can support any a priori bounded number of delegations.

## 7 Multi-Key Constrained IPF from Obfuscation

In this section, we construct a *multi-key* circuit-constrained IPF from (polynomially-hard) indistinguishability obfuscation and one-way functions. Our construction of a circuit-constrained IPF from  $i\mathcal{O}$  (and one-way functions) mirrors the Boneh-Zhandry construction [BZ14] of a circuit-constrained PRF from  $i\mathcal{O}$  (and one-way functions). More precisely, Boneh and Zhandry show that obfuscating a puncturable PRF effectively gives a circuit-constrained PRF. Similarly, our construction works by obfuscating our punctured IPF construction (Construction 4.1) using  $i\mathcal{O}$ . In our construction, each constrained IPF key contains two obfuscated programs: one for evaluating the IPF, and one for inverting the IPF. The constraint function  $f$  is embedded within the obfuscated evaluation and inversion programs. We now describe our scheme more formally. First, we review the standard definition of indistinguishability obfuscation [BGI<sup>+</sup>01, GGH<sup>+</sup>13].

**Definition 7.1** (Indistinguishability Obfuscation [BGI<sup>+</sup>01, GGH<sup>+</sup>13]). An indistinguishability obfuscator  $i\mathcal{O}$  for a circuit class  $\mathcal{C}$  is a uniform and efficient algorithm satisfying the following requirements:

- **Correctness.** For all security parameter  $\lambda \in \mathbb{N}$ , all circuits  $C \in \mathcal{C}$ , and all inputs  $x$ , we have that

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

- **Indistinguishability.** For all security parameter  $\lambda \in \mathbb{N}$ , and any two circuits  $C_0, C_1 \in \mathcal{C}_\lambda$ , if  $C_0(x) = C_1(x)$  for all inputs  $x$ , then for all efficient adversaries  $\mathcal{A}$ , we have that

$$|\Pr[\mathcal{A}(i\mathcal{O}(C_0)) = 1] - \Pr[\mathcal{A}(i\mathcal{O}(C_1)) = 1]| = \text{negl}(\lambda).$$

**Construction 7.2.** Fix a domain  $\mathcal{X} = \{0, 1\}^n$  where  $n = n(\lambda)$ . Let  $F_1: \mathcal{K}_1 \times \mathcal{X} \rightarrow \mathcal{V}$  be a puncturable PRF with key-space  $\mathcal{K}_1$  and range  $\mathcal{V}$ . Let  $F_2: \mathcal{K}_2 \times \mathcal{V} \rightarrow \mathcal{X}$  be a puncturable PRF with key-space  $\mathcal{K}_2$ . The constrained IPF  $F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$  with key-space  $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2$ , domain  $\mathcal{X}$ , and range  $\mathcal{Y} = \mathcal{V} \times \mathcal{X}$  is defined as follows:

- The IPF key is a pair of keys  $k = (k^{(1)}, k^{(2)}) \in \mathcal{K}_1 \times \mathcal{K}_2 = \mathcal{K}$ . On input a key  $(k^{(1)}, k^{(2)})$  and an input  $x \in \mathcal{X}$ , the value of the IPF is defined to be

$$F(k, x) := \left( F_1(k^{(1)}, x), F_2(k^{(2)}, F_1(k^{(1)}, x)) \oplus x \right).$$

- On input  $k = (k^{(1)}, k^{(2)}) \in \mathcal{K}_1 \times \mathcal{K}_2 = \mathcal{K}$ , and  $y = (y_1, y_2) \in \mathcal{V} \times \mathcal{X} = \mathcal{Y}$ , the inversion algorithm  $F^{-1}(k, y)$  first computes  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$  and outputs

$$F^{-1}(k, (y_1, y_2)) := \begin{cases} x & \text{if } y_1 = F_1(k^{(1)}, x) \\ \perp & \text{otherwise.} \end{cases}$$

Next, we define the setup and constraining algorithms for the IPF  $(F, F^{-1})$ .

- **F.Setup** $(1^\lambda)$ : On input the security parameter  $\lambda$ , the setup algorithm samples two puncturable PRF keys  $k^{(1)} \leftarrow F_1.\text{Setup}(1^\lambda)$  and  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$ , and outputs  $k = (k^{(1)}, k^{(2)})$ .
- **F.Constrain** $(k, f)$ : On input the IPF key  $k = (k^{(1)}, k^{(2)})$  and a constraint function  $f \in \mathcal{F}$ , the constrain algorithm outputs two obfuscated programs  $P_0 = i\mathcal{O}(P^{\text{Eval}}[f, k^{(1)}, k^{(2)}])$  and  $P_1 = i\mathcal{O}(P^{\text{Inv}}[f, k^{(1)}, k^{(2)}])$  where the programs  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}]$  and  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}]$  are defined as follows:

**Constants:** a function  $f \in \mathcal{F}$ , and two keys  $k^{(1)}$  and  $k^{(2)}$  for  $F_1$  and  $F_2$ , respectively.

On input  $x \in \mathcal{X}$ :

1. If  $f(x) = 0$ , output  $\perp$ .
2. Otherwise, output  $F(k, x) = (F_1(k^{(1)}, x), F_2(k^{(2)}, F_1(k^{(1)}, x)) \oplus x)$ .

Figure 1: The program  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}]$

**Constants:** a function  $f \in \mathcal{F}$ , and two keys  $k^{(1)}$  and  $k^{(2)}$  for  $F_1$  and  $F_2$ , respectively.

On input  $y = (y_1, y_2) \in \mathcal{V} \times \mathcal{X}$

1. Compute  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$ .
2. If  $f(x) = 0$  or  $y_1 \neq F_1(k^{(1)}, x)$ , output  $\perp$ .
3. Otherwise, output  $x$ .

Figure 2: The program  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}]$

Note that the programs  $P^{\text{Eval}}$  and  $P^{\text{Inv}}$  are padded to the maximum size of any program that appears in the proof of Theorem 7.4.

- $F.\text{Eval}(k_f, x)$ : On input the constrained key  $k_f = (P_1, P_2)$ , and a point  $x \in \mathcal{X}$ , the evaluation algorithm outputs  $P_1(x)$ .
- $F.\text{Eval}^{-1}(k_f, y)$ : On input the constrained key  $k_f = (P_1, P_2)$ , and a point  $y \in \mathcal{Y}$ , the inversion algorithm outputs  $P_2(y)$ .

We now state our correctness and security theorems, but defer their formal proofs to Appendix C.

**Theorem 7.3.** *Suppose  $F_1$  and  $F_2$  are puncturable PRFs, and  $i\mathcal{O}$  is an indistinguishability obfuscator. Then, the IPF  $(F, F^{-1})$  from Construction 7.2 is correct.*

**Theorem 7.4.** *Suppose  $F_1$  and  $F_2$  are selectively-secure puncturable PRFs,  $i\mathcal{O}$  is an indistinguishability obfuscator, and  $|\mathcal{X}| / |\mathcal{Y}| = \text{negl}(\lambda)$ . Then  $(F, F^{-1})$  from Construction 7.2 is a selectively-secure circuit-constrained IPF.*

## Acknowledgments

This work was funded by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

## References

- [ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, 2010.
- [ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *CRYPTO*, 2010.
- [BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, 2004.
- [BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, 2007.
- [BCHK07] Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.*, 36(5), 2007.
- [BF14] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In *PKC*, 2014.
- [BFO08] Alexandra Boldyreva, Serge Fehr, and Adam O'Neill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In *CRYPTO*, 2008.
- [BFOR08] Mihir Bellare, Marc Fischlin, Adam O'Neill, and Thomas Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *CRYPTO*, 2008.

- [BGI<sup>+</sup>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*, 2001.
- [BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.
- [BKM17] Dan Boneh, Sam Kim, and Hart Montgomery. Private puncturable PRFs from standard lattice assumptions. In *EUROCRYPT*, 2017.
- [BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In *CRYPTO*, 2013.
- [BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In *PKC*, 2017.
- [BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT*, 2000.
- [BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In *CRYPTO*, 2014.
- [BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, 2012.
- [BR00] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In *ASIACRYPT*, 2000.
- [BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. New tools in LWE-based cryptography: Constraint-hiding PRF and predicate encryption, 2017. Preprint.
- [BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC*, 2015.
- [BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.
- [BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.
- [CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC1 from LWE. In *EUROCRYPT*, 2017.
- [CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, 2010.
- [CRV14] Nishanth Chandran, Srinivasan Raghuraman, and Dhinakaran Vinayagamurthy. Constrained pseudorandom functions: Verifiable and delegatable. *IACR Cryptology ePrint Archive*, 2014, 2014.

- [DDM17] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Constrained pseudorandom functions for unconstrained inputs revisited: Achieving verifiability and key delegation. In *PKC*, 2017.
- [DKW16] Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. In *EUROCRYPT*, 2016.
- [FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained prfs. In *ASIACRYPT*, 2014.
- [FOR12] Benjamin Fuller, Adam O'Neill, and Leonid Reyzin. A unified approach to deterministic encryption: New constructions and a connection to computational entropy. In *TCC*, 2012.
- [Fuc14] Georg Fuchsbauer. Constrained verifiable random functions. *IACR Cryptology ePrint Archive*, 2014, 2014.
- [GGH<sup>+</sup>13] 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.
- [GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *FOCS*, 1984.
- [GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.
- [HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In *ASIACRYPT*, 2015.
- [Hof14] Dennis Hofheinz. Fully secure constrained pseudorandom functions using random oracles. *IACR Cryptology ePrint Archive*, 2014, 2014.
- [IY09a] Tetsu Iwata and Kan Yasuda. Btm: A single-key, inverse-cipher-free mode for deterministic authenticated encryption. In *SAC*, 2009.
- [IY09b] Tetsu Iwata and Kan Yasuda. Hbs: A single-key mode of operation for deterministic authenticated encryption. In *FSE*, 2009.
- [KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS*, 2013.
- [KY00] Jonathan Katz and Moti Yung. Unforgeable encryption and adaptively secure modes of operations. In *FSE*, 2000.
- [LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2), 1988.
- [MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

- [NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, 1990.
- [Pei09] Chris Peikert. Bonsai trees (or, arboriculture in lattice-based cryptography). *IACR Cryptology ePrint Archive*, 2009, 2009.
- [PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC*, 2008.
- [RBB03] Phillip Rogaway, Mihir Bellare, and John Black. Ocb: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security (TISSEC)*, 6(3), 2003.
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.
- [Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In *ACM CCS*, 2002.
- [RS91] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*, 1991.
- [RS06] Phillip Rogaway and Thomas Shrimpton. Deterministic authenticated encryption: A provable-security treatment of the key-wrap problem. In *EUROCRYPT*, 2006.
- [SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

## A Analysis of Puncturable IPF

In this section, we give the formal correctness and security analysis of our puncturable IPF (Construction 4.1) from Section 4.1.

### A.1 Proof of Theorem 4.2

Correctness of Construction 4.1 follows from correctness of the underlying puncturable PRFs. Specifically, let  $k = (k^{(1)}, k^{(2)}) \leftarrow \text{F.Setup}(1^\lambda)$ . Take any  $x^* \in \mathcal{X}$ , and let  $k_{x^*} = (k_{x^*}^{(1)}, k_{x^*}^{(2)}) \leftarrow \text{F.Puncture}(k, x^*)$ . We show the two correctness requirements separately:

- Take any  $x \neq x^*$ . Suppose  $F(k, x) = (y_1, y_2)$ , which means that  $y_1 = F_1(k^{(1)}, x)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x$ . Since  $F_1$  is injective and  $x \neq x^*$ ,  $F_1(k^{(1)}, x) \neq F_1(k^{(1)}, x^*) = v^*$ . By correctness of  $F_1$  and  $F_2$ , we have that  $F_1.\text{Eval}(k_{x^*}^{(1)}, x) = F_1(k^{(1)}, x) = y_1$  and  $F_2.\text{Eval}(k_{x^*}^{(2)}, y_1) \oplus x = F_2(k^{(2)}, y_1) \oplus x = y_2$ .
- Take any input  $(y_1, y_2) \in \mathcal{V} \times \mathcal{X}$  where there exists some  $x \neq x^*$  such that  $F(k, x) = (y_1, y_2)$ . This means that  $y_1 = F_1(k^{(1)}, x)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x$ . Moreover, since  $F_1$  is injective and  $x \neq x^*$ ,  $y_1 \neq F_1(k^{(1)}, x^*) = v^*$ . By correctness of  $F_2$ , it follows that  $F_2.\text{Eval}(k_{x^*}^{(2)}, y_1) \oplus y_2 = F_2(k^{(2)}, y_1) \oplus y_2 = x$ . By correctness of  $F_1$ ,  $F_1.\text{Eval}(k_{x^*}^{(1)}, x) = F_1(k^{(1)}, x) = y_1$ , in which case  $F.\text{Eval}^{-1}(k_{x^*}, (y_1, y_2)) = x = F^{-1}(k, (y_1, y_2))$ .  $\square$

## A.2 Proof of Theorem 4.3

Our proof proceeds via a sequence of hybrid experiments between an adversary  $\mathcal{A}$  and a challenger. Each of our hybrid experiments consist of the following phases:

1. **Setup phase.** In the selective security experiment, the adversary begins by committing to a point  $x^* \in \mathcal{X}$ . Then, the challenger generates the IPF key  $k \in \mathcal{K}$ . The challenger constructs a punctured key  $k_{x^*}$  and samples a challenge value  $y^* \in \mathcal{Y}$ . The challenger gives  $k_{x^*}$  and  $y^*$  to the adversary.
2. **Query phase.** The adversary  $\mathcal{A}$  is now allowed to issue evaluation and inversion queries to the challenger. However, the adversary is not allowed to query the evaluation oracle on the punctured point  $x^*$  or the inversion oracle on its challenge  $y^*$ .
3. **Output phase.** At the end of the experiment, the adversary outputs a bit  $b \in \{0, 1\}$ .

We now define our sequence of hybrid experiments. When defining a new hybrid, we only describe the phases that differ from the previous one.

- **Hyb<sub>0</sub>:** This is the constrained IPF security experiment  $\text{Expt}_{\mathcal{A}, \mathcal{F}}^{(\text{IPF})}(\lambda, 0)$  from Definition 3.11. In the setup phase, after the adversary commits to a point  $x^* \in \mathcal{X}$ , the challenger samples keys  $k^{(1)} \leftarrow \mathcal{F}_1.\text{Setup}(1^\lambda)$  and  $k^{(2)} \leftarrow \mathcal{F}_2.\text{Setup}(1^\lambda)$ , and sets  $k = (k^{(1)}, k^{(2)})$ . It sets  $v^* \leftarrow \mathcal{F}_1(k^{(1)}, x^*)$  and  $z^* \leftarrow \mathcal{F}_2(k^{(2)}, v^*) \oplus x^*$  and constructs the constrained keys  $k_{x^*}^{(1)} \leftarrow \mathcal{F}_1.\text{Puncture}(k^{(1)}, x^*)$  and  $k_{v^*}^{(2)} \leftarrow \mathcal{F}_2.\text{Puncture}(k^{(2)}, v^*)$ . The challenger gives the punctured key  $k_{x^*} = (k_{x^*}^{(1)}, k_{v^*}^{(2)})$  and evaluation  $y^* = (v^*, z^*)$  to the adversary. During the query phase, the challenger answers all of the evaluation queries by computing  $\mathcal{F}(k, \cdot)$  and the inversion queries by computing  $\mathcal{F}^{-1}(k, \cdot)$ , exactly as in the real scheme.
- **Hyb<sub>1</sub>:** Same as Hyb<sub>0</sub>, except during the setup phase, the challenger samples  $v^* \xleftarrow{\mathcal{R}} \mathcal{V}$ . The rest of the setup phase proceeds as in Hyb<sub>0</sub>. When responding to the evaluation and inversion queries, the challenger uses  $v^*$  in place of the value  $\mathcal{F}_1(k^{(1)}, x^*)$ .
- **Hyb<sub>2</sub>:** Same as Hyb<sub>1</sub>, except when responding to the evaluation and inversion queries, the challenger always evaluates  $\mathcal{F}_1(k^{(1)}, \cdot)$  instead of substituting the value  $v^*$  for  $\mathcal{F}_1(k^{(1)}, x^*)$ .
- **Hyb<sub>3</sub>:** Same as Hyb<sub>2</sub>, except during the setup phase, the challenger samples  $z^* \xleftarrow{\mathcal{R}} \mathcal{X}$ . The rest of the setup phase proceeds as in Hyb<sub>2</sub>. When responding to the evaluation and inversion queries, the challenger uses  $z^* \oplus x^*$  in place of the value  $\mathcal{F}_2(k^{(2)}, v^*)$ .
- **Hyb<sub>4</sub>:** Same as Hyb<sub>3</sub>, except when responding to the evaluation and inversion queries, the challenger always evaluates  $\mathcal{F}_2(k^{(2)}, \cdot)$  instead of substituting the value  $z^* \oplus x^*$  for  $\mathcal{F}_2(k^{(2)}, v^*)$ .
- **Hyb<sub>5</sub>:** Same as Hyb<sub>4</sub>, except during the setup phase, the challenger punctures  $k^{(2)}$  at  $\mathcal{F}_1(k^{(1)}, x^*)$  instead of  $v^*$ . This is the constrained IPF security experiment  $\text{Expt}_{\mathcal{A}, \mathcal{F}}^{(\text{IPF})}(\lambda, 1)$  from Definition 3.6.

