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 ($i\mathcal{O}$). 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 for many classes of constraints 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 $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ equipped with an efficient inversion algorithm $\mathsf{F}^{-1} : \mathcal{K} \times \mathcal{Y} \to \mathcal{X} \cup \{\perp\}$. The inversion algorithm is required to satisfy the following two properties for all $k \in \mathcal{K}$:

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

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 $f(x) := \mathsf{F}(k, x)$ and its inverse $f^{-1}(x) := \mathsf{F}^{-1}(k, x)$, where k is randomly chosen in $\mathcal{K}$. In the other experiment, the adversary is given oracles for a *random* injective function $g : \mathcal{X} \to \mathcal{Y}$ and its inverse $g^{-1} : \mathcal{Y} \to \mathcal{X} \cup \{\perp\}$. 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 $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ 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 $g \colon \mathcal{X} \to \{0,1\}$, 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 $g_1, \ldots, g_Q$ 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 $g_1(x) = \cdots = g_Q(x) = 0$. 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 ($i\mathcal{O}$) [SW14].

The simplest non-trivial 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 $y = \mathsf{F}(k, x') \in \mathcal{Y}$ 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 for many constraint classes 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: the constrained key holder can only encrypt/decrypt messages that satisfy a certain predicate.

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 $\mathsf{F}_1 \colon \mathcal{K}_1 \times \mathcal{X} \to \mathcal{V}$ and $\mathsf{F}_2 \colon \mathcal{K}_2 \times \mathcal{V} \to \mathcal{X}$ be two secure PRFs. Define the following IPF F on domain $\mathcal{X}$ using a key $k = (k^{(1)}, k^{(2)}) \in \mathcal{K}_1 \times \mathcal{K}_2$:

$$\begin{array}{ll} \mathsf{F}\big((k^{(1)}, k^{(2)}),\ x\big) := & \mathsf{F}^{-1}\big((k^{(1)}, k^{(2)}),\ (y_1, y_2)\big) := \\ \left\{ \begin{array}{l} y_1 \leftarrow \mathsf{F}_1(k^{(1)}, x) \\ y_2 \leftarrow x \oplus \mathsf{F}_2(k^{(2)}, y_1) \\ \text{output } (y_1, y_2) \end{array} \right\} & \left\{ \begin{array}{l} x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 \\ \text{if } \mathsf{F}_1(k^{(1)}, x) \neq y_1 \text{ then } x \leftarrow \bot \\ \text{output } x \end{array} \right\} \end{array} \tag{1.1}$$

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 $\mathsf{F}(k,x)$ from a random pair in $\mathcal{V} \times \mathcal{X}$: given a challenge value (y_1, y_2) for $\mathsf{F}(k,x)$, the adversary can simply test if $x = \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$. This will be satisfied by $\mathsf{F}(k,x)$, 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 $y_1 := \mathsf{F}_1(k^{(1)}, x)$. 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 $\mathsf{F}(k,x) = (y_1, y_2)$ 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 *private puncturable* PRF for F_2. Currently the best constructions for a private puncturable PRF rely on the LWE assumption [BKM17, CC17, BTVW17]. It is not known how to construct a private puncturable PRF from one-way functions. We show in Theorem 4.3 that with this setup, the puncturable IPF in (1.1) is secure.

A constrained IPF for circuit constraints. Next we generalize (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 (1.1) we must puncture F_2 at all points $\mathsf{F}_1(k^{(1)}, x)$ 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 $v = \mathsf{F}_1(k^{(1)}, x)$ with $g(x) = 0$. Because F_1 is not invertible, this cannot be done even given $k^{(1)}$.

We solve this problem by replacing $\mathsf{F}_1(k^{(1)}, x)$ with a CCA-secure public-key encryption $\mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$, where the randomness $r_x = \mathsf{F}_1(k^{(1)}, x)$ is derived from F_1 and pk is the public key. In this case, the input to F_2 is a ciphertext ct that encrypts the point x. The output of the IPF is the pair $(\mathsf{ct},\ \mathsf{F}_2(k^{(2)}, \mathsf{ct}) \oplus x)$. When constraining F_2, we embed the secret decryption key sk for the public-key encryption scheme in the constrained key. Then, on an input ciphertext ct, the constraint function first decrypts ct (using sk) to obtain a value $x \in \mathcal{X}$, and then checks if $g(x) = 1$. Because knowledge of sk allows one to invert on *all* points, it is *critical* that the constrained key hides 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 (ct, y_2), the evaluator uses the PRF key $k^{(2)}$ to compute a (candidate) preimage $x \leftarrow \mathsf{F}_2(k^{(2)}, \mathsf{ct}) \oplus y_2$, and the "trapdoor" method where an evaluator who holds the decryption key for the public-key encryption scheme simply decrypts 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 complete 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 (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 impossibility results for constrained PRPs rule out any constraint class that enables evaluation on a non-negligible fraction of the domain. For example, this rules out the possibility of a puncturable PRP. Can we build constrained PRPs for constraint families that allow evaluation on a more restricted subset of the domain? For instance, do prefix-constrained PRPs exist?

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 for inputs of unbounded length [DKW16, DDM17].

2 Preliminaries

For a positive integer n, we write $[n]$ to denote the set $\{1, 2, \ldots, n\}$. 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 $x \xleftarrow{\text{R}} S$ to denote that x is sampled uniformly from S. Throughout this work, we write λ for the security parameter. We say a function $f(\lambda)$ is negligible in λ 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 λ. We say that an event occurs with overwhelming probability if its complement occurs with negligible probability, and that it occurs with noticeable probability if it 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 $\mathsf{Funs}[\mathcal{X}, \mathcal{Y}]$ to denote the set of functions from $\mathcal{X}$ to $\mathcal{Y}$. We write $\mathsf{InjFuns}[\mathcal{X}, \mathcal{Y}]$ to denote the set of *injective* functions from $\mathcal{X}$ to $\mathcal{Y}$. For an injective function $f \in \mathsf{InjFuns}[\mathcal{X}, \mathcal{Y}]$, we denote by $f^{-1} : \mathcal{Y} \to \mathcal{X} \cup \{\perp\}$ the function where $f^{-1}(y) = x$ if $y = f(x)$, and $\perp$ if there is no such $x \in \mathcal{X}$. We sometimes refer to f^{-1} as the (generalized) inverse of f. When the domain and range are the same, the set $\mathsf{InjFuns}[\mathcal{X}, \mathcal{X}]$ is precisely the set of permutations on $\mathcal{X}$.

2.1 CCA-Secure Public-Key Encryption

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

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

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

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

Definition 2.1 (CCA-Security [NY90, RS91])**.** Let $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ 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 λ and a bit $b \in \{0, 1\}$, we define the CCA-security experiment $\mathsf{Expt}^{(\mathsf{CCA})}_{\mathcal{A},\mathsf{PKE}}(\lambda, b)$ as follows. The challenger first samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$. The adversary can then issue decryption oracle queries and up to one challenge oracle query.[1] Depending on the bit $b \in \{0, 1\}$, the challenger responds to each query as follows:

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

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 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 $\mathcal{A}$,

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

[1] 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).