For an adversary  $\mathcal{A}$ , we write  $\text{Hyb}_i(\mathcal{A})$  to denote the output in hybrid Hyb<sub>i</sub>. We now argue that the output of each consecutive pair of experiments is computationally indistinguishable. In the following, we implicitly assume that the adversary in each pair of hybrid arguments is admissible.



**Lemma A.1.** *If  $F_1$  is a selectively-secure puncturable PRF (Definition 3.7), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_1(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Suppose there exists an adversary  $\mathcal{A}$  that can distinguish between hybrids  $\text{Hyb}_0$  and  $\text{Hyb}_1$ . We construct an adversary  $\mathcal{B}$  to break puncturing security of  $F_1$ . Algorithm  $\mathcal{B}$  simulates the experiments as follows:

- **Setup phase.** At the beginning of the experiment, adversary  $\mathcal{A}$  commits to a challenge point  $x^* \in \mathcal{X}$ . Algorithm  $\mathcal{B}$  sends  $x^*$  to the puncturing security challenger for  $F_1$  and receives a punctured PRF key  $k_{x^*}^{(1)}$  and a challenge value  $v^* \in \mathcal{V}$ . It generates the key  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$ , the punctured key  $k_{v^*}^{(2)} \leftarrow F_2.\text{Puncture}(k^{(2)}, v^*)$ , and the evaluation  $z^* \leftarrow F_2(k^{(2)}, v^*) \oplus x^*$  exactly as in the real scheme. Finally, it sends the punctured key  $k_{x^*} = (k_{x^*}^{(1)}, k_{v^*}^{(2)})$  and the challenge evaluation  $y^* = (v^*, z^*)$  to  $\mathcal{A}$ .
- **Query phase.** Algorithm  $\mathcal{B}$  simulates the query phase exactly as in  $\text{Hyb}_0$  and  $\text{Hyb}_1$ , except whenever it needs to compute  $F_1(k^{(1)}, x)$  on some  $x \neq x^*$ , it queries the evaluation oracle for  $F_1$  on  $x$ . Whenever it needs to compute  $F_1(k^{(1)}, x^*)$ , it instead uses the value  $v^*$ .
- **Output phase.** After  $\mathcal{A}$  outputs a bit  $b \in \{0, 1\}$ , algorithm  $\mathcal{B}$  outputs the same bit  $b$ .

We now argue that if  $v^*$  is the actual value of  $F_1$  at the punctured point  $x^*$ , then  $\mathcal{B}$  perfectly simulates  $\text{Hyb}_0$ , and if  $v^*$  is a uniformly random value, then it has perfectly simulated  $\text{Hyb}_1$ . In the analysis, we write  $k^{(1)}$  to denote the key sampled by the puncturing security challenger for  $F_1$  (unknown to the reduction algorithm  $\mathcal{B}$ ). First, note that  $\mathcal{B}$  is admissible for the puncturing security game because it never queries the evaluation oracle for  $F_1$  on the challenge point  $x^*$ .

- Suppose  $v^* = F_1(k^{(1)}, x^*)$ . Then, the setup phase is simulated exactly as in  $\text{Hyb}_0$ . It suffices to argue that the evaluation and inversion queries are simulated correctly. In the reduction,  $\mathcal{B}$  queries the evaluation oracle to obtain the values  $F_1(k_{x^*}^{(1)}, x)$  whenever  $x \neq x^*$ , and it uses  $v^* = F_1(k^{(1)}, x^*)$  for the value of  $F_1(k^{(1)}, x^*)$ . Thus,  $\mathcal{B}$  perfectly simulates  $\text{Hyb}_0$ .
- Suppose  $v^*$  is uniformly random over  $\mathcal{V}$ . In this case,  $\mathcal{B}$  perfectly simulates the setup phase of  $\text{Hyb}_1$ . In the query phase, algorithm  $\mathcal{B}$  uses the evaluation oracle to compute  $F_1(k^{(1)}, x)$  whenever  $x \neq x^*$ , and the value  $v^*$  for  $F_1(k^{(1)}, x^*)$ . This is precisely the distribution in  $\text{Hyb}_1$ .

We conclude that  $\mathcal{B}$  breaks the puncturing security of  $F_1$  with the same advantage adversary  $\mathcal{A}$  has in distinguishing  $\text{Hyb}_0$  and  $\text{Hyb}_1$ . The lemma follows.  $\square$

**Lemma A.2.** *If  $F_1$  is a selectively-secure puncturable PRF (Definition 3.7), and  $1/|\mathcal{V}| = \text{negl}(\lambda)$ , then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_1(\mathcal{A}) = 1] - \Pr[\text{Hyb}_2(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* The only difference between  $\text{Hyb}_1$  and  $\text{Hyb}_2$  is that during the query phase, the challenger uses the value  $v^*$  in place of the value  $F_1(k^{(1)}, x^*)$  in  $\text{Hyb}_1$ . To show the lemma, we consider each type of query separately, and argue that the view of the adversary is computationally indistinguishable between hybrids  $\text{Hyb}_1$  and  $\text{Hyb}_2$ .

- **Evaluation queries.** On an evaluation query  $x$ , the challenger in  $\text{Hyb}_1$  and  $\text{Hyb}_2$  only needs to evaluate  $F_1(k^{(1)}, \cdot)$  on  $x$ . Since  $\text{Hyb}_1$  and  $\text{Hyb}_2$  only differ in the value the challenger uses for  $F_1(k^{(1)}, x^*)$ , the outputs of the evaluation oracle in  $\text{Hyb}_1$  and  $\text{Hyb}_2$  are identically distributed

on all inputs  $x \neq x^*$ . By admissibility, the adversary can only query the evaluation oracle on points  $x \neq x^*$ , and so, the responses of the evaluation oracle in  $\text{Hyb}_1$  and  $\text{Hyb}_2$  are distributed identically.

- **Inversion queries.** On an inversion query  $(y_1, y_2)$ , the challenger first computes  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$ . Then, the challenger checks whether  $y_1 = F_1(k^{(1)}, x)$ . If  $x \neq x^*$ , then the output of the inversion oracle in  $\text{Hyb}_1$  and  $\text{Hyb}_2$  is computed identically. It suffices to only consider the case where  $x = x^*$ , or equivalently, when  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ .
  - In  $\text{Hyb}_1$ , the challenger substitutes the value  $v^*$  for the value  $F_1(k^{(1)}, x^*)$ . For any query  $(y_1, y_2)$  where  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ , the challenger in  $\text{Hyb}_1$  always responds with  $\perp$  unless  $y_1 = v^*$ . But if  $y_1 = v^*$ , then it must be the case that  $y_2 = F_2(k^{(2)}, v^*) \oplus x^* = z^*$ . This means that  $(y_1, y_2) = (v^*, z^*)$ , which is the challenge query. Thus, the challenger in  $\text{Hyb}_1$  responds with  $\perp$  on all admissible inversion queries satisfying  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ .
  - In  $\text{Hyb}_2$ , on a query  $(y_1, y_2)$  where  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ , the challenger in  $\text{Hyb}_2$  responds with  $\perp$  unless  $y_1 = F_1(k^{(1)}, x^*)$ . This in particular means that  $y_2 = F_2(k^{(2)}, F_1(k^{(1)}, x^*)) \oplus x^*$ .

We conclude that the response of the challenger on an inversion query is identically distributed in  $\text{Hyb}_1$  and  $\text{Hyb}_2$  unless the adversary makes an inversion query on the tuple  $(y_1, y_2)$  where  $y_1 = F_1(k^{(1)}, x^*)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$ .

Suppose now that an adversary  $\mathcal{A}$  is able to distinguish between  $\text{Hyb}_1$  and  $\text{Hyb}_2$  with some non-negligible probability  $\varepsilon$ . By the above analysis, the adversary's view in  $\text{Hyb}_1$  and  $\text{Hyb}_2$  is *identically* distributed unless the adversary makes an inversion query on the input  $(y_1, y_2)$  where  $y_1 = F_1(k^{(1)}, x^*)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$ . Since  $\mathcal{A}$  distinguishes  $\text{Hyb}_1$  from  $\text{Hyb}_2$  with advantage  $\varepsilon$ , it must make an inversion query on  $(y_1, y_2)$  with probability at least  $\varepsilon$ . We use  $\mathcal{A}$  to construct an adversary  $\mathcal{B}$  that breaks the puncturing security of  $F_1$ . Algorithm  $\mathcal{B}$  simulates an execution of  $\text{Hyb}_1$  as follows:

- **Setup phase.** At the beginning of the experiment, adversary  $\mathcal{A}$  commits to a challenge point  $x^* \in \mathcal{X}$ . Algorithm  $\mathcal{B}$  sends  $x^*$  to the puncturing security challenger for  $F_1$  and receives a punctured PRF key  $k_{x^*}^{(1)}$  and a challenge value  $\hat{v}^* \in \mathcal{V}$ . Algorithm  $\mathcal{B}$  then samples a random point  $v^* \xleftarrow{\mathbb{R}} \mathcal{V}$ . It generates the key  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$ , the punctured key  $k_{v^*}^{(2)} \leftarrow F_2.\text{Puncture}(k^{(2)}, v^*)$ , and the evaluation  $z^* \leftarrow F_2(k^{(2)}, v^*) \oplus x^*$  exactly as in  $\text{Hyb}_1$  and  $\text{Hyb}_2$ . Finally, it sends the punctured key  $k_{x^*} = (k_{x^*}^{(1)}, k_{v^*}^{(2)})$  and the challenge evaluation  $y^* = (v^*, z^*)$  to  $\mathcal{A}$ . Importantly,  $\mathcal{B}$  does not use the challenge value  $\hat{v}^*$  in this phase.
- **Query phase.** All of the (admissible) evaluation queries are simulated as in  $\text{Hyb}_1$  and  $\text{Hyb}_2$ . In particular, whenever it needs to compute  $x = F_1(k^{(1)}, x)$  for  $x \neq x^*$ , it queries the evaluation oracle for  $F_1$  on the point  $x$ . For an inversion query  $(y_1, y_2)$ , algorithm  $\mathcal{B}$  first computes  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$ . If  $x \neq x^*$ , then  $\mathcal{B}$  proceeds as in  $\text{Hyb}_1$  and  $\text{Hyb}_2$ , except it queries the evaluation oracle for  $F_1$  to compute the value of  $F_1(k^{(1)}, x)$ . On (admissible) inversion queries  $(y_1, y_2)$  where  $x = F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ , algorithm  $\mathcal{B}$  responds with  $\perp$ .

At the end of the experiment (or if  $\mathcal{A}$  aborts),  $\mathcal{B}$  checks whether  $\mathcal{A}$  ever queried the inversion oracle on the tuple  $(\hat{v}^*, F_2(k^{(2)}, \hat{v}^*) \oplus x^*)$  where  $\hat{v}^*$  is the challenge it received from the puncturing security challenger. If  $\mathcal{A}$  made such a query, then  $\mathcal{B}$  outputs 1. Otherwise, it outputs 0. By construction, we note that  $\mathcal{B}$  never needs to evaluate  $F_1$  on the challenge point  $x^*$ , so it is admissible.

To complete the argument, we first note that by construction of  $\mathcal{B}$  and the above analysis, algorithm  $\mathcal{B}$  perfectly simulates the setup and query phases in  $\text{Hyb}_1$  for  $\mathcal{A}$ . This means that with probability at least  $\varepsilon$ ,  $\mathcal{A}$  will submit an inversion query on the tuple  $(y_1, y_2)$  where  $y_1 = F_1(k^{(1)}, x^*)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$  at some point during the challenge phase (otherwise, the view of  $\mathcal{A}$  is identically distributed in  $\text{Hyb}_1$  and  $\text{Hyb}_2$ ). We now consider the probability that  $\mathcal{B}$  outputs 1.

- Suppose  $\hat{v}^* = F_1(k^{(1)}, x^*)$ . By assumption,  $\mathcal{A}$  queries the inversion oracle on the tuple  $(y_1, y_2)$  where  $y_1 = F_1(k^{(1)}, x^*) = \hat{v}^*$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^* = F_2(k^{(2)}, \hat{v}^*) \oplus x^*$  with probability at least  $\varepsilon$ . Correspondingly, algorithm  $\mathcal{B}$  outputs 1 with probability at least  $\varepsilon$ .
- Suppose  $\hat{v}^*$  is random in  $\mathcal{V}$ . By construction of  $\mathcal{B}$ , the view of  $\mathcal{A}$  in the reduction is *independent* of  $\hat{v}^*$ . Let  $Q = \text{poly}(\lambda)$  be the number of inversion queries  $\mathcal{A}$  makes. Since  $\hat{v}^*$  is uniformly random over  $\mathcal{V}$ , the probability that  $\hat{v}^*$  is equal to one of the adversary's queries is at most  $Q/|\mathcal{V}| = \text{negl}(\lambda)$  since we assume that  $1/|\mathcal{V}| = \text{negl}(\lambda)$ .

We conclude that if  $\mathcal{A}$  is able to distinguish between hybrids  $\text{Hyb}_1$  and  $\text{Hyb}_2$  with non-negligible advantage  $\varepsilon$ , then  $\mathcal{B}$  is able to break puncturing security of  $F_1$  with probability at least  $\varepsilon - \text{negl}(\lambda)$ .  $\square$

**Lemma A.3.** *If  $F_2$  is a selectively-secure puncturable PRF (Definition 3.7), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_2(\mathcal{A}) = 1] - \Pr[\text{Hyb}_3(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* This follows very similarly to the proof of Lemma A.1. Suppose there exists an adversary  $\mathcal{A}$  that can distinguish between hybrids  $\text{Hyb}_2$  and  $\text{Hyb}_3$ . We construct an adversary to break the puncturing security of  $F_2$ . Algorithm  $\mathcal{B}$  simulates the experiments as follows:

- **Setup phase.** At the beginning of the experiment,  $\mathcal{A}$  commits to a point  $x^*$ . Algorithm  $\mathcal{B}$  generates  $k^{(1)} \leftarrow F_1.\text{Setup}(1^\lambda)$ ,  $k_{x^*}^{(1)} \leftarrow F_1.\text{Puncture}(k^{(1)}, x^*)$ , and chooses  $v^* \xleftarrow{\mathbb{R}} \mathcal{V}$ . It submits  $v^*$  to the puncturing security challenger for  $F_2$  and receives a punctured key  $k_{v^*}^{(2)}$  and a challenge value  $z^* \in \mathcal{X}$ . Algorithm  $\mathcal{B}$  gives  $k_{x^*} = (k_{x^*}^{(1)}, k_{v^*}^{(2)})$  and  $y^* = (v^*, z^* \oplus x^*)$  to  $\mathcal{A}$ .
- **Query phase.** Algorithm  $\mathcal{B}$  simulates the query phase exactly as in  $\text{Hyb}_2$  and  $\text{Hyb}_3$ , except whenever it needs to compute  $F_2(k^{(2)}, v)$  on some  $v \neq v^*$ , it queries the evaluation oracle for  $F_2$  on  $v$ . Whenever it needs to compute  $F_2(k^{(2)}, v^*)$ , it uses the value  $z^*$ .

At the end of the simulation,  $\mathcal{B}$  outputs whatever  $\mathcal{A}$  outputs. Note first that  $\mathcal{B}$  is admissible because it never needs to query the evaluation oracle on the challenge point  $v^*$ . We now show that depending on whether  $z^*$  is pseudorandom or truly random,  $\mathcal{B}$  either simulates  $\text{Hyb}_2$  or  $\text{Hyb}_3$  for  $\mathcal{A}$ :

- Suppose  $z^* = F_2(k^{(2)}, v^*)$ . Then, the setup and challenge phases are simulated as described in  $\text{Hyb}_2$ .
- Suppose  $z^*$  is uniformly random over  $\mathcal{X}$ . Since  $z^*$  is independent of  $x^*$ , this means that  $z^* \oplus x^*$  is also uniform over  $\mathcal{X}$ . Thus, the setup phase is simulated exactly as in  $\text{Hyb}_3$ . By construction,  $\mathcal{B}$  perfectly simulates the challenge phase in  $\text{Hyb}_2$ .

We conclude that  $\mathcal{B}$  breaks puncturing security of  $\mathcal{A}$  with the same advantage  $\mathcal{A}$  has in distinguishing  $\text{Hyb}_2$  and  $\text{Hyb}_3$ . The lemma follows.  $\square$

**Lemma A.4.** *If  $|\mathcal{X}| / |\mathcal{V}| = \text{negl}(\lambda)$ , then for all adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_3(\mathcal{A}) = 1] - \Pr[\text{Hyb}_4(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* The only difference between  $\text{Hyb}_3$  and  $\text{Hyb}_4$  is that during the query phase, the challenger uses the value  $z^* \oplus x^*$  in place of  $F_2(k^{(2)}, v^*)$  in  $\text{Hyb}_3$ . To show the lemma, we consider each type of query and argue that the views of the adversary is statistically indistinguishable in hybrids  $\text{Hyb}_3$  and  $\text{Hyb}_4$ .

- **Evaluation queries.** On an evaluation query  $x$ , the challenger in  $\text{Hyb}_3$  and  $\text{Hyb}_4$  first computes  $v \leftarrow F_1(k^{(1)}, x)$  and then computes  $F_2(k^{(2)}, v)$ . Thus, the response to the evaluation query in  $\text{Hyb}_3$  and  $\text{Hyb}_4$  differs only on points  $x \in \mathcal{X}$  where  $F_1(k^{(1)}, x) = v^*$ . But in  $\text{Hyb}_3$  and  $\text{Hyb}_4$ , the point  $v^*$  is sampled uniformly at random from  $\mathcal{V}$ , and in particular, independently of  $k^{(1)}$ . Thus, for any  $x \in \mathcal{X}$ , it follows that  $\Pr[v^* = F_1(k^{(1)}, x)] = 1/|\mathcal{V}|$ . Taking an union bound over all  $x \in \mathcal{X}$ , we have that

$$\Pr[\forall x: F_1(k^{(1)}, x) \neq v^*] = 1 - |\mathcal{X}|/|\mathcal{V}| = 1 - \text{negl}(\lambda).$$

Thus, with overwhelming probability (over the choice of  $v^*$ ),  $F_1(k^{(1)}, x) \neq v^*$  for all  $x \in \mathcal{X}$ . This means that with overwhelming probability, the outputs of the evaluation oracle in  $\text{Hyb}_3$  is the same as that in  $\text{Hyb}_4$  on all inputs.

- **Inversion queries.** On an inversion query  $(y_1, y_2)$ , the challenger in  $\text{Hyb}_3$  and  $\text{Hyb}_4$  first computes  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$  and then checks whether  $y_1 = F_1(k^{(1)}, x)$ . On all queries where  $y_1 \neq v^*$ , the challenger's behavior in  $\text{Hyb}_3$  and  $\text{Hyb}_4$  is identical. On the queries  $(v^*, y_2)$ , the only instance where the adversary does not output  $\perp$  is if  $F_1(k^{(1)}, x) = v^*$ . But as argued in the case of evaluation queries, with overwhelming probability over the choice of  $v^*$ , there does not exist any  $x \in \mathcal{X}$  such that this holds. Thus, regardless of the value the challenger uses for  $F_2(k^{(2)}, v^*)$ , the output in  $\text{Hyb}_3$  and  $\text{Hyb}_4$  will be  $\perp$  on all inversion queries of the form  $(v^*, y_2)$ . We conclude that with overwhelming probability, the output of the inversion oracle in  $\text{Hyb}_3$  is the same as that in  $\text{Hyb}_4$  on all inputs.

The above analysis shows that with overwhelming probability over the choice of  $v^*$ , the distribution of outputs of the evaluation and inversion oracles in  $\text{Hyb}_3$  and  $\text{Hyb}_4$  is statistically indistinguishable.  $\square$

**Lemma A.5.** *If  $F_1$  is a selectively-secure puncturable PRF (Definition 3.7),  $F_2$  is a selectively-private puncturable PRF (Definition 3.14), and  $|\mathcal{X}|/|\mathcal{V}| = \text{negl}(\lambda)$ , then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_4(\mathcal{A}) = 1] - \Pr[\text{Hyb}_5(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* We introduce two intermediate hybrids where the challenger uses the simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  for  $F_2$  (from Definition 3.14) to construct the punctured key and answer the evaluation queries:

- $\text{Hyb}_{4,1}$ : Same as  $\text{Hyb}_4$  except the challenger uses the simulator  $\mathcal{S}_1$  to construct the punctured key  $k_{v^*}^{(2)}$  and the simulator  $\mathcal{S}_2$  in place of  $F_2(k^{(2)}, \cdot)$  in the challenge phase. More precisely, in the setup phase, the challenger computes  $(k_{v^*}^{(2)}, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$ . During the query phase, the challenger answers all queries as in  $\text{Hyb}_4$ , except whenever it needs to evaluate  $F_2(k^{(2)}, \cdot)$  on  $v \in \mathcal{V}$ , the challenger instead computes  $(x, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(v, 1, \text{st}_{\mathcal{S}})$ , and uses  $x$  in place of  $F_2(k^{(2)}, v)$ .
- $\text{Hyb}_{4,2}$ : Same as  $\text{Hyb}_{4,1}$  except during the query phase, whenever it needs to evaluate  $F_2(k^{(2)}, \cdot)$  on  $v = F_1(k^{(1)}, x^*)$ , the challenger instead computes  $(x, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(v, 0, \text{st}_{\mathcal{S}})$ , and uses  $x$  in place of  $F_2(k^{(2)}, v)$ . All other queries to  $F_2(k^{(2)}, \cdot)$  are handled as in  $\text{Hyb}_{4,1}$ .

We now show that each pair of consecutive hybrids  $\text{Hyb}_4$ ,  $\text{Hyb}_{4,1}$ ,  $\text{Hyb}_{4,2}$ , and  $\text{Hyb}_5$  is computationally indistinguishable.

**Claim A.6.** *If  $F_2$  is a selectively-private puncturable PRF (Definition 3.14) and  $|\mathcal{X}| / |\mathcal{V}| = \text{negl}(\lambda)$ , then for all efficient adversaries  $\mathcal{A}$ , we have that  $|\Pr[\text{Hyb}_4(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{4,1}(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Suppose that there exists an adversary  $\mathcal{A}$  that can distinguish between  $\text{Hyb}_4$  and  $\text{Hyb}_{4,1}$ . We use  $\mathcal{A}$  to build an adversary  $\mathcal{B}$  that can distinguish between  $\text{Real}_{\mathcal{B}, F_2}(\lambda)$  and  $\text{Ideal}_{\mathcal{B}, \mathcal{S}, F_2}(\lambda)$  from Definition 3.14. Algorithm  $\mathcal{B}$  runs  $\mathcal{A}$  and simulates the experiment as follows:

- **Setup phase.** At the beginning of the game,  $\mathcal{A}$  commits to a challenge point  $x^*$ . Algorithm  $\mathcal{B}$  generates  $k^{(1)} \leftarrow F_1.\text{Setup}(1^\lambda)$ ,  $k_{x^*}^{(1)} \leftarrow F_1.\text{Puncture}(k^{(1)}, x^*)$  and chooses  $v^* \xleftarrow{\mathcal{R}} \mathcal{V}$ . It commits to  $v^*$  in the privacy game and receives a key  $k_{v^*}^{(2)}$ . Next,  $\mathcal{B}$  chooses  $z^* \xleftarrow{\mathcal{R}} \mathcal{X}$  and gives the key  $k_{x^*} = (k_{x^*}^{(1)}, k_{v^*}^{(2)})$  and the challenge value  $y^* = (x^*, z^*)$  to  $\mathcal{A}$ .
- **Query phase.** Algorithm  $\mathcal{B}$  simulates the evaluation queries exactly as in  $\text{Hyb}_4$  and  $\text{Hyb}_{4,1}$ , except whenever it needs to compute  $F_2(k^{(2)}, v)$  on some  $v \in \mathcal{V}$ , it queries its evaluation oracle for  $F_2$  on  $v$ . On an inversion query  $(y_1, y_2)$ , if  $y_1 \neq v^*$ , then algorithm  $\mathcal{B}$  proceeds exactly as in  $\text{Hyb}_4$  and  $\text{Hyb}_{4,1}$ , except it queries its evaluation oracle for  $F_2$  whenever it needs to compute  $F_2(k^{(2)}, v)$ . If  $y_1 = v^*$ , then algorithm  $\mathcal{B}$  replies with  $\perp$ .

At the end of the simulation, algorithm  $\mathcal{B}$  outputs whatever  $\mathcal{A}$  outputs. We consider the two distributions  $\text{Real}_{\mathcal{B}, F_2}$  and  $\text{Ideal}_{\mathcal{B}, \mathcal{S}, F_2}$  separately.

- If the key  $k_{v^*}^{(2)}$  and PRF evaluations are computed according to  $\text{Real}_{\mathcal{B}, F_2}$ , then the setup and evaluation queries are simulated exactly as in  $\text{Hyb}_4$ . By construction, the inversion queries  $(y_1, y_2)$  where  $y_1 \neq v^*$  are also simulated perfectly. When  $y_1 = v^*$ , the challenger in  $\text{Hyb}_4$  outputs something other than  $\perp$  only if there exists some  $x \in \mathcal{X}$  where  $F_1(k^{(1)}, x) = v^*$ . By the same argument as in the proof of Lemma A.4, with overwhelming probability over the choice of  $v^*$  (and taking a union bound over all  $x \in \mathcal{X}$ ), no such  $x$  exists, and so, the view  $\mathcal{B}$  simulates for  $\mathcal{A}$  is statistically indistinguishable from its view in  $\text{Hyb}_4$ .
- If the key  $k_{v^*}^{(2)}$  and PRF evaluations are computed according to  $\text{Ideal}_{\mathcal{B}, \mathcal{S}, F_2}$ , then we claim that  $\mathcal{B}$  correctly simulates  $\text{Hyb}_{4,1}$  for  $\mathcal{A}$ . There are two conditions to check. First, it must be the case that during the challenge phase,  $\mathcal{B}$  *never* needs to compute  $F_2(k^{(2)}, v^*)$ . If algorithm  $\mathcal{B}$  needs to query  $F_2$  on  $v^*$ , then in  $\text{Ideal}_{\mathcal{B}, \mathcal{S}, F_2}$ , the response is computed using  $\mathcal{S}_2(v^*, 0, \text{st}_{\mathcal{S}})$  whereas in  $\text{Hyb}_{4,1}$ , the response is computed using  $\mathcal{S}_2(v^*, 1, \text{st}_{\mathcal{S}})$ . We consider the two types of queries  $\mathcal{A}$  makes in the challenge phase.
  - **Evaluation queries.** On an evaluation query  $x \in \mathcal{X}$ ,  $\mathcal{B}$  first computes  $v \leftarrow F_1(k^{(1)}, x)$  and then queries  $F_2$  on  $v$ . By the same argument as in Lemma A.4, with overwhelming probability, there does not exist any  $x \in \mathcal{X}$  such that  $v^* = F_1(k^{(1)}, x)$ , so  $\mathcal{B}$  never needs to evaluate  $F_2$  on  $v^*$ .
  - **Inversion queries.** On an inversion query  $(y_1, y_2)$ , algorithm  $\mathcal{B}$  first computes  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$ . If  $y_1 \neq v^*$ , then  $\mathcal{B}$  does not need to evaluate  $F_2$  on  $v^*$ . When  $y_1 = v^*$ ,  $\mathcal{B}$  does not need to make any query to  $F_2$ .

Second, we show that  $\mathcal{B}$  correctly simulates the distribution in  $\text{Hyb}_{4,1}$ . By construction, the setup phase, the evaluation queries, and the inversion queries  $(y_1, y_2)$  where  $y_1 \neq v^*$  are simulated exactly as described in  $\text{Hyb}_{4,1}$ . For the inversion queries of the form  $(v^*, y_2)$ , the challenger's response in  $\text{Hyb}_{4,1}$  is always  $\perp$  with overwhelming probability (by the same argument as in Lemma A.4). Thus, the distribution  $\mathcal{B}$  simulates in this case is statistically indistinguishable from  $\text{Hyb}_{4,1}$ .

By puncturing security of  $F_2$ , hybrids  $\text{Hyb}_4$  and  $\text{Hyb}_{4,1}$  are computationally indistinguishable.  $\square$

**Claim A.7.** *If  $F_1$  is a selectively-secure puncturable PRF (Definition 3.7) and  $1/|\mathcal{V}| = \text{negl}(\lambda)$ , then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_{4,1}(\mathcal{A}) = 1] - \Pr[\text{Hyb}_{4,2}(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* By construction,  $\text{Hyb}_{4,1}$  and  $\text{Hyb}_{4,2}$  only differ on how the challenger handles the evaluation and inversion queries that require computing  $F_2(k^{(2)}, F_1(k^{(1)}, x^*))$ . There are two possibilities:

- The adversary makes an (admissible) evaluation query on  $x \neq x^*$  where  $F_1(k^{(1)}, x) = F_1(k^{(1)}, x^*)$ .
- The adversary makes an inversion query on  $(y_1, y_2)$  where  $y_1 = F_1(k^{(1)}, x^*)$ .