Smoothness. In our security analysis, we require that our public-key encryption scheme satisfy an additional smoothness property. We say that a public-key encryption scheme is smooth if every message can encrypt to a super-polynomial number of potential ciphertexts. This property is satisfied by most natural public-key encryption schemes. After all, if the adversary can find a message m that has only polynomially-many ciphertexts, then the adversary can trivially break semantic security of the scheme. Of course, it is possible to craft public-key encryption schemes [BHK15] where there exist (hard-to-find) messages that encrypt to only polynomially-many ciphertexts. We give the formal definition of smoothness in Definition 2.2.

Definition 2.2 (Smoothness [BHK15, adapted])**.** A PKE scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ with message space $\mathcal{M}$ and ciphertext space $\mathcal{T}$ is *smooth* if for all messages $m \in \mathcal{M}$ and all strings $\mathsf{ct} \in \mathcal{T}$,

$$\Pr\left[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda) : \mathsf{PKE.Encrypt}(\mathsf{pk}, m) = \mathsf{ct}\right] = \operatorname{negl}(\lambda),$$

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

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 $\mathsf{F} \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ that can be computed by a deterministic polynomial-time algorithm. A PRF can also include a setup algorithm $\mathsf{F.Setup}(1^\lambda)$ that on input the security parameter λ, outputs a key $k \in \mathcal{K}$. A function F is a secure PRF if for all efficient adversaries $\mathcal{A}$,

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

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 $\mathsf{F} \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ and $\mathsf{F}^{-1} \colon \mathcal{K} \times \mathcal{Y} \to \mathcal{X} \cup \{\bot\}$. An IPF can also include a setup algorithm $\mathsf{F.Setup}(1^\lambda)$ that on input the security parameter λ, 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 λ and all keys k output by $\mathsf{F.Setup}(1^\lambda)$, the function $\mathsf{F}(k, \cdot)$ is an injective function from $\mathcal{X}$ to $\mathcal{Y}$. Moreover, the function $\mathsf{F}^{-1}(k, \cdot)$ is the (generalized) inverse of $\mathsf{F}(k, \cdot)$.

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

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

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 $\mathsf{F}(k,\cdot)$, but not an inversion oracle $\mathsf{F}^{-1}(k,\cdot)$. 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 $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is said to be constrained with respect to a predicate family $\mathcal{F} = \{f : \mathcal{X} \to \{0,1\}\}$ if there are two additional algorithms $(\mathsf{F.Constrain}, \mathsf{F.Eval})$ with the following properties:

- $\mathsf{F.Constrain}(k, f) \to k_f$: 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.
- $\mathsf{F.Eval}(k_f, x) \to y$: 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 $k \leftarrow \mathsf{F.Setup}(1^\lambda)$, every function $f \in \mathcal{F}$, and every input $x \in \mathcal{X}$ where $f(x) = 1$, we have that

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

Definition 3.6 (Constrained PRF Security Experiment)**.** Let $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ 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 $\mathsf{Expt}^{(\mathsf{PRF})}_{\mathcal{A},\mathsf{F}}(\lambda, b)$ (parameterized by a security parameter λ and a bit $b \in \{0,1\}$), the challenger begins by sampling a key $k \leftarrow \mathsf{F.Setup}(1^\lambda)$ and a random function $\mathsf{R} \xleftarrow{\mathrm{R}} \mathsf{Funs}[\mathcal{X},\mathcal{Y}]$. 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 $k_f \leftarrow \mathsf{F.Constrain}(k, f)$.
- **Evaluation oracle.** On input a point $x \in \mathcal{X}$, the challenger returns $y = \mathsf{F}(k, x)$.
- **Challenge oracle.** On input a point $x \in \mathcal{X}$, the challenger returns $y = \mathsf{F}(k, x)$ to $\mathcal{A}$ if $b = 0$ and $y = \mathsf{R}(x)$ 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 $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ 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 $\mathcal{A}$,

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

Without loss of generality, we restrict the adversary to make at most one challenge query in the constrained PRF security experiment.[2]

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

- $\mathsf{F.Constrain}(k, f) \to k_f$: 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.
- $\mathsf{F.Eval}(k_f, x) \to y$: On input a constrained key k_f and a value $x \in \mathcal{X}$, the evaluation algorithm outputs a value $y \in \mathcal{Y}$.

[2]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.

- $\mathsf{F.Eval}^{-1}(k_f, y) \to x$: 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 $k \leftarrow \mathsf{F.Setup}(1^\lambda)$, every function $f \in \mathcal{F}$, and $k_f \leftarrow \mathsf{F.Constrain}(k, f)$, the following two properties hold:

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

Definition 3.11 (Constrained IPF Security Experiment)**.** Let $(\mathsf{F}, \mathsf{F}^{-1})$ 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 $\mathsf{Expt}^{(\mathsf{IPF})}_{\mathcal{A},\mathsf{F}}(\lambda, b)$ is defined exactly as the constrained PRF security experiment $\mathsf{Expt}^{(\mathsf{PRF})}_{\mathcal{A},\mathsf{F}}(\lambda, b)$ (except with the IPF in place of the PRF and the random function R is sampled from $\mathsf{InjFuns}[\mathcal{X}, \mathcal{Y}]$), and 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 $\mathsf{F}^{-1}(k, y)$.

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 $(\mathsf{F}, \mathsf{F}^{-1})$ 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 $\mathcal{A}$,

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

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: $\mathcal{F} = \{f_{x^*} \colon \mathcal{X} \to \{0,1\} \mid x^* \in \mathcal{X}\}$, where $f_{x^*}(x) = 1$ for all $x \neq x^*$ and $f_{x^*}(x^*) = 0$. For notational convenience, when working with a puncturable PRF $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, we replace the $\mathsf{F.Constrain}$ algorithm with the $\mathsf{F.Puncture}$ 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], 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 $\mathsf{F} \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ 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 $\mathsf{Real}_{\mathcal{A},\mathsf{F}}(\lambda)$:
- $(f, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$
- $k \leftarrow \mathsf{F.Setup}(1^\lambda)$
- $k_f \leftarrow \mathsf{F.Constrain}(k, f)$
- $b \leftarrow \mathcal{A}^{\mathsf{F}(k,\cdot)}(k_f, \mathsf{st}_{\mathcal{A}})$
- Output b

Experiment $\mathsf{Ideal}_{\mathcal{A},\mathcal{S},\mathsf{F}}(\lambda)$:
- $(f, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$
- $(k_f, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$
- $b \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Eval}}(\cdot)}(k_f, \mathsf{st}_{\mathcal{A}})$, where the ideal evaluation oracle $\mathcal{O}_{\mathsf{Eval}}(\cdot)$ takes as input a point $x \in \mathcal{X}$, computes $(y, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(x, f(x), \mathsf{st}_{\mathcal{S}})$, 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 $\mathsf{F}(k, x)$. 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 (for any family of constraints) on polynomial-size domains do not exist. Next, we show that even over large domains, security for many natural classes of constraints, including puncturing, is impossible to achieve. 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 a non-negligible fraction of the domain).

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).