Suppose there exists an adversary  $\mathcal{A}$  that can distinguish between  $\text{Hyb}_{4,1}$  and  $\text{Hyb}_{4,2}$  with non-negligible probability  $\varepsilon$ . This means that with probability at least  $\varepsilon$ , adversary  $\mathcal{A}$  makes a query satisfying one of the above two conditions in  $\text{Hyb}_{4,1}$ . We use  $\mathcal{A}$  to build an algorithm  $\mathcal{B}$  that breaks puncturing security of  $F_1$ :

- **Setup phase.** At the beginning of the game, the adversary commits to a challenge point  $x^* \in \mathcal{X}$ . Algorithm  $\mathcal{B}$  commits to  $x^*$  in the puncturing security game for  $F_1$ , and receives a punctured key  $k_{x^*}^{(1)}$  and a challenge value  $\hat{v}^*$ . Algorithm  $\mathcal{B}$  then samples  $y^* \xleftarrow{\mathcal{R}} \mathcal{V} \times \mathcal{X}$ , computes  $(k_{v^*}^{(2)}, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$ , and gives the key  $k_{x^*} = (k_{x^*}^{(1)}, k_{v^*}^{(2)})$  as well as the challenge value  $y^*$  to  $\mathcal{A}$ .
- **Query phase.** Algorithm  $\mathcal{B}$  simulates the evaluation and inversion queries as follows:
  - **Evaluation queries.** On an (admissible) evaluation query  $x \neq x^*$ , algorithm  $\mathcal{B}$  queries the evaluation oracle for  $F_1$  on  $x$  to obtain a value  $v$ . It then computes  $(z, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(v, 1, \text{st}_{\mathcal{S}})$ , and returns  $(v, z \oplus x)$ , exactly as in  $\text{Hyb}_{4,1}$ . If  $v = \hat{v}^*$ , then  $\mathcal{B}$  sets the flag Good.
  - **Inversion queries.** On an inversion query  $(y_1, y_2) \neq (v^*, z^*)$ , algorithm  $\mathcal{B}$  first computes  $(x', \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(y_1, 1, \text{st}_{\mathcal{S}})$  and  $x \leftarrow x' \oplus y_2$ . If  $x \neq x^*$ , then  $\mathcal{B}$  queries its evaluation oracle for  $F_1$  and continues as in  $\text{Hyb}_{4,1}$ . Otherwise,  $\mathcal{B}$  responds with  $\perp$ . If  $y_1 = \hat{v}^*$ , then  $\mathcal{B}$  sets the flag Good.

At the end of the experiment, algorithm  $\mathcal{B}$  outputs 1 if the Good flag is set. Note that the “good” events precisely correspond to the types of queries  $\mathcal{A}$  makes with probability  $\varepsilon$  in  $\text{Hyb}_{4,1}$ . We now compute the probability that  $\mathcal{B}$  outputs 1 in the pseudorandom setting and the truly random setting:

- If  $\hat{v}^* = F_1(k^{(1)}, x^*)$ , we show that the Good flag is set with probability at least  $\varepsilon$ . There are two cases:
  - Suppose  $\mathcal{A}$  queries the inversion oracle on the point  $(y_1, y_2)$  where  $y_1 = F_1(k^{(1)}, x^*)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$ . In this case, the Good flag is always set.

- Suppose  $\mathcal{A}$  does not query the inversion oracle on the point  $(y_1, y_2)$  where  $y_1 = F_1(k^{(1)}, x^*)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$ . Then, by construction,  $\mathcal{B}$  perfectly simulates the view of  $\mathcal{A}$  in  $\text{Hyb}_{4,1}$ . In this case, with probability  $\varepsilon$ ,  $\mathcal{A}$  makes a query in the query phase that causes algorithm  $\mathcal{B}$  to set the **Good** flag.

We conclude that with probability at most  $\varepsilon$ ,  $\mathcal{B}$  outputs 1 when  $\hat{v}^* = F_1(k^{(1)}, x^*)$ .

- If  $\hat{v}^*$  is uniformly random over  $\mathcal{V}$ , then adversary  $\mathcal{B}$  outputs 1 with probability at most  $Q/|\mathcal{V}| = \text{negl}(\lambda)$ , where  $Q = \text{poly}(\lambda)$  is the number of evaluation or inversion queries the adversary makes. This follows from the fact that the adversary's view in the simulation is independent of  $\hat{v}^*$ . The probability that the **Good** flag is set is then equal to the probability that the randomly sampled  $\hat{v}^*$  matches one of the target values appearing in each of the queries.

The above analysis shows that  $\mathcal{B}$  breaks the puncturing security of  $F_1$  with probability  $\varepsilon - \text{negl}(\lambda)$ , where  $\varepsilon$  is the advantage  $\mathcal{A}$  has in distinguishing hybrids  $\text{Hyb}_{4,1}$  from  $\text{Hyb}_{4,2}$ . The claim follows.  $\square$

**Claim A.8.** *If  $F_2$  is a selectively-private puncturable PRF (Definition 3.14), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_{4,2}(\mathcal{A}) = 1] - \Pr[\text{Hyb}_5(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* This proof is very similar to the proof of Claim A.6. Suppose there exists an adversary  $\mathcal{A}$  that can distinguish between experiments  $\text{Hyb}_{4,2}$  and  $\text{Hyb}_5$ . We use  $\mathcal{A}$  to build an algorithm  $\mathcal{B}$  that can distinguish between  $\text{Real}_{\mathcal{B}, F_2}(\lambda)$  and  $\text{Ideal}_{\mathcal{B}, \mathcal{S}, F_2}(\lambda)$ . Algorithm  $\mathcal{B}$  behaves very similarly to the corresponding algorithm used in the proof of Claim A.6.

- **Setup phase.** Same behavior as algorithm  $\mathcal{B}$  in the proof of Claim A.6, except  $\mathcal{B}$  commits to the input  $F_1(k^{(1)}, x^*)$  in the privacy game (instead of  $v^*$ ).
- **Query phase.** Algorithm  $\mathcal{B}$  simulates the evaluation and inversion queries exactly as in  $\text{Hyb}_{4,2}$  and  $\text{Hyb}_5$ , except whenever it needs to compute  $F_2(k^{(2)}, v)$  on some  $v \in \mathcal{V}$ , it instead queries its evaluation oracle for  $F_2$  on  $v$ .

At the end of the simulation, algorithm  $\mathcal{B}$  outputs whatever  $\mathcal{A}$  outputs. Now, if the key  $k_{v^*}^{(2)}$  and PRF evaluations are computed according to  $\text{Real}_{\mathcal{B}, F_2}$ , then the setup and evaluation queries are simulated exactly as in  $\text{Hyb}_5$ . Conversely, if the key  $k_{v^*}^{(2)}$  and PRF evaluations are computed according to  $\text{Ideal}_{\mathcal{B}, \mathcal{S}, F_2}$ , then  $\mathcal{B}$  perfectly simulates  $\text{Hyb}_{4,2}$ . The claim follows.  $\square$

By Claims A.6 through A.8, we conclude that hybrids  $\text{Hyb}_4$  and  $\text{Hyb}_5$  are computationally indistinguishable.  $\square$

Combining Lemmas A.1 through A.5, we conclude that Construction 4.1 is a puncturable IPF.  $\square$

## B Analysis of Circuit-Constrained IPF

In this section, we give the formal correctness and security analysis of our circuit-constrained IPF (Construction 4.5) from Section 4.2.

## B.1 Proof of Theorem 4.6

Correctness of Construction 4.5 follows from correctness of the underlying constrained PRF and correctness of the public-key encryption scheme. Take any function  $f \in \mathcal{F}$  and let  $k = (k^{(1)}, k^{(2)}, \text{pk}, \text{sk}) \leftarrow \text{F.Setup}(1^\lambda)$ ,  $k_f = (k_f^{(1)}, k_f^{(2)}, \text{pk}) \leftarrow \text{F.Constrain}(k, f)$ . We now show the two correctness requirements separately:

- Take any  $x \in \mathcal{X}$  where  $f(x) = 1$ . Let  $(y_1, y_2) \leftarrow \text{F}(k, x)$ . Then,  $y_1 = \text{PKE.Encrypt}(\text{pk}, x; r_x)$  where  $r_x = \text{F}_1(k^{(1)}, x)$  and  $y_2 = \text{F}_2(k^{(2)}, y_1) \oplus x$ . By correctness of  $\text{F}_1$ , it follows that  $r_x = \text{F}_1.\text{Eval}(k_f^{(1)}, x)$ . Furthermore, by correctness of PKE,  $\text{Decrypt}(\text{sk}, y_1) = x$  and so,  $\text{F}_{\text{sk},f}(y_1) = f(\text{Decrypt}(\text{sk}, y_1)) = f(x) = 1$ . By correctness of  $\text{F}_2$ ,  $y_2 = \text{F}_2.\text{Eval}(k_f^{(2)}, y_1) \oplus x$ .
- Take any input  $(y_1, y_2) \in \mathcal{T} \times \mathcal{X}$  where there exists some  $x$  such that  $f(x) = 1$  and  $\text{F}(k, x) = (y_1, y_2)$ . This means that  $y_1 = \text{PKE.Encrypt}(\text{pk}, x; r_x)$  for  $r_x = \text{F}_1(k^{(1)}, x)$  and  $y_2 = \text{F}_2(k^{(2)}, y_1) \oplus x$ . By correctness of PKE,  $\text{F}_{\text{sk},f}(y_1) = 1$ . Then, by correctness of  $\text{F}_2$ ,  $\text{F}_2.\text{Eval}(k_f^{(2)}, y_1) \oplus y_2 = \text{F}_2(k^{(2)}, y_1) \oplus y_2 = x$ . By correctness of  $\text{F}_1$ ,  $r_x = \text{F}_1(k^{(1)}, x) = \text{F}_1.\text{Eval}(k_f^{(1)}, x)$ . In this case,  $\text{F.Eval}^{-1}(k_f, (y_1, y_2)) = x = \text{F}^{-1}(k, (y_1, y_2))$ .  $\square$

## B.2 Proof of Theorem 4.7

Our proof proceeds via a sequence of hybrid experiments between an adversary  $\mathcal{A}$  and a challenger. Recall that in the (single-key) selective-function security game (Remark 3.9), the adversary begins by committing to the constraint. The adversary can choose the challenge point adaptively. Similar to the proof of Theorem 4.3, we use a hybrid argument. Each hybrid experiment consists of several phases, which we describe below:

- **Setup phase.** In the selective-function security experiment, the adversary begins by committing to a function  $f^* \in \mathcal{F}$ . Then, the challenger samples an IPF key  $k \in \mathcal{K}$  as well as a constrained key  $k_{f^*}$ . The challenger gives  $k_{f^*}$  to the adversary.
- **Query phase.** The adversary  $\mathcal{A}$  is now allowed to make evaluation, inversion, and challenge queries. Without loss of generality, we assume the adversary makes at most one challenge query, denoted  $x^* \in \mathcal{X}$ . We sometimes need to distinguish between pre-challenge inversion queries and post-challenge inversion queries. By the admissibility requirement (Definition 3.12), the adversary's queries must satisfy the following conditions:
  - The challenge query  $x^* \in \mathcal{X}$  must satisfy  $f^*(x^*) = 0$ .
  - For all evaluation queries  $x \in \mathcal{X}$  the adversary makes,  $x \neq x^*$ .
  - For all inversion queries  $y \in \mathcal{Y}$  the adversary makes,  $y \neq y^*$ , where  $y^*$  is the challenger's response to the challenge query.
- **Output phase.** At the end of the experiment, the adversary outputs a bit  $b \in \{0, 1\}$ .

We now define our sequence of hybrid experiments. When defining a new hybrid, we only describe the phases that differ from the previous one.

- **Hyb<sub>0</sub>:** This is the constrained IPF security experiment  $\text{Expt}_{\mathcal{A}, \text{F}}^{(\text{IPF})}(\lambda, 0)$  from Definition 3.11. During the setup phase, after the adversary commits to a function  $f^* \in \mathcal{F}$ , the challenger



samples PRF keys  $k^{(1)} \leftarrow F_1.\text{Setup}(1^\lambda)$ ,  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$ , as well as a PKE public/secret key-pair  $(\text{pk}, \text{sk}) \leftarrow \text{PKE}.\text{Setup}(1^\lambda)$  and sets  $k = (k^{(1)}, k^{(2)}, \text{pk}, \text{sk})$ . Next, it computes  $k_{f^*}^{(1)} \leftarrow F_1.\text{Constrain}(k^{(1)}, f^*)$ ,  $k_{F^*}^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, F_{\text{sk}, f^*})$ , and gives  $k_{f^*} = (k_{f^*}^{(1)}, k_{F^*}^{(2)}, \text{pk})$  to the adversary. The challenger answers the evaluation and inversion queries by computing  $F(k, \cdot)$  and  $F^{-1}(k, \cdot)$ , respectively. On a challenge query  $x^* \in \mathcal{X}$ , the challenger computes  $r^* \leftarrow F_1(k^{(1)}, x^*)$ ,  $\text{ct}^* \leftarrow \text{PKE}.\text{Encrypt}(\text{pk}, x^*; r^*)$ , and returns  $y^* = (\text{ct}^*, F_2(k^{(2)}, \text{ct}^*) \oplus x^*)$ .

- **Hyb<sub>1</sub>**: Same as **Hyb<sub>0</sub>**, except whenever the adversary makes a pre-challenge inversion query  $(y_1, y_2) \in \mathcal{Y}$ , if  $F^{-1}(k, (y_1, y_2)) = x$  and  $f^*(x) = 0$ , and the challenger did not previously make an evaluation query for  $x$ , then the challenger replies with  $\perp$ . All other inversion queries are handled exactly as in **Hyb<sub>0</sub>**. In particular, we can rewrite the challenger's algorithm for the pre-challenge inversion queries as follows:
  - **Pre-challenge inversion queries.** On input  $(y_1, y_2) \in \mathcal{Y}$ , the challenger first computes  $x \leftarrow \text{PKE}.\text{Decrypt}(\text{sk}, y_1)$ . If  $f^*(x) = 1$  or  $\mathcal{A}$  has previously made an evaluation query on  $x$ , then the challenger checks to see if  $y_1 = \text{PKE}.\text{Encrypt}(\text{pk}, x; r_x)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x$ , where  $r_x \leftarrow F_1(k^{(1)}, x)$ . If so, the challenger replies with  $x$ , and otherwise, it replies with  $\perp$ . If  $\mathcal{A}$  has not previously made an evaluation query on  $x$  and  $f^*(x) = 0$ , then the challenger responds with  $\perp$ .
- **Hyb<sub>2</sub>**: Same as **Hyb<sub>1</sub>**, except when responding to the challenge query during the query phase, the challenger samples  $r^* \xleftarrow{\mathcal{R}} \mathcal{V}$ . Moreover, when responding to the (subsequent) evaluation and inversion queries, the challenger always uses  $r^*$  in place of the value  $F_1(k^{(1)}, x^*)$ .
- **Hyb<sub>3</sub>**: Same as **Hyb<sub>2</sub>**, except when responding to the evaluation and (post-challenge) inversion queries, the challenger always evaluates  $F_1(k^{(1)}, \cdot)$  instead of substituting the value  $r^*$  for  $F_1(k^{(1)}, x^*)$ .
- **Hyb<sub>4</sub>**: Same as **Hyb<sub>3</sub>**, except when responding to the challenge query, the challenger samples  $z^* \xleftarrow{\mathcal{R}} \mathcal{X}$  and returns  $(\text{ct}^*, z^*)$  to the adversary. When responding to the (subsequent) evaluation and inversion queries, the challenger uses  $z^* \oplus x^*$  in place of the value  $F_2(k^{(2)}, \text{ct}^*)$ .
- **Hyb<sub>5</sub>**: Same as **Hyb<sub>4</sub>**, except when responding to the evaluation and (post-challenge) inversion queries, the challenger always evaluates  $F_2(k^{(2)}, \cdot)$  instead of substituting the value  $z^* \oplus x^*$  for  $F_2(k^{(2)}, \text{ct}^*)$ .
- **Hyb<sub>6</sub>**: Same as **Hyb<sub>5</sub>**, except the challenger invokes the simulator  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  to generate the constrained key and answer the adversary's queries. The specific differences are outlined below:
  - **Setup phase.** Same as **Hyb<sub>5</sub>** except the challenge computes  $(k_{F^*}^{(2)}, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$  instead of setting  $k_{F^*}^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, F_{\text{sk}, f^*})$ .
  - **Query phase.** Same as **Hyb<sub>5</sub>** except whenever the challenger needs to evaluate  $F_2(k^{(2)}, \text{ct})$  on some  $\text{ct} \in \mathcal{T}$ , it instead computes  $(z, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(\text{ct}, F_{\text{sk}, f^*}(\text{ct}), \text{st}_{\mathcal{S}})$  and uses  $z$  in place of  $F_2(k^{(2)}, \text{ct})$ .
- **Hyb<sub>7</sub>**: Same as **Hyb<sub>6</sub>**, except the challenger answers the challenge query by choosing a random value  $\hat{x}^* \xleftarrow{\mathcal{R}} \mathcal{X}$  and returning  $(\text{PKE}.\text{Encrypt}(\text{pk}, \hat{x}^*; r^*), z^*)$ .

- **Hyb<sub>8</sub>**: Same as Hyb<sub>7</sub>, except the challenger sets  $k_{F^*}^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, F_{\text{sk}, f^*})$  in the setup phase, and uses the real evaluations  $F_2(k^{(2)}, \cdot)$  rather than the simulated ones in the query phase.
- **Hyb<sub>9</sub>**: Same as Hyb<sub>8</sub>, except the pre-challenge inversion queries are now handled as in the real scheme. This is the constrained IPF security experiment  $\text{Expt}_{\mathcal{A}, F}^{(\text{IPF})}(\lambda, 1)$ .

As in the proof of Theorem 4.3, we write  $\text{Hyb}_i(\mathcal{A})$  to denote the output of experiment **Hyb<sub>i</sub>** with an adversary  $\mathcal{A}$ . We now show that the output of each consecutive pair of hybrids is computationally indistinguishable. In the following, we make the implicit assumptions that the adversary  $\mathcal{A}$  is admissible for the constrained IPF security game and that PKE is (perfectly) correct.

**Lemma B.1.** *If PKE is a CCA-secure public-key encryption scheme (Definition 2.1), and  $F_1$  is a single-key, selective-function-secure constrained PRF, then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_1(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* By construction, Hyb<sub>0</sub> and Hyb<sub>1</sub> differ only in how the challenger responds to the pre-challenge inversion queries. Suppose there is an efficient algorithm  $\mathcal{A}$  that can distinguish Hyb<sub>0</sub> from Hyb<sub>1</sub> with non-negligible advantage  $\varepsilon$ . Then, with probability  $\varepsilon$ , algorithm  $\mathcal{A}$  will make a pre-challenge inversion query  $(y_1, y_2)$  such that the challenger's response is  $\perp$  in Hyb<sub>1</sub>, but not in Hyb<sub>0</sub>. We use  $\mathcal{A}$  to construct an adversary  $\mathcal{B}$  that breaks the security of  $F_1$ . Algorithm  $\mathcal{B}$  simulates an execution of Hyb<sub>0</sub> and Hyb<sub>1</sub> as follows:

- **Setup phase.** At the beginning of the experiment, adversary  $\mathcal{A}$  commits to  $f^* \in \mathcal{F}$ . Algorithm  $\mathcal{B}$  sends  $f^*$  to the challenger for  $F_1$  to obtain a constrained key  $k_{f^*}^{(1)}$ . It runs  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Setup}(1^\lambda)$ ,  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$  and constructs  $k_{F^*}^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, F_{\text{sk}, f^*})$ . It gives the constrained key  $k_{f^*} = (k_{f^*}^{(1)}, k_{F^*}^{(2)}, \text{pk})$  to  $\mathcal{A}$ . In addition, algorithm  $\mathcal{B}$  initializes an empty set  $S = \emptyset$  and an empty table of mappings  $T = \emptyset$ .
- **Query phase.** Algorithm  $\mathcal{B}$  responds to the queries as follows:
  - **Evaluation queries.** On input a point  $x \in \mathcal{X}$ , algorithm  $\mathcal{B}$  queries  $F_1$  on  $x$  to obtain a point  $r_x$ . It then computes  $\text{ct} \leftarrow \text{PKE.Encrypt}(\text{pk}, x; r_x)$  and  $z \leftarrow F_2(k^{(2)}, \text{ct})$  and replies to  $\mathcal{A}$  with  $(\text{ct}, z \oplus x)$ . In addition,  $\mathcal{B}$  adds the mapping  $\text{ct} \mapsto (z, x)$  to  $T$ .
  - **Pre-challenge inversion queries.** On input an inversion query  $(y_1, y_2)$ , algorithm  $\mathcal{B}$  first checks to see if there is a mapping of the form  $y_1 \mapsto (z, x)$  in  $T$  for some  $z, x \in \mathcal{X}$ . If such a mapping exists, and moreover,  $y_2 = z \oplus x$ , then  $\mathcal{B}$  replies with  $x$ . If a mapping exists and  $y_2 \neq z \oplus x$ , then  $\mathcal{B}$  replies with  $\perp$ . Otherwise, if no mapping exists,  $\mathcal{B}$  computes  $x \leftarrow \text{PKE.Decrypt}(\text{sk}, y_1)$ . If  $f^*(x) = 0$ , then  $\mathcal{B}$  responds with  $\perp$  and adds  $(y_1, y_2)$  to  $S$  (if  $x \neq \perp$ ). Otherwise, if  $f^*(x) = 1$ ,  $\mathcal{B}$  queries  $F_1$  on  $x$  to obtain a point  $r_x$ . It then computes  $\text{ct} \leftarrow \text{PKE.Encrypt}(\text{pk}, x; r_x)$  and  $z \leftarrow F_2(k^{(2)}, \text{ct}) \oplus x$ . Finally, algorithm  $\mathcal{B}$  replies with  $x$  if  $(\text{ct}, z \oplus x) = (y_1, y_2)$  and  $\perp$  otherwise.
  - **Challenge query.** When  $\mathcal{A}$  issues a challenge query,  $\mathcal{B}$  halts the simulation.

At the end of the simulation (or if  $\mathcal{A}$  aborts prematurely), algorithm  $\mathcal{B}$  does the following:

1. If the set  $S$  is empty, then  $\mathcal{B}$  outputs 0. Otherwise, it samples a random element  $(y_1^*, y_2^*) \xleftarrow{\mathcal{R}} S$ . It computes  $x^* \leftarrow \text{PKE.Decrypt}(\text{sk}, y_1^*)$  and submits  $x^*$  as its challenge query to  $F_1$ .

2. The challenger replies with a value  $r^* \in \mathcal{V}$ . Algorithm  $\mathcal{B}$  outputs 1 if  $y_1^* = \text{PKE.Encrypt}(\text{pk}, x^*; r^*)$  and 0 otherwise.

We first show that  $\mathcal{B}$  is admissible. By construction, the challenge point  $x^*$  satisfies  $f^*(x^*) = 0$ . Next, algorithm  $\mathcal{B}$  does not query  $F_1$  on  $x^*$  when answering the evaluation queries (if it did, then  $x^*$  would be contained in  $T$  and not added to  $S$ ). In addition, if  $\mathcal{B}$  added  $x^*$  to  $S$ , then it never queries  $F_1$  on  $x^*$ . We conclude that  $\mathcal{B}$  is admissible.

Moreover,  $\mathcal{B}$  perfectly simulates the setup and the pre-challenge queries according to the specification in  $\text{Hyb}_1$ . By assumption, with probability  $\varepsilon$ ,  $\mathcal{A}$  will issue a pre-challenge inversion query  $(y_1^*, y_2^*)$  such that the challenger's response is  $\perp$  in  $\text{Hyb}_1$ , but not in  $\text{Hyb}_0$ . In particular, this implies the following:

- First,  $y_1^*$  must not have been part of the output of *any* previous evaluation query. Otherwise, the behavior in  $\text{Hyb}_0$  and  $\text{Hyb}_1$  is identical.
- Since the challenger's response in  $\text{Hyb}_0$  is not  $\perp$ , it must be the case that  $y_1^* = \text{PKE.Encrypt}(\text{pk}, x; r_x)$  for some  $x \in \mathcal{X}$ ,  $r_x = F_1(k^{(1)}, x)$ , and  $y_2^* = F_2(k^{(2)}, y_1^*) \oplus x$ . From the first requirement, we have that the adversary did not previously submit an evaluation query for  $x$ . Since the challenger's response in  $\text{Hyb}_1$  is  $\perp$ , it must additionally be the case that  $f^*(x) = 0$ .

With probability  $\varepsilon$ ,  $\mathcal{A}$  will make a query  $(y_1^*, y_2^*)$  satisfying the above criterion when interacting with  $\mathcal{B}$ . On such a query  $(y_1^*, y_2^*)$ ,  $\mathcal{B}$  replies with  $\perp$  and adds  $(y_1^*, y_2^*)$  to the set  $S$ . Thus, with probability  $\varepsilon/Q$  (where  $Q = \text{poly}(\lambda)$  is the total number of queries  $\mathcal{A}$  makes prior to the challenge phase),  $\mathcal{B}$  will choose  $(y_1^*, y_2^*)$  to construct its challenge at the end of the simulation. We consider the probability that  $\mathcal{B}$  outputs 1 depending on whether  $r^*$  is pseudorandom or uniformly random. Here,  $k^{(1)}$  is the PRF key sampled by the constrained security challenger for  $F_1$ .

- Suppose  $r^* = F_1(k^{(1)}, x^*)$ . Then, with probability at least  $\varepsilon/Q$ ,  $y_1^* = \text{PKE.Encrypt}(\text{pk}, x^*; r^*)$ , in which case,  $\mathcal{B}$  outputs 1. Thus, in this case,  $\mathcal{B}$  outputs 1 with probability at least  $\varepsilon/Q$ .
- Suppose  $r^*$  is uniform over  $\mathcal{V}$  (and independent of all other quantities). Then, we can invoke Lemma 2.2 to argue that over the randomness of  $r^*$ ,  $\Pr[\text{PKE.Encrypt}(\text{pk}, x^*; r^*) = y_1^*] = \text{negl}(\lambda)$ . In this case,  $\mathcal{B}$  outputs 1 with negligible probability.

Since  $Q = \text{poly}(\lambda)$  and  $\varepsilon$  is non-negligible, we have that  $\mathcal{B}$  breaks puncturing security of  $F_1$  with probability at least  $\varepsilon/Q - \text{negl}(\lambda)$ . The lemma follows.  $\square$

**Lemma B.2.** *If  $F_1$  is a single-key, selective-function-secure constrained PRF (Definition 3.6), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_1(\mathcal{A}) = 1] - \Pr[\text{Hyb}_2(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Let  $\mathcal{A}$  be a distinguisher for  $\text{Hyb}_1$  and  $\text{Hyb}_2$ . We construct an adversary  $\mathcal{B}$  to break constrained security of  $F_1$ . Algorithm  $\mathcal{B}$  simulates the experiments as follows:

- **Setup phase.** At the beginning of the experiment, adversary  $\mathcal{A}$  commits to a function  $f^* \in \mathcal{F}$ . Algorithm  $\mathcal{B}$  sends  $f^*$  to the constrained PRF challenger for  $F_1$  and receives a constrained PRF key  $k_{f^*}^{(1)}$ . Next, it samples  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Setup}(1^\lambda)$ ,  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$ , and  $k_{F^*}^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, F_{\text{sk}, f^*})$  as in  $\text{Hyb}_1$  and  $\text{Hyb}_2$ . It gives the constrained key  $k_{f^*} = (k_{f^*}^{(1)}, k_{F^*}^{(2)}, \text{pk})$  to  $\mathcal{A}$ .

- **Query phase.** Algorithm  $\mathcal{B}$  simulates the pre-challenge evaluation and inversion queries exactly as described in  $\text{Hyb}_1$  and  $\text{Hyb}_2$ . Whenever it needs to compute  $F_1(k^{(1)}, x)$ , it instead queries its evaluation oracle for  $F_1$ . When  $\mathcal{A}$  makes a challenge query for  $x^* \in \mathcal{X}$ , algorithm  $\mathcal{B}$  submits  $x^*$  as its challenge to the constrained security challenger for  $F_1$ , and receives back a value  $r^*$ . It then computes  $\text{ct}^* \leftarrow \text{PKE.Encrypt}(\text{pk}, x^*; r^*)$  and  $z^* \leftarrow F_2(k^{(2)}, \text{ct}^*)$ . It gives  $(\text{ct}^*, F_2(k^{(2)}, \text{ct}^*) \oplus x^*)$  to  $\mathcal{A}$ . The post-challenge queries are handled exactly as in  $\text{Hyb}_1$  and  $\text{Hyb}_2$ , except  $\mathcal{B}$  queries its evaluation oracle for  $F_1$  whenever it needs to compute  $F_1(k^{(1)}, x)$  for  $x \neq x^*$ , and  $\mathcal{B}$  uses  $r^*$  in place of the value of  $F_1(k^{(1)}, x^*)$ .
- **Output phase.** After  $\mathcal{A}$  outputs a bit  $b \in \{0, 1\}$ , algorithm  $\mathcal{B}$  outputs the same bit  $b$ .

First, we argue that  $\mathcal{B}$  is admissible for the constrained security game. Since  $\mathcal{A}$  is admissible, it follows that  $f^*(x^*) = 0$ , and moreover,  $\mathcal{A}$  never makes an evaluation query on  $x^*$ . Correspondingly, algorithm  $\mathcal{B}$  never has to query  $F_1$  on  $x^*$  when answering an evaluation query. For the pre-challenge inversion queries,  $\mathcal{B}$  only needs to query  $F_1$  on points  $x$  where  $f^*(x) = 1$  (in which case,  $x \neq x^*$ ) or on points  $x$  that previously appeared in an evaluation query (in which case, again,  $x \neq x^*$ ). For the post-challenge inversion queries, algorithm  $\mathcal{B}$  always substitutes  $r^*$  for the value  $F_1(k^{(1)}, x^*)$ , so it never needs to query  $F_1$  on  $x^*$ . We conclude that  $\mathcal{B}$  is admissible.

To complete the proof, we see that if  $r^* = F_1(k^{(1)}, x^*)$ , then  $\mathcal{B}$  perfectly simulates  $\text{Hyb}_1$ . If  $r^*$  is uniformly random over  $\mathcal{V}$ , then  $\mathcal{B}$  perfectly simulates  $\text{Hyb}_2$  for  $\mathcal{A}$ . Thus,  $\mathcal{B}$  breaks the constrained security of  $F_1$  with the same advantage as  $\mathcal{A}$ . The lemma follows.  $\square$

**Lemma B.3.** *If PKE is a CCA-secure public-key encryption scheme (Definition 2.1), and  $F_1$  is a single-key, selective-function-secure constrained PRF (Definition 3.7), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_2(\mathcal{A}) = 1] - \Pr[\text{Hyb}_3(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* The only difference between  $\text{Hyb}_2$  and  $\text{Hyb}_3$  is that the challenger uses the value  $r^*$  in place of the value  $F_1(k^{(1)}, x^*)$  to answer the *post-challenge* evaluation and inversion queries in  $\text{Hyb}_2$  while the challenger uses the real value  $F_1(k^{(1)}, x^*)$  in  $\text{Hyb}_3$ . To show the lemma, we consider each type of query separately, and argue that the view of the adversary is computationally indistinguishable between hybrids  $\text{Hyb}_2$  and  $\text{Hyb}_3$ .