Theorem 3.16 (Limitations on Constrained PRPs)**.** *Let $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ be a PRP constrained with respect to a predicate family $\mathcal{F}$. For each predicate $f \in \mathcal{F}$, let $S_f = \{x \in \mathcal{X} : f(x) = 1\}$ denote the set of allowable points for f. If there exists $f \in \mathcal{F}$ where the quantity $|S_f| / |\mathcal{X}|$ is non-negligible, then F cannot be secure in the sense of Definition 3.12.*

Proof. Suppose there exists $f \in \mathcal{F}$ where $|S_f| / |\mathcal{X}|$ is non-negligible. We construct the following adversary for the constrained security game:

1. First, $\mathcal{A}$ makes a constrain query for f and a challenge query on an arbitrary $x^* \in \mathcal{X}$ where $f(x^*) = 0$. It receives from the challenger a punctured key k_f and a challenge value y^*.
2. Then, $\mathcal{A}$ computes $x \leftarrow \mathsf{F.Eval}^{-1}(k_f, y^*)$, and outputs 1 if either of the following conditions hold:
   - if $f(x) = 0$, or
   - if $\mathsf{F.Eval}(k_f, x) \neq y^*$.

   Otherwise, $\mathcal{A}$ outputs 0.

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

- Suppose $y^* = \mathsf{F}(k, x^*)$. Consider the case where $f(x) = 1$. Note in particular that this means $x \neq x^*$. By correctness of F, we have that $\mathsf{F.Eval}(k_f, x) = \mathsf{F}(k, x)$. Moreover, since $\mathsf{F}(k, \cdot)$ is a permutation, it follows that $\mathsf{F}(k, x) \neq \mathsf{F}(k, x^*) = y^*$. Thus, in this case, either $f(x) = 0$ or $\mathsf{F.Eval}(k_f, x) \neq y^*$, so we conclude that $\mathcal{A}$ outputs 1 with probability 1.
- Suppose y^* is uniformly random over $\mathcal{X}$. Let $\hat{x} = \mathsf{F}^{-1}(k, y^*)$. Suppose that $f(\hat{x}) = 1$. Then, by correctness of F, we have that
$$x = \mathsf{F.Eval}^{-1}(k_f, y^*) = \mathsf{F}^{-1}(k, y^*) = \hat{x}.$$
Moreover, since $f(\hat{x}) = 1$, we have
$$\mathsf{F.Eval}(k_f, x) = \mathsf{F.Eval}(k_f, \hat{x}) = \mathsf{F}(k, \hat{x}) = y^*.$$
Thus, whenever $f(\hat{x}) = 1$, adversary $\mathcal{A}$ outputs 1 with probability 0. Since y^* is uniformly random over $\mathcal{X}$ and $\mathsf{F}(k, \cdot)$ is a permutation,
$$\Pr[\mathcal{A} \text{ outputs } 1] \leq \Pr[f(\hat{x}) = 0] = 1 - |S_f| / |\mathcal{X}|.$$

We conclude that $\mathcal{A}$ breaks the constrained security of F with advantage $|S_f| / |\mathcal{X}|$, which is non-negligible by assumption. ☐

Corollary 3.17 (Puncturable PRPs are Insecure)**.** *Let* $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ *be a puncturable PRP. Then,* F *is insecure in the sense of Definition 3.12.*

Proof. The set of allowable points S_f for a puncturing constraint f is always $|\mathcal{X}| - 1$, so the ratio $|S_f| / |\mathcal{X}|$ is always non-negligible. The claim then follows from Theorem 3.16. ☐

Remark 3.18 (Constrained PRPs for Very Restricted Constraint Classes)**.** Theorem 3.16 rules out any constrained PRP that supports issuing constrained keys that can be used to evaluate on a non-negligible fraction of the domain. It does leave open the possibility of building constrained PRPs where each constrained key can only be used to evaluate on a *negligible* fraction of the domain. A natural class of constraints that satisfies this property is the class of *prefix-constrained PRPs* (for a prefix of super-logarithmic size). We leave it as an open problem to construct a prefix-constrained PRP, or more generally, a constrained PRP where all of the constrained keys can only be used to evaluate on a negligible fraction of the domain.

Remark 3.19 (Constrained IPFs Must be Expanding)**.** The attack from the proof of Theorem 3.16 also extends to the setting where $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a constrained *IPF* with a small range. Specifically, if $|\mathcal{Y}| \leq |\mathcal{X}| \cdot \mathrm{poly}(\lambda)$, and F supports issuing a constrained key for a function $f \colon \mathcal{X} \to \{0,1\}$ where $|S_f| / |\mathcal{X}|$ is non-negligible, then F cannot be secure in the sense of Definition 3.12. In this setting, we would modify the distinguisher in the proof of Theorem 3.16 to additionally output 1 if $x = \perp$. With this modification, the distinguishing advantage of the attack only decreases by a polynomial factor $|\mathcal{X}| / |\mathcal{Y}| = 1/\mathrm{poly}(\lambda)$. Therefore, any constrained IPF that admits a constraint that can be used to evaluate the IPF on a non-negligible fraction of the domain must necessarily have a range that is larger than the domain by at least a super-polynomial factor. Concretely, a puncturable IPF must have a range that is super-polynomially larger than the domain.

Remark 3.20 (Weaker Security Relations)**.** The lower bound in Theorem 3.16 only applies when we require that the IPF value at a constrained point appear 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 constrained point to be *unpredictable* rather than pseudorandom (given the constrained key). In other words, no efficient adversary should be able to predict $\mathsf{F}(k, x)$ 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 private 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 private 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 $\mathsf{F}_1 \colon \mathcal{K}_1 \times \mathcal{X} \to \mathcal{V}$ be an injective puncturable PRF with key-space $\mathcal{K}_1$ and range $\mathcal{V}$. Let $\mathsf{F}_2 \colon \mathcal{K}_2 \times \mathcal{V} \to \mathcal{X}$ be a private puncturable PRF with key-space $\mathcal{K}_2$. The puncturable IPF $\mathsf{F} \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ 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

$$\mathsf{F}\big((k^{(1)}, k^{(2)}),\ x\big) := \Big(\mathsf{F}_1(k^{(1)}, x),\quad x \oplus \mathsf{F}_2(k^{(2)}, \mathsf{F}_1(k^{(1)}, x)\Big).$$

- 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 $\mathsf{F}^{-1}(k, y)$ first computes $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$ and outputs

$$\mathsf{F}^{-1}(k, (y_1, y_2)) := \begin{cases} x & \text{if } y_1 = \mathsf{F}_1(k^{(1)}, x) \\ \perp & \text{otherwise.} \end{cases}$$

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

- $\mathsf{F.Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples two puncturable PRF keys $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$ and $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$. The setup algorithm outputs the IPF key $k = (k^{(1)}, k^{(2)})$.
- $\mathsf{F.Puncture}(k, x^*)$: 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 $v^* \leftarrow \mathsf{F}_1(k^{(1)}, x^*)$. It then generates two punctured keys $k^{(1)}_{x^*} \leftarrow \mathsf{F}_1.\mathsf{Puncture}(k^{(1)}, x^*)$ and $k^{(2)}_{v^*} \leftarrow \mathsf{F}_2.\mathsf{Puncture}(k^{(2)}, v^*)$ and returns $k_{x^*} = \big(k^{(1)}_{x^*}, k^{(2)}_{v^*}\big)$.
- $\mathsf{F.Eval}(k_{x^*}, x)$: On input the punctured key $k_{x^*} = (k^{(1)}_{x^*}, k^{(2)}_{v^*})$ and a point $x \in \mathcal{X}$, the evaluation algorithm first computes $y_1 \leftarrow \mathsf{F}_1.\mathsf{Eval}(k^{(1)}_{x^*}, x)$ and returns $y = (y_1, \mathsf{F}_2.\mathsf{Eval}(k^{(2)}_{v^*}, y_1) \oplus x)$.
- $\mathsf{F.Eval}^{-1}(k_{x^*}, y)$: On input the punctured key $k_{x^*} = (k^{(1)}_{x^*}, k^{(2)}_{v^*})$, and $y = (y_1, y_2) \in \mathcal{V} \times \mathcal{X} = \mathcal{Y}$, the inversion algorithm begins by computing the quantity $x \leftarrow \mathsf{F}_2.\mathsf{Eval}(k^{(2)}_{v^*}, y_1) \oplus y_2$. It returns x if $\mathsf{F}_1.\mathsf{Eval}(k^{(1)}_{x^*}, x) = y_1$ 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* $(\mathsf{F}, \mathsf{F}^{-1})$ *from Construction 4.1 is correct.*