- **Evaluation queries.** On an evaluation query  $x$ , the challenger in  $\text{Hyb}_2$  and  $\text{Hyb}_3$  only needs to evaluate  $F_1(k^{(1)}, \cdot)$  on  $x$ . Since  $\text{Hyb}_2$  and  $\text{Hyb}_3$  only differ in the value the challenger uses for  $F_1(k^{(1)}, x^*)$ , the outputs of the evaluation oracle in  $\text{Hyb}_2$  and  $\text{Hyb}_3$  are identically distributed on all inputs  $x \neq x^*$ . By admissibility, the adversary can only query the evaluation oracle on points  $x \neq x^*$ , and so, the responses of the evaluation oracle in  $\text{Hyb}_2$  and  $\text{Hyb}_3$  are distributed identically.
- **Post-challenge inversion queries.** On a (post-challenge) inversion query  $(y_1, y_2)$ , the challenger computes  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$ ,  $r_x \leftarrow F_1(k^{(1)}, x)$ , and  $\text{ct} \leftarrow \text{PKE.Encrypt}(\text{pk}, x; r_x)$ . Then, it checks whether  $y_1 = \text{ct}$ . If  $x \neq x^*$ , then the output of the inversion oracle in  $\text{Hyb}_2$  and  $\text{Hyb}_3$  is computed in the same way. It suffices to just consider the case where  $x = x^*$ , or equivalently, when  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ . Let  $\text{ct}^* = \text{PKE.Encrypt}(\text{pk}, x^*; r^*)$  be the ciphertext component in the challenge that is constructed using a uniformly random  $r^*$ .

- In  $\text{Hyb}_2$ , the challenger uses the value  $r^*$  for the value  $F_1(k^{(1)}, x^*)$ . For any query  $(y_1, y_2)$  where  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ , the challenger in  $\text{Hyb}_2$  always responds with  $\perp$  unless

$y_1 = \text{ct}^*$ . But if  $y_1 = \text{ct}^*$ , then it must be the case that  $y_2 = F_2(k^{(2)}, \text{ct}^*) \oplus x^* = z^*$ . This means that  $(y_1, y_2) = (\text{ct}^*, z^*)$ , which is the challenge query. Therefore, the challenger in  $\text{Hyb}_2$  responds with  $\perp$  on all admissible inversion queries satisfying  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ .

- In  $\text{Hyb}_3$ , on a query  $(y_1, y_2)$  where  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ , the challenger in  $\text{Hyb}_2$  responds with  $\perp$  unless  $y_1 = \text{PKE.Encrypt}(\text{pk}, x^*; r_{x^*})$  where  $r_{x^*} = F_1(k^{(1)}, x^*)$ . In this case,  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$ .

We conclude that the response of the challenger on an inversion query is identically distributed in  $\text{Hyb}_2$  and  $\text{Hyb}_3$  unless the adversary makes a (post-challenge) inversion query on the tuple  $(y_1, y_2)$  where  $y_1 = \text{PKE.Encrypt}(\text{pk}, x^*; r_{x^*})$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$ .

Suppose now that an adversary  $\mathcal{A}$  is able to distinguish between  $\text{Hyb}_2$  and  $\text{Hyb}_3$  with some non-negligible probability  $\varepsilon$ . By the above analysis, the adversary's view in  $\text{Hyb}_2$  and  $\text{Hyb}_3$  is identically distributed unless the adversary makes a (post-challenge) inversion query on an input  $(y_1, y_2)$  where  $y_1 = \text{PKE.Encrypt}(\text{pk}, x^*; r_{x^*})$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$ . Since  $\mathcal{A}$  distinguishes  $\text{Hyb}_2$  from  $\text{Hyb}_3$  with advantage  $\varepsilon$ , it will make an inversion query on a pair  $(y_1, y_2)$  satisfying the above property with probability at least  $\varepsilon$ . We use  $\mathcal{A}$  to construct an adversary  $\mathcal{B}$  that breaks the constrained PRF security of  $F_1$ . Algorithm  $\mathcal{B}$  simulates an execution of  $\text{Hyb}_2$  as follows:

- **Setup phase.** At the beginning of the experiment, the adversary commits to a constraint  $f^* \in \mathcal{F}$ . Algorithm  $\mathcal{B}$  sends  $f^*$  to the constrained PRF challenger of  $F_1$  and receives a constrained key  $k_{f^*}^{(1)}$ . It generates the key  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$ , a public/secret key-pair  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Setup}(1^\lambda)$ , and constructs the constrained key  $k_{F^*}^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, F_{\text{sk}, f^*})$ . It sends  $k_{f^*} = (k_{f^*}^{(1)}, k_{F^*}^{(2)}, \text{pk})$  to  $\mathcal{A}$ .
- **Query phase.** Algorithm  $\mathcal{B}$  simulates the evaluation, pre-challenge inversion, and challenge queries exactly as described in  $\text{Hyb}_2$  and  $\text{Hyb}_3$ , with the exception that it queries its evaluation oracle for  $F_1$  whenever it needs to evaluate  $F_1(k^{(1)}, \cdot)$ . For post-challenge inversion queries  $(y_1, y_2)$  where  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ , algorithm  $\mathcal{B}$  always responds with  $\perp$ . The other post-challenge inversion queries are handled as in the real scheme, except  $\mathcal{B}$  queries  $F_1$  on  $x$  whenever it needs to evaluate  $F_1(k^{(1)}, x)$ .
- **Output phase.** At the end of the experiment (or if  $\mathcal{A}$  aborts),  $\mathcal{B}$  sends the challenge point  $x^*$  as its challenge query to the constrained PRF challenger for  $F_1$ . It receives the challenge  $\hat{r}^*$  and computes  $\hat{\text{ct}}^* \leftarrow \text{PKE.Encrypt}(\text{pk}, x^*; \hat{r}^*)$ . Now,  $\mathcal{B}$  checks whether  $\mathcal{A}$  made an inversion query on the tuple  $(\hat{\text{ct}}^*, F_2(k^{(2)}, \hat{\text{ct}}^*) \oplus x^*)$ . If  $\mathcal{A}$  made such a query, then  $\mathcal{B}$  output 1. Otherwise, it outputs 0.

We first argue that if  $\mathcal{A}$  is admissible, then  $\mathcal{B}$  is admissible for the constrained PRF security game. To see this, we first note that for each of  $\mathcal{A}$ 's evaluation queries  $x \in \mathcal{X}$ , we require (by admissibility) that  $x \neq x^*$  and therefore,  $\mathcal{B}$  never needs to query its evaluation oracle on  $x^*$ . For the pre-challenge inversion queries, adversary  $\mathcal{B}$  only queries  $F_1$  on points  $x \in \mathcal{X}$  where  $f^*(x) = 1$  or that appeared in a previous evaluation query. In particular, this means that  $\mathcal{B}$  never needs to query on  $x^*$ . For the post-challenge inversion queries, algorithm  $\mathcal{B}$  only needs to evaluate  $F_1$  on points  $x \neq x^*$ . Thus,  $\mathcal{B}$  is admissible.

By construction, algorithm  $\mathcal{B}$  perfectly simulates  $\text{Hyb}_2$  for  $\mathcal{A}$ . This means that with probability at least  $\varepsilon$ , adversary  $\mathcal{A}$  will submit an inversion query on the tuple  $(y_1, y_2)$  where  $y_1 =$

$\text{PKE.Encrypt}(\text{pk}, x^*; r_{x^*})$ ,  $r_{x^*} = F_1(k^{(1)}, x^*)$ , and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$ . We now consider the probability that  $\mathcal{B}$  outputs 1.

- Suppose  $\hat{r}^* = F_1(k^{(1)}, x^*)$ . By assumption, with probability at least  $\varepsilon$ , adversary  $\mathcal{A}$  will query the inversion oracle on the tuple  $(y_1, y_2)$  where  $y_1 = \text{PKE.Encrypt}(\text{pk}, x^*; r_{x^*}) = \text{PKE.Encrypt}(\text{pk}, x^*; \hat{r}^*)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x^*$  at some point in the execution. In this case, algorithm  $\mathcal{B}$  outputs 1.
- Suppose  $\hat{r}^*$  is uniformly random in  $\mathcal{V}$ . By Lemma 2.2, over the randomness of  $\hat{r}^*$ , for any fixed string  $\text{ct} \in \mathcal{T}$ , we have that  $\Pr[\text{PKE.Encrypt}(\text{pk}, x^*; \hat{r}^*) = \text{ct}] = \text{negl}(\lambda)$ . If the adversary makes a total of  $Q = \text{poly}(\lambda)$  inversion queries, then by a union bound, the probability that  $\hat{\text{ct}}^* = y_1$  for some  $y_1$  appearing in one of  $\mathcal{A}$ 's inversion queries is bounded by  $Q \cdot \text{negl}(\lambda) = \text{negl}(\lambda)$ . Correspondingly, in this case, algorithm  $\mathcal{B}$  outputs 1 with  $\text{negl}(\lambda)$  probability.

We conclude that if  $\mathcal{A}$  is able to distinguish between hybrids  $\text{Hyb}_2$  and  $\text{Hyb}_3$  with non-negligible advantage  $\varepsilon$ , then  $\mathcal{B}$  is able to break puncturing security of  $F_1$  with probability  $\varepsilon - \text{negl}(\lambda)$ .  $\square$

**Lemma B.4.** *If  $F_2$  is a single-key, selective-function-secure constrained PRF (Definition 3.7), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_3(\mathcal{A}) = 1] - \Pr[\text{Hyb}_4(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Suppose that there exists an adversary  $\mathcal{A}$  that can distinguish between hybrids  $\text{Hyb}_3$  and  $\text{Hyb}_4$ . We construct an adversary to break the constrained PRF security of  $F_2$ . Algorithm  $\mathcal{B}$  simulates the experiment as follows:

- **Setup phase.** At the beginning of the experiment,  $\mathcal{A}$  commits to a function  $f^*$ . Algorithm  $\mathcal{B}$  generates the key  $k^{(1)} \leftarrow F_1.\text{Setup}(1^\lambda)$ , the constrained key  $k_{f^*}^{(1)} \leftarrow F_1.\text{Constrain}(k^{(1)}, f^*)$ , and the PKE key  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Setup}(1^\lambda)$ . It submits  $F_{\text{sk}, f^*}$  to the constrained PRF challenger for  $F_2$  and receives a constrained key  $k_{F^*}^{(2)}$ . Algorithm  $\mathcal{B}$  gives  $k_{f^*} = (k_{f^*}^{(1)}, k_{F^*}^{(2)})$  to  $\mathcal{A}$ .
- **Query phase.** Algorithm  $\mathcal{B}$  simulates the adversary's evaluation and inversion queries exactly as in  $\text{Hyb}_3$  and  $\text{Hyb}_4$ , except whenever it needs to compute  $F_2(k^{(2)}, \cdot)$ , it queries its evaluation oracle for  $F_2$ . For the challenge query  $x^*$ ,  $\mathcal{B}$  samples a random element  $r^* \xleftarrow{\mathcal{R}} \mathcal{V}$ , computes  $\text{ct}^* \leftarrow \text{PKE.Encrypt}(\text{pk}, x^*; r^*)$ , and submits  $\text{ct}^*$  as the challenge query to the constrained PRF challenger of  $F_2$  to receive  $z^*$ . It returns  $(\text{ct}^*, z^* \oplus x^*)$  to the adversary. Subsequently, whenever it needs to compute  $F_2(k^{(2)}, \text{ct}^*)$ , it uses the value  $z^*$ .
- **Output phase.** Algorithm  $\mathcal{B}$  outputs whatever  $\mathcal{A}$  outputs.

First, we argue that if  $\mathcal{A}$  is admissible, then  $\mathcal{B}$  is admissible. Certainly,  $F_{\text{sk}, f^*}(\text{ct}^*) = 0$  since  $f^*(x^*) = 0$  and  $\text{ct}^*$  is an encryption of  $x^*$ . Next, when answering an evaluation query for a point  $x \in \mathcal{X}$ , algorithm  $\mathcal{B}$  needs to query  $F_2$  on a ciphertext  $\text{ct}$  encrypting  $x$ . Since  $x \neq x^*$  and  $\text{ct}^*$  is an encryption of  $x^*$ , correctness of PKE implies that  $\text{ct} \neq \text{ct}^*$ . For the pre-challenge inversion queries,  $\mathcal{B}$  only queries  $F_2$  on ciphertexts that encrypt a value  $x$  where either  $f^*(x) = 1$  or if  $\mathcal{A}$  has previously made an evaluation query on  $x$ . Both of these conditions imply that  $x \neq x^*$  and, correspondingly  $\text{ct} \neq \text{ct}^*$ . In the post-challenge phase,  $\mathcal{B}$  substitutes  $z^*$  in place of  $F_2(k^{(2)}, \text{ct}^*)$ , so we conclude that  $\mathcal{B}$  never needs to make an evaluation query for  $\text{ct}^*$ .

To conclude the proof, we note that if  $z^* = F_2(k^{(2)}, \text{ct}^*)$ , then  $\mathcal{B}$  perfectly simulates  $\text{Hyb}_3$  for  $\mathcal{A}$ . If  $z^*$  is uniformly random over  $\mathcal{X}$ ,  $\mathcal{B}$  perfectly simulates  $\text{Hyb}_4$  for  $\mathcal{A}$  (specifically, if  $z^*$  is uniform over  $\mathcal{X}$ , so is  $z^* \oplus x^*$ ). We conclude that algorithm  $\mathcal{B}$  breaks the constrained PRF security of  $F_2$  with the same advantage  $\mathcal{A}$  has in distinguishing  $\text{Hyb}_3$  and  $\text{Hyb}_4$ .  $\square$

**Lemma B.5.** *If PKE is a CCA-secure public-key encryption scheme (Definition 2.1), then for all adversaries  $\mathcal{A}$ , we have that  $|\Pr[\text{Hyb}_4(\mathcal{A}) = 1] - \Pr[\text{Hyb}_5(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* The only difference between  $\text{Hyb}_4$  and  $\text{Hyb}_5$  is that to answer the (post-challenge) evaluation and inversion queries, the challenger uses the value  $z^* \oplus x^*$  (where  $z^*$  is uniformly random over  $\mathcal{X}$ ) in place of  $F_2(k^{(2)}, \text{ct}^*)$  in  $\text{Hyb}_4$ , while the challenger uses the real value  $F_2(k^{(2)}, \text{ct}^*)$  in  $\text{Hyb}_5$ . Importantly, we will use the fact that in hybrids  $\text{Hyb}_4$  and  $\text{Hyb}_5$ , the ciphertext  $\text{ct}^*$  is an encryption of the challenge point  $x^*$  under PKE with uniformly-sampled randomness. To show the lemma, we consider each type of query and argue that the views of the adversary in  $\text{Hyb}_4$  and  $\text{Hyb}_5$  are statistically indistinguishable.