Theorem 4.3. *Suppose* F_1 *is a selectively-secure puncturable PRF,* F_2 *is a selectively-secure, private puncturable PRF, and* $|\mathcal{X}| \,/\, |\mathcal{V}| = \mathrm{negl}(\lambda)$. *Then* $(\mathsf{F}, \mathsf{F}^{-1})$ *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 (private) 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$^+$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 NC^1 (resp., $\mathsf{P/poly}$) relies on several primitives:

- Let $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ be a PKE scheme with message space $\mathcal{X}$, ciphertext space $\mathcal{T}$, and whose decryption function can be computed in NC^1 (resp., $\mathsf{P/poly}$). 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 $\mathsf{F}_1 : \mathcal{K}_1 \times \mathcal{X} \to \mathcal{V}$ be a circuit-constrained PRF for NC^1 (resp., $\mathsf{P/poly}$).
- Let $\mathsf{F}_2 : \mathcal{K}_2 \times \mathcal{T} \to \mathcal{X}$ be a private circuit-constrained PRF for NC^1 (resp., $\mathsf{P/poly}$).[3]

The constrained IPF $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ 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 $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{PK} \times \mathcal{SK}$ for the public-key encryption scheme PKE.
- On input a key $k = (k^{(1)}, k^{(2)}, \mathsf{pk}, \mathsf{sk}) \in \mathcal{K}$, and $x \in \mathcal{X}$, the IPF $\mathsf{F}(k, x)$ computes randomness $r_x \leftarrow \mathsf{F}_1(k^{(1)}, x)$, a ciphertext $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$, and outputs

$$\mathsf{F}(k, x) := \Big(\mathsf{ct},\ \ \mathsf{F}_2(k^{(2)}, \mathsf{ct}) \oplus x\Big).$$

Note that the public key pk can also be included as part of the public parameters for the IPF.

[3]To simplify the presentation, we implicitly assume that the PRFs F_1 and F_2 support general circuit constraints (i.e., NC^1 constraints or $\mathsf{P/poly}$ 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 $k = (k^{(1)}, k^{(2)}, \mathsf{pk}, \mathsf{sk}) \in \mathcal{K}$, and $(y_1, y_2) \in \mathcal{Y}$, the inversion function $\mathsf{F}^{-1}(k, (y_1, y_2))$ first computes $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$ and $r_x \leftarrow \mathsf{F}_1(k^{(1)}, x)$. Finally, it outputs

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

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

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

- $\mathsf{F.Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples two PRF keys $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$, $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$, and a public/secret key-pair for the PKE scheme: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$. It outputs the IPF key $k = (k^{(1)}, k^{(2)}, \mathsf{pk}, \mathsf{sk})$.

- $\mathsf{F.Constrain}(k, f)$: On input the IPF key $k = (k^{(1)}, k^{(2)}, \mathsf{pk}, \mathsf{sk})$ and a constraint function $f \in \mathcal{F}$, the algorithm first constrains $k_f^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Constrain}(k^{(1)}, f)$. Then, it defines the function $F_{\mathsf{sk},f} : \mathcal{T} \to \{0, 1\}$ as follows:

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

  The constrain algorithm constrains the key $k^{(2)}$ to $F_{\mathsf{sk},f}$ and obtains $k_F^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(2)}, F_{\mathsf{sk},f})$. It then defines and returns the constrained key $k_f = (k_f^{(1)}, k_F^{(2)}, \mathsf{pk})$. Note that if $\mathsf{PKE.Decrypt}(\mathsf{sk}, \cdot)$ can be computed in NC^1 (resp., $\mathsf{P/poly}$), then the function $F_{\mathsf{sk},f}$ can also be computed in NC^1 (resp., $\mathsf{P/poly}$).

- $\mathsf{F.Eval}(k_f, x)$: On input the constrained key $k_f = (k_f^{(1)}, k_F^{(2)}, \mathsf{pk})$, and a point $x \in \mathcal{X}$, the algorithm first computes $r_x \leftarrow \mathsf{F}_1.\mathsf{Eval}(k_f^{(1)}, x)$. Then, it encrypts $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$ and returns the tuple $y = \Big(\mathsf{ct},\ \mathsf{F}_2.\mathsf{Eval}(k_F^{(2)}, \mathsf{ct}) \oplus x\Big)$.

- $\mathsf{F.Eval}^{-1}(k_f, y)$: On input the constrained key $k_f = (k_f^{(1)}, k_F^{(2)}, \mathsf{pk})$, and a point $y = (y_1, y_2) \in \mathcal{Y}$, the algorithm first computes $x \leftarrow \mathsf{F}_2.\mathsf{Eval}(k_F^{(2)}, y_1) \oplus y_2$. Then, it computes $r_x \leftarrow \mathsf{F}_1.\mathsf{Eval}(k_f^{(1)}, x)$ and $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$. If $y_1 = \mathsf{ct}$, 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 public-key encryption scheme, and* F_1, F_2 *are circuit-constrained PRFs for* NC^1 *(resp.,* $\mathsf{P/poly}$*). Then, the IPF* $(\mathsf{F}, \mathsf{F}^{-1})$ *from Construction 4.5 is a circuit-constrained IPF for* NC^1 *(resp.,* $\mathsf{P/poly}$*).*

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

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).

Remark 4.9 (Computational Notion of Smoothness)**.** As stated, Theorem 4.7 imposes an additional smoothness requirement (Definition 2.2) on the underlying public-key encryption scheme. While most semantically-secure public-key encryption schemes naturally satisfy this property, a weaker notion of "computational smoothness" also suffices for Theorem 4.7. In particular, we say a public-key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Encrypt}, \mathsf{PKE.Decrypt})$ with message space $\mathcal{M}$ and ciphertext space $\mathcal{T}$ satisfies computational smoothness if for all messages $m \in \mathcal{M}$ output by an efficient adversary (on input the security parameter λ and the public key pk), and all strings $\mathsf{ct} \in \mathcal{T}$, $\Pr[\mathsf{PKE.Encrypt}(\mathsf{pk}, m) = \mathsf{ct}] = \text{negl}(\lambda)$. Clearly, if PKE is semantically secure, then PKE satisfies computational smoothness. It is straightforward to modify the proof of Theorem 4.7 to rely on the computational version of smoothness. In this case, we can use any CCA-secure public-key encryption scheme to instantiate 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 NC^1 and $\mathsf{P/poly}$), 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[4] 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 NC^1 [CC17] and $\mathsf{P/poly}$ [BTVW17], we can leverage Construction 4.5 to obtain (single-key) circuit-constrained IPFs for NC^1 and $\mathsf{P/poly}$, respectively. We give two candidate instantiations based on standard lattice assumptions:

- To construct a circuit-constrained IPF for NC^1-constraints, we require a private circuit-constrained PRF for NC^1 and a CCA-secure public-key encryption scheme with an NC^1 decryption circuit. We can instantiate the private circuit-constrained PRF for NC^1 using the construction of Canetti and Chen [CC17]. The CCA-secure encryption scheme with NC^1 decryption can

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

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 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 NC^1 constraints from standard lattice assumptions.

- To construct a circuit-constrained IPF for $\mathsf{P/poly}$, we primarily require a private constrained PRF for $\mathsf{P/poly}$. 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.

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 $\mathsf{F.Eval}(k_f, x) \neq \mathsf{F}(k, x)$, 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:

$$\mathsf{F}((k_1, k_2), x) := \mathsf{F}_1(k^{(1)}, x) \oplus \mathsf{F}_2(k^{(2)}, x),$$

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 $(k_f^{(1)}, k^{(2)})$ where $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Constrain}(k^{(1)}, f)$. The holder of the constrained key $(k_f^{(1)}, k^{(2)})$ can further constrain to a function of the form $f \wedge g$ by computing $(k_f^{(1)}, k_g^{(2)})$ where $k_g^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(2)}, g)$. 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 $k^{(\mathsf{IPF})}$ for IPF, and the public/secret key-pair pk, sk for the PKE scheme. Our delegatable IPF works as follows:

```
$\mathsf{F}\big((k^{(1)}, k^{(2)}, k^{(3)}, k^{(\mathsf{IPF})}, \mathsf{pk}, \mathsf{sk}),\ x\big) :=$
    $r \leftarrow \mathsf{F}_1(k^{(1)}, x) \oplus \mathsf{F}_3(k^{(3)}, x)$
    $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r)$
    $z \leftarrow \mathsf{F}_2(k^{(2)}, \mathsf{ct}) \oplus \mathsf{IPF}(k^{(\mathsf{IPF})}, x)$
    output $(\mathsf{ct}, z)$
```

```
$\mathsf{F}^{-1}\big((k^{(1)}, k^{(2)}, k^{(3)}, k^{(\mathsf{IPF})}, \mathsf{pk}, \mathsf{sk}),\ (\mathsf{ct}, z)\big) :=$
    $x \leftarrow \mathsf{IPF}^{-1}(k^{(\mathsf{IPF})}, z \oplus \mathsf{F}_2(k^{(2)}, \mathsf{ct}))$
    $r \leftarrow \mathsf{F}_1(k^{(1)}, x) \oplus \mathsf{F}_3(k^{(3)}, x)$
    if $\mathsf{ct} \neq \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r)$
      then $x \leftarrow \perp$
    output $x$
```

To constrain a key $(k^{(1)}, k^{(2)}, k^{(3)}, k^{(\mathsf{IPF})}, \mathsf{pk}, \mathsf{sk})$ 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 $k_f^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Constrain}(k^{(1)}, f)$ and $k_F^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(2)}, F_{\mathsf{sk},f})$, where $F_{\mathsf{sk},f}$ is defined as in Eq. (4.1). The constrained key is the tuple $k_f = (k_f^{(1)}, k_F^{(2)}, k^{(3)}, k^{(\mathsf{IPF})}, \mathsf{pk})$. To further constrain (that is, delegate) to a function g, we constrain F_3 and IPF to g. In other words, we compute $k_g^{(3)} \leftarrow \mathsf{F}_3.\mathsf{Constrain}(k^{(3)}, g)$ and $k_g^{(\mathsf{IPF})} \leftarrow \mathsf{IPF}.\mathsf{Constrain}(k^{(\mathsf{IPF})}, g)$. The constrained key $k_{f \wedge g}$ for the function $f \wedge g$ is defined to be $k_{f \wedge g} := (k_f^{(1)}, k_F^{(2)}, k_g^{(3)}, k_g^{(\mathsf{IPF})}, \mathsf{pk})$. 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+01, GGH+13].

Definition 7.1 (Indistinguishability Obfuscation [BGI⁺01, GGH⁺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
$$|\Pr[\mathcal{A}(i\mathcal{O}(C_0)) = 1] - \Pr[\mathcal{A}(i\mathcal{O}(C_1)) = 1]| = \mathrm{negl}(\lambda).$$

Construction 7.2. Fix a domain $\mathcal{X} = \{0,1\}^n$ where $n = n(\lambda)$. Let $\mathsf{F}_1 \colon \mathcal{K}_1 \times \mathcal{X} \to \mathcal{V}$ be a puncturable PRF with key-space $\mathcal{K}_1$ and range $\mathcal{V}$. Let $\mathsf{F}_2 \colon \mathcal{K}_2 \times \mathcal{V} \to \mathcal{X}$ be a puncturable PRF with key-space $\mathcal{K}_2$. The constrained IPF $\mathsf{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ 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
$$\mathsf{F}(k, x) := \left(\mathsf{F}_1(k^{(1)}, x),\ \ \mathsf{F}_2(k^{(2)}, \mathsf{F}_1(k^{(1)}, x)) \oplus x\right).$$
- 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 $\mathsf{F}^{-1}(k, y)$ first computes $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$ and outputs
$$\mathsf{F}^{-1}(k, (y_1, y_2)) := \begin{cases} x & \text{if } y_1 = \mathsf{F}_1(k^{(1)}, x) \\ \perp & \text{otherwise.} \end{cases}$$

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

- $\mathsf{F.Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples two puncturable PRF keys $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$ and $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$, and outputs $k = (k^{(1)}, k^{(2)})$.
- $\mathsf{F.Constrain}(k, f)$: 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 $P_0 = i\mathcal{O}(P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}])$ and $P_1 = i\mathcal{O}(P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}])$ where the programs $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}]$ and $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}]$ 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 $\mathsf{F}(k, x) = \left(\mathsf{F}_1(k^{(1)}, x), \mathsf{F}_2(k^{(2)}, \mathsf{F}_1(k^{(1)}, x)) \oplus x\right)$.

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

```
Constants: a function $f \in \mathcal{F}$, and two keys $k^{(1)}$ and $k^{(2)}$ for $\mathsf{F}_1$ and $\mathsf{F}_2$, respectively.

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

  1. Compute $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$.
  2. If $f(x) = 0$ or $y_1 \neq \mathsf{F}_1(k^{(1)}, x)$, output $\perp$.
  3. Otherwise, output $x$.
```

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

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

- $\mathsf{F.Eval}(k_f, x)$: On input the constrained key $k_f = (P_1, P_2)$, and a point $x \in \mathcal{X}$, the evaluation algorithm outputs $P_1(x)$.
- $\mathsf{F.Eval}^{-1}(k_f, y)$: 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 $(\mathsf{F}, \mathsf{F}^{-1})$ 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 $|\mathcal{X}| \,/\, |\mathcal{V}| = \text{negl}(\lambda)$. Then $(\mathsf{F}, \mathsf{F}^{-1})$ from Construction 7.2 is a selectively-secure circuit-constrained IPF.*

Acknowledgments

We thank the anonymous TCC reviewers for helpful comments on this work. This work was funded by NSF, the DARPA/ARL SAFEWARE project, 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 ONeill. 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 ONeill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In *CRYPTO*, 2008.