- **Evaluation queries.** On an evaluation query  $x$ , the challenger in  $\text{Hyb}_4$  and  $\text{Hyb}_5$  first computes  $r_x \leftarrow F_1(k^{(1)}, x)$ ,  $\text{ct} \leftarrow \text{PKE.Encrypt}(\text{pk}, x; r_x)$ , and then evaluates  $F_2(k^{(2)}, \text{ct})$ . Therefore, the response to the evaluation query in  $\text{Hyb}_4$  and  $\text{Hyb}_5$  differ only on points  $x \in \mathcal{X}$  where  $\text{PKE.Encrypt}(\text{pk}, x; r_x) = \text{ct}^*$ . By construction,  $\text{ct}^*$  is an encryption of  $x^*$  under  $\text{pk}$ , so by correctness of PKE, for all  $x \neq x^*$ ,  $\text{PKE.Encrypt}(\text{pk}, x; r_x) \neq \text{ct}^*$ . This means that the outputs of the evaluation oracle (on all admissible queries) in  $\text{Hyb}_4$  is the same as that in  $\text{Hyb}_5$ .
- **Inversion queries.** On an inversion query  $(y_1, y_2)$ , the challenger in  $\text{Hyb}_4$  and  $\text{Hyb}_5$  first computes  $x \leftarrow F_2(k^{(2)}, y_1) \oplus y_2$  and then checks whether  $y_1 = \text{PKE.Encrypt}(\text{pk}, x; r_x)$ , where  $r_x = F_1(k^{(1)}, x)$ . On all queries where  $y_1 \neq \text{ct}^*$ , the challenger's behavior in  $\text{Hyb}_4$  and  $\text{Hyb}_5$  is identical. On queries  $(\text{ct}^*, y_2)$ , the only instance where the adversary does not output  $\perp$  is if  $\text{PKE.Encrypt}(\text{pk}, x; r_x) = \text{ct}^*$ . By correctness of PKE, if  $x \neq x^*$ , then  $\text{PKE.Encrypt}(\text{pk}, x; r_x) \neq \text{ct}^*$ . Finally, by Lemma 2.2, over the randomness used to construct  $\text{ct}^*$  (more specifically, over the randomness used to sample  $r^*$ ),  $\Pr[\text{ct}^* = \text{PKE.Encrypt}(\text{pk}, x^*; r_{x^*})] = \text{negl}(\lambda)$ , where  $r_{x^*} = F_1(k^{(1)}, x^*)$ . This means that with overwhelming probability, the response to all of the inversion queries of the form  $(\text{ct}^*, y_2)$  in  $\text{Hyb}_4$  and  $\text{Hyb}_5$  will be  $\perp$ . The claim follows.  $\square$

**Lemma B.6.** *If  $F_2$  is a single-key, selective-function-private constrained PRF (Definition 3.14), then for all efficient adversaries  $\mathcal{A}$ , we have that  $|\Pr[\text{Hyb}_5(\mathcal{A}) = 1] - \Pr[\text{Hyb}_6(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Suppose there exists an adversary  $\mathcal{A}$  that can distinguish between hybrids  $\text{Hyb}_5$  and  $\text{Hyb}_6$ . We construct an adversary  $\mathcal{B}$  to distinguish the two experiments  $\text{Real}_{\mathcal{B}, F_2}$  and  $\text{Ideal}_{\mathcal{B}, \mathcal{S}, F_2}$  in the single-key, selective-function-privacy game for  $F_2$  (Definition 3.14). Algorithm  $\mathcal{B}$  simulates the experiments as follows:

- **Setup phase.** At the beginning of the experiment, adversary  $\mathcal{A}$  commits to a function  $f^* \in \mathcal{F}$ . Algorithm  $\mathcal{B}$  samples the key  $k^{(1)} \leftarrow F_1.\text{Setup}(1^\lambda)$ , the constrained key  $k_{f^*}^{(1)} \leftarrow F_1.\text{Constrain}(k^{(1)}, f^*)$ , and a public/secret key-pair  $(\text{pk}, \text{sk}) \leftarrow \text{PKE.Setup}(1^\lambda)$ . It commits to the function  $F_{\text{sk}, f^*}$  in the privacy game, and receives a constrained key  $k_{F^*}^{(2)}$  from the challenger. Finally, it gives  $(k_{f^*}^{(1)}, k_{F^*}^{(2)}, \text{pk})$  to  $\mathcal{A}$ .
- **Query phase.** Algorithm  $\mathcal{B}$  answers all of the queries exactly as described in  $\text{Hyb}_5$ , except whenever it needs to evaluate  $F_2(k^{(2)}, \text{ct})$ , it instead queries its evaluation oracle on  $\text{ct}$ , and uses the response of its evaluation oracle in place of  $F_2(k^{(2)}, \text{ct})$ .

In  $\text{Real}_{\mathcal{B}, F_2}$ , the constrained key  $k_{f^*}^{(1)}$  is constructed by first computing  $k^{(2)} \leftarrow F_2.\text{Setup}(1^\lambda)$  and then  $k_{F^*}^{(2)} \leftarrow F_2.\text{Constrain}(k^{(2)}, F_{\text{sk}, f^*})$ . The evaluation oracle is implemented by  $F_2(k^{(2)}, \cdot)$ . This

precisely corresponds to  $\text{Hyb}_5$ . In  $\text{Ideal}_{\mathcal{B}, \mathcal{S}, F_2}$ , the constrained key is constructed by computing by invoking  $\mathcal{S}_1(1^\lambda)$ , and the evaluations  $F_2(k^{(2)}, \text{ct})$  is implemented by  $\mathcal{S}_2(\text{ct}, F_{\text{sk}, f^*}(\text{ct}), \text{st}_{\mathcal{S}})$ , which precisely corresponds to the distribution in  $\text{Hyb}_6$ . Thus, if  $\mathcal{A}$  is able to distinguish between hybrids  $\text{Hyb}_5$  and  $\text{Hyb}_6$  with non-negligible advantage, then  $\mathcal{B}$  is able to break privacy of  $F_2$  with the same probability.  $\square$

**Lemma B.7.** *If PKE is a CCA-secure public-key encryption scheme (Definition 2.1), then for all efficient adversaries  $\mathcal{A}$ , we have that  $|\Pr[\text{Hyb}_6(\mathcal{A}) = 1] - \Pr[\text{Hyb}_7(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Suppose there exists an adversary  $\mathcal{A}$  that can distinguish between hybrids  $\text{Hyb}_6$  and  $\text{Hyb}_7$ . We construct an adversary  $\mathcal{B}$  to break CCA-security of PKE. Algorithm  $\mathcal{B}$  simulates the experiments as follows:

- **Setup phase.** Adversary  $\mathcal{A}$  begins by committing to a function  $f^* \in \mathcal{F}$ . Algorithm  $\mathcal{B}$  generates the keys  $k^{(1)} \leftarrow F_1.\text{Setup}(1^\lambda)$ ,  $k_{f^*}^{(1)} \leftarrow F_1.\text{Constrain}(k^{(1)}, f)$ , and  $(k_{F^*}^{(2)}, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$ . It receives  $\text{pk}$  from the CCA-security challenger, and gives the constrained key  $(k_{f^*}^{(1)}, k_{F^*}^{(2)}, \text{pk})$  to  $\mathcal{A}$ .
- **Query phase.** Algorithm  $\mathcal{B}$  answers  $\mathcal{A}$ 's queries as follows:
  - **Evaluation queries.** On an evaluation query  $x \in \mathcal{X}$ ,  $\mathcal{B}$  computes  $r_x \leftarrow F_1(k^{(1)}, x)$ , and  $\text{ct} \leftarrow \text{PKE}.\text{Encrypt}(\text{pk}, x; r_x)$ . Then, it computes  $(z, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(\text{ct}, f^*(x), \text{st}_{\mathcal{S}})$  and returns  $(\text{ct}, z)$ .
  - **Pre-challenge inversion queries.** On input  $(y_1, y_2) \in \mathcal{Y}$ , algorithm  $\mathcal{B}$  first submits  $y_1$  to the decryption oracle to obtain some value  $x$ . The rest of the query handling proceeds as in  $\text{Hyb}_6$  and  $\text{Hyb}_7$ .
  - **Challenge query.** On input  $x^* \in \mathcal{X}$ ,  $\mathcal{B}$  samples a random point  $\hat{x}^* \leftarrow^{\mathcal{R}} \mathcal{X}$ . It submits the pair  $(x^*, \hat{x}^*)$  as its challenge in the CCA-security game, and receives a challenger ciphertext  $\text{ct}^*$ . Next,  $\mathcal{B}$  samples a random point  $z^* \leftarrow^{\mathcal{R}} \mathcal{X}$  and gives  $(\text{ct}^*, z^*)$  to  $\mathcal{A}$ .
  - **Post-challenge inversion query.** On input  $(y_1, y_2) \in \mathcal{Y}$ , if  $y_1 = \text{ct}^*$ , then  $\mathcal{B}$  replies with  $\perp$ . Otherwise,  $\mathcal{B}$  submits  $y_1$  to the decryption oracle to obtain some value  $x$ . The rest of the query handling proceeds as in  $\text{Hyb}_6$  and  $\text{Hyb}_7$ .

By construction,  $\mathcal{B}$  never queries the decryption oracle on the challenge ciphertext  $\text{ct}^*$ , and thus, is admissible. To conclude the proof, we show that if the challenge ciphertext  $\text{ct}^*$  is an encryption of  $x^*$ , then  $\mathcal{B}$  correctly simulates the distribution in  $\text{Hyb}_6$  and if  $\text{ct}^*$  is an encryption of  $\hat{x}^*$ , then  $\mathcal{B}$  correctly simulates the distribution in  $\text{Hyb}_7$ . Clearly,  $\mathcal{B}$  perfectly simulates the setup phase, the evaluation queries, and the pre-challenge inversion queries for  $\mathcal{A}$ . It suffices to reason about the challenge query and the post-challenge inversion queries.

- Suppose  $\text{ct}^*$  is an encryption of  $x^*$ . Then, the challenge query is simulated exactly as in  $\text{Hyb}_6$  (recall that in  $\text{Hyb}_6$ , the ciphertext  $\text{ct}^*$  is an encryption of  $x^*$  with uniform randomness). For the (post-challenge) inversion queries, it suffices to argue that in  $\text{Hyb}_6$ , the challenger's response to any (post-challenge) inversion query of the form  $(\text{ct}^*, z)$  for any  $z \in \mathcal{X}$  is  $\perp$ . In  $\text{Hyb}_6$ ,  $\text{ct}^*$  is an encryption of  $x^*$  using randomness that is sampled uniformly at random. By Lemma 2.2,  $\Pr[\text{ct}^* = \text{PKE}.\text{Encrypt}(\text{pk}, x^*; r_{x^*})] = \text{negl}(\lambda)$  where  $r_{x^*} = F_1(k^{(1)}, x^*)$ . Thus, with overwhelming probability, the output on the inversion queries of the form  $(\text{ct}^*, z)$  in  $\text{Hyb}_6$  is  $\perp$ , and so the simulation is correct.



- Suppose  $\text{ct}^*$  is an encryption of  $\hat{x}^*$ . Then, the challenge query is simulated exactly as in  $\text{Hyb}_7$ . For the (post-challenge) inversion queries, it suffices to argue that in  $\text{Hyb}_7$ , the challenger's response to any (post-challenge) inversion query of the form  $(\text{ct}^*, z)$  for any  $z \in \mathcal{X}$  is  $\perp$ . This follows by a similar argument as in the previous case. The challenger's response to an inversion query of the form  $(\text{ct}^*, z)$  is not  $\perp$  only if  $\text{ct}^* = \text{PKE.Encrypt}(\text{pk}, \hat{x}^*; r_{\hat{x}^*})$  where  $r_{\hat{x}^*} = F_1(k^{(1)}, \hat{x}^*)$ . By Lemma 2.2, over the randomness used to sample  $\text{ct}^*$ , this occurs with negligible probability.

We conclude that if  $\mathcal{A}$  is able to distinguish between hybrids  $\text{Hyb}_6$  and  $\text{Hyb}_7$  with non-negligible probability, then  $\mathcal{B}$  breaks CCA-security of PKE with the same probability.  $\square$

**Lemma B.8.** *If  $F_2$  is a single-key, selectively-function-private constrained PRF (Definition 3.14), then for all efficient adversaries  $\mathcal{A}$ , we have that  $|\Pr[\text{Hyb}_7(\mathcal{A}) = 1] - \Pr[\text{Hyb}_8(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Follows by an analogous argument as that in the proof of Lemma B.6.  $\square$

**Lemma B.9.** *If PKE is a CCA-secure public-key encryption scheme (Definition 2.1) and  $F_1$  is a single-key, selective-function-secure constrained PRF, then for all efficient adversaries  $\mathcal{A}$ , we have that  $|\Pr[\text{Hyb}_8(\mathcal{A}) = 1] - \Pr[\text{Hyb}_9(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Follows by an analogous argument as that in the proof of Lemma B.1.  $\square$

Combining Lemmas B.1 through B.9, we conclude that Construction 4.5 is a single-key, selective-function-secure circuit-constrained IPF.  $\square$

## C Analysis of Multi-Key Circuit-Constrained IPF

In this section, we give the formal correctness and security analysis of our multi-key circuit-constrained IPF (Construction 7.2) from Section 7.

### C.1 Proof of Theorem 7.3

Correctness of Construction 7.2 follows from the correctness of the underlying puncturable PRFs and correctness of  $i\mathcal{O}$ . Take any function  $f \in \mathcal{F}$ , and let  $k = (k^{(1)}, k^{(2)}) \leftarrow \text{F.Setup}(1^\lambda)$ . Let  $k_f = (P_1, P_2) \leftarrow \text{F.Constrain}(k, f)$ . We consider each of the two correctness requirements separately.

- By correctness of  $i\mathcal{O}$ ,  $P_1(x) = P^{\text{Eval}}[f, k^{(1)}, k^{(2)}](x)$  for all  $x \in \mathcal{X}$ . Since  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}](x) = F(k, x)$  for all  $x$  where  $f(x) = 1$ ,  $F.\text{Eval}(k_f, x) = P_1(x) = F(k, x)$  for all  $x$  where  $f(x) = 1$ .
- By correctness of  $i\mathcal{O}$ ,  $F.\text{Eval}^{-1}(k_f, y) = P_2(y) = P^{\text{Inv}}[f, k^{(1)}, k^{(2)}](y)$  for all  $y \in \mathcal{Y}$ . Take any input  $y = (y_1, y_2) \in \mathcal{V} \times \mathcal{X}$  where there exists  $x \in \mathcal{X}$  such that  $f(x) = 1$  and  $F(k, x) = (y_1, y_2)$ . This means that  $y_1 = F_1(k^{(1)}, x)$  and  $y_2 = F_2(k^{(2)}, y_1) \oplus x$ . Then, by construction,  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}](y) = x$ .  $\square$

## C.2 Proof of Theorem 7.4

Our proof proceeds via a sequence of hybrid experiments between an adversary  $\mathcal{A}$  and a challenger. Similar to the proofs of Theorem 4.3 and Theorem 4.7, our hybrid experiments consist of several phases. Unlike the case of previous theorems, we work in the multi-key setting, where the adversary is allowed to request constrained keys for multiple functions. In this case, selective security refers to selectivity in the choice of the challenge query. In other words, the adversary is required to commit to its challenge query at the beginning of the game. We now give the general structure of our hybrid experiments:

- **Setup phase.** In the selective-security experiment, the adversary begins by committing to a point  $x^* \in \mathcal{X}$ . Then, the challenger generates an IPF key  $k \in \mathcal{K}$  and a challenge evaluation  $y^* \in \mathcal{Y}$ . The challenger gives  $y^*$  to the adversary.
- **Query phase.** The adversary  $\mathcal{A}$  is now allowed to make constrain, evaluation and inversion queries. By the admissibility requirement (Definition 3.12), the queries that the adversary makes must satisfy the following properties:
  - For all constrain queries  $f \in \mathcal{F}$  the adversary makes,  $f(x^*) = 0$ .
  - For all evaluation queries  $x \in \mathcal{X}$  the adversary makes,  $x \neq x^*$ .
  - For all inversion queries  $y \in \mathcal{Y}$  the adversary makes,  $y \neq y^*$ .
- **Output phase.** At the end of the experiment, the adversary outputs a bit  $b \in \{0, 1\}$ .

We now define our sequence of hybrid experiments. When defining a new hybrid, we only describe the phases that differ from the previous one.