[BFOR08] Mihir Bellare, Marc Fischlin, Adam ONeill, and Thomas Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *CRYPTO*, 2008.

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

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.

[BHK15] Mihir Bellare, Dennis Hofheinz, and Eike Kiltz. Subtleties in the definition of IND-CCA: when and how should challenge decryption be disallowed? *J. Cryptology*, 28(1), 2015.

[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. Private constrained PRFs (and more) from LWE. In *TCC*, 2017.

[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 Oneill, 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$^+$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 $k = (k^{(1)}, k^{(2)}) \leftarrow \mathsf{F.Setup}(1^\lambda)$. Take any $x^* \in \mathcal{X}$, and let $k_{x^*} = (k^{(1)}_{x^*}, k^{(2)}_{v^*}) \leftarrow \mathsf{F.Puncture}(k, x^*)$. We show the two correctness requirements separately:

- Take any $x \neq x^*$. Suppose $\mathsf{F}(k, x) = (y_1, y_2)$, which means that $y_1 = \mathsf{F}_1(k^{(1)}, x)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x$. Since F_1 is injective and $x \neq x^*$, $\mathsf{F}_1(k^{(1)}, x) \neq \mathsf{F}_1(k^{(1)}, x^*) = v^*$. By correctness of F_1 and F_2, we have that $\mathsf{F}_1.\mathsf{Eval}(k^{(1)}_{x^*}, x) = \mathsf{F}_1(k^{(1)}, x) = y_1$ and $\mathsf{F}_2.\mathsf{Eval}(k^{(2)}_{v^*}, y_1) \oplus x = \mathsf{F}_2(k^{(2)}, y_1) \oplus x = y_2$.
- Take any input $(y_1, y_2) \in \mathcal{V} \times \mathcal{X}$ where there exists some $x \neq x^*$ such that $\mathsf{F}(k, x) = (y_1, y_2)$. This means that $y_1 = \mathsf{F}_1(k^{(1)}, x)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x$. Moreover, since F_1 is injective and $x \neq x^*$, $y_1 \neq \mathsf{F}_1(k^{(1)}, x^*) = v^*$. By correctness of F_2, it follows that $\mathsf{F}_2.\mathsf{Eval}(k^{(2)}_{v^*}, y_1) \oplus y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x$. By correctness of F_1, $\mathsf{F}_1.\mathsf{Eval}(k^{(1)}_{x^*}, x) = \mathsf{F}_1(k^{(1)}, x) = y_1$, in which case $\mathsf{F}.\mathsf{Eval}^{-1}(k_{x^*}, (y_1, y_2)) = x = \mathsf{F}^{-1}(k, (y_1, y_2))$. □

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_0: This is the constrained IPF security experiment $\mathsf{Expt}^{(\mathsf{IPF})}_{\mathcal{A},\mathsf{F}}(\lambda, 0)$ from Definition 3.11. In the setup phase, after the adversary commits to a point $x^* \in \mathcal{X}$, the challenger samples keys $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$ and $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$, and sets $k = (k^{(1)}, k^{(2)})$. It sets $v^* \leftarrow \mathsf{F}_1(k^{(1)}, x^*)$ and $z^* \leftarrow \mathsf{F}_2(k^{(2)}, v^*) \oplus x^*$ and constructs the constrained keys $k^{(1)}_{x^*} \leftarrow \mathsf{F}_1.\mathsf{Puncture}(k^{(1)}, x^*)$ and $k^{(2)}_{v^*} \leftarrow \mathsf{F}_2.\mathsf{Puncture}(k^{(2)}, v^*)$. The challenger gives the punctured key $k_{x^*} = (k^{(1)}_{x^*}, k^{(2)}_{v^*})$ and evaluation $y^* = (v^*, z^*)$ to the adversary. During the query phase, the challenger answers all of the evaluation queries by computing $\mathsf{F}(k, \cdot)$ and the inversion queries by computing $\mathsf{F}^{-1}(k, \cdot)$, exactly as in the real scheme.
- Hyb_1: Same as Hyb_0, except during the setup phase, the challenger samples $v^* \xleftarrow{\text{R}} \mathcal{V}$. The rest of the setup phase proceeds as in Hyb_0. When responding to the evaluation and inversion queries, the challenger uses v^* in place of the value $\mathsf{F}_1(k^{(1)}, x^*)$.

- Hyb_2: Same as Hyb_1, except when responding to the evaluation and inversion queries, the challenger always evaluates $\mathsf{F}_1(k^{(1)}, \cdot)$ instead of substituting the value v^* for $\mathsf{F}_1(k^{(1)}, x^*)$.

- Hyb_3: Same as Hyb_2, except during the setup phase, the challenger samples $z^* \xleftarrow{\mathrm{R}} \mathcal{X}$. The rest of the setup phase proceeds as in Hyb_2. When responding to the evaluation and inversion queries, the challenger uses $z^* \oplus x^*$ in place of the value $\mathsf{F}_2(k^{(2)}, v^*)$.

- Hyb_4: Same as Hyb_3, except when responding to the evaluation and inversion queries, the challenger always evaluates $\mathsf{F}_2(k^{(2)}, \cdot)$ instead of substituting the value $z^* \oplus x^*$ for $\mathsf{F}_2(k^{(2)}, v^*)$.

- Hyb_5: Same as Hyb_4, except during the setup phase, the challenger punctures $k^{(2)}$ at $\mathsf{F}_1(k^{(1)}, x^*)$ instead of v^*. This is the constrained IPF security experiment $\mathsf{Expt}^{(\mathsf{IPF})}_{\mathcal{A},\mathsf{F}}(\lambda, 1)$ from Definition 3.6.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output in hybrid Hyb_i. 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 $\mathcal{A}$,* $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathrm{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between hybrids Hyb_0 and 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 $k^{(1)}_{x^*}$ and a challenge value $v^* \in \mathcal{V}$. It generates the key $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$, the punctured key $k^{(2)}_{v^*} \leftarrow \mathsf{F}_2.\mathsf{Puncture}(k^{(2)}, v^*)$, and the evaluation $z^* \leftarrow \mathsf{F}_2(k^{(2)}, v^*) \oplus x^*$ exactly as in the real scheme. Finally, it sends the punctured key $k_{x^*} = (k^{(1)}_{x^*}, k^{(2)}_{v^*})$ and the challenge evaluation $y^* = (v^*, z^*)$ to $\mathcal{A}$.

- **Query phase.** Algorithm $\mathcal{B}$ simulates the query phase exactly as in Hyb_0 and Hyb_1, except whenever it needs to compute $\mathsf{F}_1(k^{(1)}, x)$ on some $x \neq x^*$, it queries the evaluation oracle for F_1 on x. Whenever it needs to compute $\mathsf{F}_1(k^{(1)}, x^*)$, 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 Hyb_0, and if v^* is a uniformly random value, then it has perfectly simulated 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 $v^* = \mathsf{F}_1(k^{(1)}, x^*)$. Then, the setup phase is simulated exactly as in 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 $\mathsf{F}_1(k^{(1)}_{x^*}, x)$ whenever $x \neq x^*$, and it uses $v^* = \mathsf{F}_1(k^{(1)}, x^*)$ for the value of $\mathsf{F}_1(k^{(1)}, x^*)$. Thus, $\mathcal{B}$ perfectly simulates Hyb_0.