- **Hyb<sub>0</sub>:** This is the constrained IPF security experiment  $\text{Expt}_{\mathcal{A}, \mathcal{F}}^{(\text{IPF})}(\lambda, 0)$  from Definition 3.11. During the setup phase, after the adversary commits to a point  $x^* \in \mathcal{X}$ , the challenger samples PRF keys  $k^{(1)} \leftarrow \text{F}_1.\text{Setup}(1^\lambda)$ ,  $k^{(2)} \leftarrow \text{F}_2.\text{Setup}(1^\lambda)$  and sets  $k = (k^{(1)}, k^{(2)})$ . Next, it computes  $v^* \leftarrow \text{F}_1(k^{(1)}, x^*)$  and  $z^* \leftarrow \text{F}_2(k^{(2)}, v^*)$ . It then gives the challenge  $y^* = (v^*, z^* \oplus x^*)$  to the adversary. The challenger answers the constrain, evaluation, and inversion queries using  $\text{F.Constrain}(k, \cdot)$ ,  $\text{F}(k, \cdot)$ , and  $\text{F}^{-1}(k, \cdot)$ , respectively.
- **Hyb<sub>1</sub>:** Same as Hyb<sub>0</sub>, except the challenger substitutes a punctured key  $k_{x^*}^{(1)}$  for  $k^{(1)}$  and a punctured evaluation algorithm  $\text{F}_1.\text{Eval}(k_{x^*}^{(1)}, \cdot)$  for  $\text{F}_1(k^{(1)}, \cdot)$  when constructing the obfuscated programs  $P_1$  and  $P_2$  in the constrain queries. More concretely, during the setup phase, the challenger sets  $k_{x^*}^{(1)} \leftarrow \text{F}_1.\text{Puncture}(k^{(1)}, x^*)$ . When responding to the constrain queries, the challenger constructs  $P_1 \leftarrow i\mathcal{O}(P^{\text{Eval}}[f, k_{x^*}^{(1)}, k^{(2)}])$  and  $P_2 \leftarrow i\mathcal{O}(P^{\text{Inv}}[f, k_{x^*}^{(1)}, k^{(2)}])$ , where we write  $P^{\text{Eval}}[f, k_{x^*}^{(1)}, k^{(2)}]$  and  $P^{\text{Inv}}[f, k_{x^*}^{(1)}, k^{(2)}]$  to denote the programs in Figures 1 and 2, respectively, where  $\text{F}_1(k^{(1)}, \cdot)$  is replaced by  $\text{F}_1.\text{Eval}(k_{x^*}^{(1)}, \cdot)$ .
- **Hyb<sub>2</sub>:** Same as Hyb<sub>1</sub>, except the challenger samples  $v^* \xleftarrow{\mathbb{R}} \mathcal{V}$  during the setup phase. The remaining of the setup phase is unchanged. When responding to the evaluation and inversion queries, the challenger uses  $v^*$  in place of the value  $\text{F}_1(k^{(1)}, x^*)$ .
- **Hyb<sub>3</sub>:** Same as Hyb<sub>2</sub>, except when responding to the evaluation and inversion queries, the challenger always evaluates  $\text{F}_1(k^{(1)}, \cdot)$  instead of substituting the value  $v^*$  for  $\text{F}_1(k^{(1)}, x^*)$ .

- **Hyb<sub>4</sub>**: Same as **Hyb<sub>3</sub>**, except the challenger reverts to using the real key  $k^{(1)}$  (instead of  $k_{x^*}^{(1)}$ ) and the real evaluation algorithm  $F_1(k^{(1)}, \cdot)$  (instead of  $F_1.\text{Eval}(k_{x^*}^{(1)}, \cdot)$ ) when constructing the obfuscated programs  $P_1$  and  $P_2$  in the constrain queries.
- **Hyb<sub>5</sub>**: Same as **Hyb<sub>4</sub>**, except the challenger substitutes a punctured key  $k_{v^*}^{(2)}$  for  $k^{(2)}$  and a punctured evaluation algorithm  $F_2.\text{Eval}(k_{v^*}^{(2)}, \cdot)$  for  $F_2(k^{(2)}, \cdot)$  when constructing the obfuscation programs  $P_1$  and  $P_2$  in the constrain queries. More concretely, during the setup phase, after the challenger samples  $v^* \xleftarrow{\mathcal{R}} \mathcal{V}$ , the challenger sets  $k_{v^*}^{(2)} \leftarrow F_2.\text{Puncture}(k^{(2)}, v^*)$ . When responding to the constrain queries, the challenger constructs  $P_1 \leftarrow i\mathcal{O}(P^{\text{Eval}}[f, k^{(1)}, k_{v^*}^{(2)}])$  and  $P_2 \leftarrow i\mathcal{O}(P^{\text{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}])$ , where we write  $P^{\text{Eval}}[f, k^{(1)}, k_{v^*}^{(2)}]$  and  $P^{\text{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}]$  to denote the programs in Figures 1 and 2, respectively, where  $F_2(k^{(2)}, \cdot)$  is replaced by  $F_2.\text{Eval}(k_{v^*}^{(2)}, \cdot)$ .
- **Hyb<sub>6</sub>**: Same as **Hyb<sub>5</sub>**, except during the setup phase, the challenger samples  $z^* \xleftarrow{\mathcal{R}} \mathcal{X}$ , and sets the challenge to be  $y^* = (v^*, z^*)$ . The other steps in the setup phase remain unchanged. When responding to the evaluation and inversion queries, the challenger uses  $(z^* \oplus x^*)$  in place of the value  $F_2(k^{(2)}, v^*)$ .
- **Hyb<sub>7</sub>**: Same as **Hyb<sub>6</sub>**, except when responding to the evaluation and inversion queries, the challenger always computes  $F_2(k^{(2)}, \cdot)$  instead of substituting the value  $(z^* \oplus x^*)$  for  $F_2(k^{(2)}, v^*)$ .
- **Hyb<sub>8</sub>**: Same as **Hyb<sub>7</sub>**, except for each constrain query that the adversary makes, instead of using the program  $P^{\text{Eval}}[f, k^{(1)}, k_{v^*}^{(2)}]$  and  $P^{\text{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}]$ , the challenger uses  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}]$  and  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}]$  by replacing  $F_2.\text{Eval}(k_{v^*}^{(2)}, \cdot)$  with  $F_2(k^{(2)}, \cdot)$ . It sets  $P_1 \leftarrow i\mathcal{O}(P^{\text{Eval}}[f, k^{(1)}, k^{(2)}])$  and  $P_2 \leftarrow i\mathcal{O}(P^{\text{Inv}}[f, k^{(1)}, k^{(2)}])$  and uses  $k_f = (P_1, P_2)$  as the constrained key. This is the constrained IPF security experiment  $\text{Expt}_{\mathcal{A}, F}^{(\text{IPF})}(\lambda, 1)$  from Definition 3.11.

As in the proofs of Theorems 4.3 and 4.7, we write  $\text{Hyb}(\mathcal{A})$  to denote the output of experiment  $\text{Hyb}_i$  with an adversary  $\mathcal{A}$ . In the following lemmas, we make the implicit assumptions that the adversary  $\mathcal{A}$  is admissible for the constrained IPF security game.

**Lemma C.1.** *If  $i\mathcal{O}$  is an indistinguishability obfuscator (Definition 7.1), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_0(\mathcal{A}) = 1] - \Pr[\text{Hyb}_1(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* The two experiments are identical except the challenger uses the programs  $P^{\text{Eval}}[f, k_{x^*}^{(1)}, k^{(2)}]$  and  $P^{\text{Inv}}[f, k_{x^*}^{(1)}, k^{(2)}]$  in place of  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}]$  and  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}]$ , respectively, in **Hyb<sub>1</sub>** when answering the constrain queries. We now show that by security of  $i\mathcal{O}$ , for all admissible constraint functions  $f \in \mathcal{F}$ , the obfuscated programs  $P_1$  and  $P_2$  are computationally indistinguishable in **Hyb<sub>0</sub>** and **Hyb<sub>1</sub>**. The lemma then follows by a standard hybrid argument (over the constrain queries). We reason about each case individually:

- First, we show that the distribution of  $P_1$  in **Hyb<sub>0</sub>** and **Hyb<sub>1</sub>** is computationally indistinguishable. It suffices to show that for all  $x \in \mathcal{X}$ ,  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}](x) = P^{\text{Eval}}[f, k_{x^*}^{(1)}, k^{(2)}](x)$ . Take  $x \in \mathcal{X}$ . By correctness of  $F_1$ , we have that  $F_1(k^{(1)}, x) = F_1.\text{Eval}(k_{x^*}^{(1)}, x)$  for all  $x \neq x^*$ . Thus,  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}](x) = P^{\text{Eval}}[f, k_{x^*}^{(1)}, k^{(2)}](x)$  for all  $x \neq x^*$ . When  $x = x^*$ , we have that  $f(x^*) = 0$  by admissibility. Then,  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}](x^*) = \perp = P^{\text{Eval}}[f, k_{x^*}^{(1)}, k^{(2)}](x^*)$ . The claim follows by security of  $i\mathcal{O}$ .

- Next, we show that the distribution of  $P_2$  in  $\text{Hyb}_0$  and  $\text{Hyb}_1$  is computationally indistinguishable. As above, we show that for all  $y \in \mathcal{Y}$ ,  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}](y) = P^{\text{Inv}}[f, k_{x^*}^{(1)}, k^{(2)}](y)$ . Take any  $y = (y_1, y_2) \in \mathcal{Y}$ . By construction of  $P^{\text{Inv}}$  and correctness of  $F_1$ ,  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}](y) = P^{\text{Inv}}[f, k_{x^*}^{(1)}, k^{(2)}](y)$  whenever  $F_2(k^{(2)}, y_1) \oplus y_2 \neq x^*$ . When  $F_2(k^{(2)}, y_1) \oplus y_2 = x^*$ , we have that  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}](y) = \perp = P^{\text{Inv}}[f, k_{x^*}^{(1)}, k^{(2)}](x^*)$  because  $f(x^*) = 0$  by admissibility. The claim follows by security of  $i\mathcal{O}$ .  $\square$

**Lemma C.2.** *If  $F_1$  is a selectively-secure puncturable PRF (Definition 3.7), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_1(\mathcal{A}) = 1] - \Pr[\text{Hyb}_2(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Follows by essentially the same argument as that used in the proof of Lemma A.1. The only difference is that the reduction algorithm must additionally simulate the constrained key queries, but these can be handled exactly as described in  $\text{Hyb}_1$  and  $\text{Hyb}_2$  (using the punctured key  $k_{x^*}^{(1)}$ ).  $\square$

**Lemma C.3.** *If  $F_1$  is a selectively-secure puncturable PRF (Definition 3.12), and  $1/|\mathcal{V}| = \text{negl}(\lambda)$ , then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_2(\mathcal{A}) = 1] - \Pr[\text{Hyb}_3(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Follows by essentially the same argument as that used in the proof of Lemma A.2. Note that the constrain queries are handled identically in  $\text{Hyb}_2$  and  $\text{Hyb}_3$ , so they do not complicate the proof.  $\square$

**Lemma C.4.** *If  $i\mathcal{O}$  is an indistinguishability obfuscator (Definition 7.1), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_3(\mathcal{A}) = 1] - \Pr[\text{Hyb}_4(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Follows by the same argument as that used in the proof of Lemma C.1.  $\square$

**Lemma C.5.** *If  $i\mathcal{O}$  is an indistinguishability obfuscator (Definition 7.1) and  $|\mathcal{X}|/|\mathcal{V}| = \text{negl}(\lambda)$ , then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_4(\mathcal{A}) = 1] - \Pr[\text{Hyb}_5(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* The two experiments are identical except the challenger uses the programs  $P^{\text{Eval}}[f, k^{(1)}, k_{v^*}^{(2)}]$  and  $P^{\text{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}]$  in place of  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}]$  and  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}]$ , respectively, in  $\text{Hyb}_5$  when responding to the constrain queries. As in the proof of Lemma C.1, it suffices to show that by security of  $i\mathcal{O}$ , for all admissible constraint functions  $f \in \mathcal{F}$ , the obfuscated programs  $P_1$  and  $P_2$  in each constrain query is computationally indistinguishable. We reason about the two cases separately:

- First, we show that distribution of  $P_1$  in  $\text{Hyb}_4$  and  $\text{Hyb}_5$  is computationally indistinguishable. It suffices to show that for all  $x \in \mathcal{X}$ ,  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}](x) = P^{\text{Eval}}[f, k^{(1)}, k_{v^*}^{(2)}](x)$ . Take  $x \in \mathcal{X}$ . Since  $v^*$  is uniform over  $\mathcal{V}$  (and sampled independently of  $k^{(1)}$ ), with probability  $1 - |\mathcal{X}|/|\mathcal{V}| = 1 - \text{negl}(\lambda)$ , there does not exist any  $x \in \mathcal{X}$  such that  $F_1(k^{(1)}, x) = v^*$ . Thus, with overwhelming probability,  $F_1(k^{(1)}, x) \neq v^*$ . By correctness of  $F_2$ ,  $F_2(v) = F_2.\text{Eval}(k^{(2)}, v)$  for all  $v \neq v^*$ . This means that with overwhelming probability,  $P^{\text{Eval}}[f, k^{(1)}, k^{(2)}](x) = P^{\text{Eval}}[f, k^{(1)}, k_{v^*}^{(2)}](x)$  on all  $x \in \mathcal{X}$ , in which case the claim follows by  $i\mathcal{O}$  security.
- Next, we show that the distribution of  $P_2$  in  $\text{Hyb}_4$  and  $\text{Hyb}_5$  is computationally indistinguishable. It suffices to show that for all  $y \in \mathcal{Y}$ ,  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}](y) = P^{\text{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}](y)$  for all  $y \in \mathcal{Y}$ . Take any  $y = (y_1, y_2) \in \mathcal{Y}$ . By correctness of  $F_2$ , we have that  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}](y_1, y_2) = P^{\text{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}](y_1, y_2)$  whenever  $y_1 \neq v^*$ . Consider the case where  $y_1 = v^*$ . By the same

argument as in the previous case, with overwhelming probability, there does not exist any  $x \in \mathcal{X}$  where  $F_1(k^{(1)}, x) = v^*$ . Thus, with overwhelming probability (over the choice of  $v^*$ ), the programs  $P^{\text{Inv}}[f, k^{(1)}, k^{(2)}]$  and  $P^{\text{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}]$  both output  $\perp$  whenever  $y_1 = v^*$ . The claim then follows by  $i\mathcal{O}$  security.  $\square$

**Lemma C.6.** *If  $F_2$  is a selectively-secure puncturable PRF (Definition 3.12), then for all efficient adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_5(\mathcal{A}) = 1] - \Pr[\text{Hyb}_6(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Follows by essentially the same argument as that used in the proof of Lemma A.3. The only difference is that the reduction algorithm additionally needs to simulate the constrained key queries, but these can be handled exactly as described in  $\text{Hyb}_5$  and  $\text{Hyb}_6$  (using the punctured key  $k_{v^*}^{(2)}$ ).  $\square$

**Lemma C.7.** *If  $|\mathcal{X}| / |\mathcal{V}| = \text{negl}(\lambda)$ , then for all adversaries  $\mathcal{A}$ ,  $|\Pr[\text{Hyb}_6(\mathcal{A}) = 1] - \Pr[\text{Hyb}_7(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Follows by an analogous argument as that in the proof of Lemma A.4.  $\square$

**Lemma C.8.** *If  $i\mathcal{O}$  is an indistinguishable obfuscator, then for all efficient adversaries  $\mathcal{A}$ , we have that  $|\Pr[\text{Hyb}_7(\mathcal{A}) = 1] - \Pr[\text{Hyb}_8(\mathcal{A}) = 1]| = \text{negl}(\lambda)$ .*

*Proof.* Follows by an analogous argument as that in the proof of Lemma C.5.  $\square$

Combining Lemmas C.1 through C.8, we conclude that Construction 7.2 is a selectively-secure multi-key IPF for general circuit constraints.  $\square$