- Suppose v^* is uniformly random over $\mathcal{V}$. In this case, $\mathcal{B}$ perfectly simulates the setup phase of Hyb_1. In the query phase, algorithm $\mathcal{B}$ uses the evaluation oracle to compute $\mathsf{F}_1(k^{(1)}, x)$ whenever $x \neq x^*$, and the value v^* for $\mathsf{F}_1(k^{(1)}, x^*)$. This is precisely the distribution in Hyb_1.

We conclude that $\mathcal{B}$ breaks the puncturing security of F_1 with the same advantage adversary $\mathcal{A}$ has in distinguishing Hyb_0 and 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 $\mathcal{A}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \text{negl}(\lambda)$.*

Proof. The only difference between Hyb_1 and Hyb_2 is that during the query phase, the challenger uses the value v^* in place of the value $\mathsf{F}_1(k^{(1)}, x^*)$ in 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 Hyb_1 and Hyb_2.

- **Evaluation queries.** On an evaluation query x, the challenger in Hyb_1 and Hyb_2 only needs to evaluate $\mathsf{F}_1(k^{(1)}, \cdot)$ on x. Since Hyb_1 and Hyb_2 only differ in the value the challenger uses for $\mathsf{F}_1(k^{(1)}, x^*)$, the outputs of the evaluation oracle in Hyb_1 and 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 Hyb_1 and Hyb_2 are distributed identically.

- **Inversion queries.** On an inversion query (y_1, y_2), the challenger first computes $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$. Then, the challenger checks whether $y_1 = \mathsf{F}_1(k^{(1)}, x)$. If $x \neq x^*$, then the output of the inversion oracle in Hyb_1 and Hyb_2 is computed identically. It suffices to only consider the case where $x = x^*$, or equivalently, when $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$.

  - In Hyb_1, the challenger substitutes the value v^* for the value $\mathsf{F}_1(k^{(1)}, x^*)$. For any query (y_1, y_2) where $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$, the challenger in Hyb_1 always responds with $\perp$ unless $y_1 = v^*$. But if $y_1 = v^*$, then it must be the case that $y_2 = \mathsf{F}_2(k^{(2)}, v^*) \oplus x^* = z^*$. This means that $(y_1, y_2) = (v^*, z^*)$, which is the challenge query. Thus, the challenger in Hyb_1 responds with $\perp$ on all admissible inversion queries satisfying $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$.
  - In Hyb_2, on a query (y_1, y_2) where $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$, the challenger in Hyb_2 responds with $\perp$ unless $y_1 = \mathsf{F}_1(k^{(1)}, x^*)$. This in particular means that $y_2 = \mathsf{F}_2(k^{(2)}, \mathsf{F}_1(k^{(1)}, x^*)) \oplus x^*$.

  We conclude that the response of the challenger on an inversion query is identically distributed in Hyb_1 and Hyb_2 unless the adversary makes an inversion query on the tuple (y_1, y_2) where $y_1 = \mathsf{F}_1(k^{(1)}, x^*)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$.

Suppose now that an adversary $\mathcal{A}$ is able to distinguish between Hyb_1 and Hyb_2 with some non-negligible probability ε. By the above analysis, the adversary's view in Hyb_1 and Hyb_2 is *identically* distributed unless the adversary makes an inversion query on the input (y_1, y_2) where $y_1 = \mathsf{F}_1(k^{(1)}, x^*)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$. Since $\mathcal{A}$ distinguishes Hyb_1 from Hyb_2 with advantage ε, it must make an inversion query on (y_1, y_2) with probability at least ε. 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 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 $v^* \xleftarrow{\text{R}} \mathcal{V}$. It generates the key $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$, the punctured key $k_{v^*}^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Puncture}(k^{(2)}, v^*)$, and the evaluation $z^* \leftarrow \mathsf{F}_2(k^{(2)}, v^*) \oplus x^*$ exactly as in Hyb_1 and 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 Hyb_1 and Hyb_2. In particular, whenever it needs to compute $x = \mathsf{F}_1(k^{(1)}, x)$ 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 $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$. If $x \neq x^*$, then $\mathcal{B}$ proceeds as in Hyb_1 and Hyb_2, except it queries the evaluation oracle for F_1 to compute the value of $\mathsf{F}_1(k^{(1)}, x)$. On (admissible) inversion queries (y_1, y_2) where $x = \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$, 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 $(\hat{v}^*, \mathsf{F}_2(k^{(2)}, \hat{v}^*) \oplus x^*)$ 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 Hyb_1 for $\mathcal{A}$. This means that with probability at least ε, $\mathcal{A}$ will submit an inversion query on the tuple (y_1, y_2) where $y_1 = \mathsf{F}_1(k^{(1)}, x^*)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$ at some point during the challenge phase (otherwise, the view of $\mathcal{A}$ is identically distributed in Hyb_1 and Hyb_2). We now consider the probability that $\mathcal{B}$ outputs 1.

- Suppose $\hat{v}^* = \mathsf{F}_1(k^{(1)}, x^*)$. By assumption, $\mathcal{A}$ queries the inversion oracle on the tuple (y_1, y_2) where $y_1 = \mathsf{F}_1(k^{(1)}, x^*) = \hat{v}^*$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^* = \mathsf{F}_2(k^{(2)}, \hat{v}^*) \oplus x^*$ with probability at least ε. Correspondingly, algorithm $\mathcal{B}$ outputs 1 with probability at least ε.

- 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 Hyb_1 and Hyb_2 with non-negligible advantage ε, then $\mathcal{B}$ is able to break puncturing security of F_1 with probability at least $\varepsilon - \text{negl}(\lambda)$. ☐

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

Proof. This follows very similarly to the proof of Lemma A.1. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between hybrids Hyb_2 and 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 $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$, $k_{x^*}^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Puncture}(k^{(1)}, x^*)$, and chooses $v^* \xleftarrow{\text{R}} \mathcal{V}$. 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 Hyb_2 and Hyb_3, except whenever it needs to compute $\mathsf{F}_2(k^{(2)}, v)$ on some $v \neq v^*$, it queries the evaluation oracle for F_2 on v. Whenever it needs to compute $\mathsf{F}_2(k^{(2)}, v^*)$, 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 Hyb_2 or Hyb_3 for $\mathcal{A}$:

- Suppose $z^* = \mathsf{F}_2(k^{(2)}, v^*)$. Then, the setup and challenge phases are simulated as described in 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 Hyb_3. By construction, $\mathcal{B}$ perfectly simulates the challenge phase in Hyb_2.

We conclude that $\mathcal{B}$ breaks puncturing security of $\mathcal{A}$ with the same advantage $\mathcal{A}$ has in distinguishing Hyb_2 and Hyb_3. The lemma follows. □

Lemma A.4. *If* $|\mathcal{X}| / |\mathcal{V}| = \text{negl}(\lambda)$*, then for all adversaries* $\mathcal{A}$*,*

$$|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| = \text{negl}(\lambda).$$

Proof. The only difference between Hyb_3 and Hyb_4 is that during the query phase, the challenger uses the value $z^* \oplus x^*$ in place of $\mathsf{F}_2(k^{(2)}, v^*)$ in 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 Hyb_3 and Hyb_4.

- **Evaluation queries.** On an evaluation query x, the challenger in Hyb_3 and Hyb_4 first computes $v \leftarrow \mathsf{F}_1(k^{(1)}, x)$ and then computes $\mathsf{F}_2(k^{(2)}, v)$. Thus, the response to the evaluation query in Hyb_3 and Hyb_4 differs only on points $x \in \mathcal{X}$ where $\mathsf{F}_1(k^{(1)}, x) = v^*$. But in Hyb_3 and 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 $\Pr[v^* = \mathsf{F}_1(k^{(1)}, x)] = 1/|\mathcal{V}|$. Taking an union bound over all $x \in \mathcal{X}$, we have that

  $$\Pr[\forall x \colon \mathsf{F}_1(k^{(1)}, x) \neq v^*] = 1 - |\mathcal{X}| / |\mathcal{V}| = 1 - \text{negl}(\lambda).$$

  Thus, with overwhelming probability (over the choice of v^*), $\mathsf{F}_1(k^{(1)}, x) \neq v^*$ for all $x \in \mathcal{X}$. This means that with overwhelming probability, the outputs of the evaluation oracle in Hyb_3 is the same as that in Hyb_4 on all inputs.

- **Inversion queries.** On an inversion query (y_1, y_2), the challenger in Hyb_3 and Hyb_4 first computes $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$ and then checks whether $y_1 = \mathsf{F}_1(k^{(1)}, x)$. On all queries where $y_1 \neq v^*$, the challenger's behavior in Hyb_3 and Hyb_4 is identical. On the queries (v^*, y_2), the only instance where the challenger does not output $\perp$ is if $\mathsf{F}_1(k^{(1)}, x) = v^*$. 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 $\mathsf{F}_2(k^{(2)}, v^*)$, the output in Hyb_3 and 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 Hyb_3 is the same as that in 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 Hyb_3 and 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 $\mathcal{A}$, $|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]| = \text{negl}(\lambda)$.*

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:

- $\mathsf{Hyb}_{4,1}$: Same as Hyb_4 except the challenger uses the simulator $\mathcal{S}_1$ to construct the punctured key $k^{(2)}_{v^*}$ and the simulator $\mathcal{S}_2$ in place of $\mathsf{F}_2(k^{(2)}, \cdot)$ in the challenge phase. More precisely, in the setup phase, the challenger computes $(k^{(2)}_{v^*}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$. During the query phase, the challenger answers all queries as in Hyb_4, except whenever it needs to evaluate $\mathsf{F}_2(k^{(2)}, \cdot)$ on $v \in \mathcal{V}$, the challenger instead computes $(x, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(v, 1, \mathsf{st}_{\mathcal{S}})$, and uses x in place of $\mathsf{F}_2(k^{(2)}, v)$.

- $\mathsf{Hyb}_{4,2}$: Same as $\mathsf{Hyb}_{4,1}$ except during the query phase, whenever it needs to evaluate $\mathsf{F}_2(k^{(2)}, \cdot)$ on $v = \mathsf{F}_1(k^{(1)}, x^*)$, the challenger instead computes $(x, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(v, 0, \mathsf{st}_{\mathcal{S}})$, and uses x in place of $\mathsf{F}_2(k^{(2)}, v)$. All other queries to $\mathsf{F}_2(k^{(2)}, \cdot)$ are handled as in $\mathsf{Hyb}_{4,1}$.

We now show that each pair of consecutive hybrids Hyb_4, $\mathsf{Hyb}_{4,1}$, $\mathsf{Hyb}_{4,2}$, and 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 $\left|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{4,1}(\mathcal{A}) = 1]\right| = \text{negl}(\lambda)$.*

Proof. Suppose that there exists an adversary $\mathcal{A}$ that can distinguish between Hyb_4 and $\mathsf{Hyb}_{4,1}$. We use $\mathcal{A}$ to build an adversary $\mathcal{B}$ that can distinguish between $\mathsf{Real}_{\mathcal{B},\mathsf{F}_2}(\lambda)$ and $\mathsf{Ideal}_{\mathcal{B},\mathcal{S},\mathsf{F}_2}(\lambda)$ 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 $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$, $k^{(1)}_{x^*} \leftarrow \mathsf{F}_1.\mathsf{Puncture}(k^{(1)}, x^*)$ and chooses $v^* \xleftarrow{\text{R}} \mathcal{V}$. It commits to v^* in the privacy game and receives a key $k^{(2)}_{v^*}$. Next, $\mathcal{B}$ chooses $z^* \xleftarrow{\text{R}} \mathcal{X}$ and gives the key $k_{x^*} = (k^{(1)}_{x^*}, k^{(2)}_{v^*})$ and the challenge value $y^* = (x^*, z^*)$ to $\mathcal{A}$.

- **Query phase.** Algorithm $\mathcal{B}$ simulates the evaluation queries exactly as in Hyb_4 and $\mathsf{Hyb}_{4,1}$, except whenever it needs to compute $\mathsf{F}_2(k^{(2)}, v)$ 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 Hyb_4 and $\mathsf{Hyb}_{4,1}$, except it queries its evaluation oracle for F_2 whenever it needs to compute $\mathsf{F}_2(k^{(2)}, v)$. 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 $\mathsf{Real}_{\mathcal{B},\mathsf{F}_2}$ and $\mathsf{Ideal}_{\mathcal{B},\mathcal{S},\mathsf{F}_2}$ separately.

- If the key $k^{(2)}_{v^*}$ and PRF evaluations are computed according to $\mathsf{Real}_{\mathcal{B},\mathsf{F}_2}$, then the setup and evaluation queries are simulated exactly as in 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 Hyb_4

outputs something other than $\perp$ only if there exists some $x \in \mathcal{X}$ where $\mathsf{F}_1(k^{(1)}, x) = v^*$. 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 Hyb_4.

- If the key $k_{v^*}^{(2)}$ and PRF evaluations are computed according to $\mathsf{Ideal}_{\mathcal{B},\mathcal{S},\mathsf{F}_2}$, then we claim that $\mathcal{B}$ correctly simulates $\mathsf{Hyb}_{4,1}$ 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 $\mathsf{F}_2(k^{(2)}, v^*)$. If algorithm $\mathcal{B}$ needs to query F_2 on v^*, then in $\mathsf{Ideal}_{\mathcal{B},\mathcal{S},\mathsf{F}_2}$, the response is computed using $\mathcal{S}_2(v^*, 0, \mathsf{st}_{\mathcal{S}})$ whereas in $\mathsf{Hyb}_{4,1}$, the response is computed using $\mathcal{S}_2(v^*, 1, \mathsf{st}_{\mathcal{S}})$. 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 $v \leftarrow \mathsf{F}_1(k^{(1)}, x)$ 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 $v^* = \mathsf{F}_1(k^{(1)}, x)$, 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 $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$. 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 $\mathsf{Hyb}_{4,1}$. 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 $\mathsf{Hyb}_{4,1}$. For the inversion queries of the form (v^*, y_2), the challenger's response in $\mathsf{Hyb}_{4,1}$ 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 $\mathsf{Hyb}_{4,1}$.

By puncturing security of F_2, hybrids Hyb_4 and $\mathsf{Hyb}_{4,1}$ are computationally indistinguishable. $\square$

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

Proof. By construction, $\mathsf{Hyb}_{4,1}$ and $\mathsf{Hyb}_{4,2}$ only differ on how the challenger handles the evaluation and inversion queries that require computing $\mathsf{F}_2(k^{(2)}, \mathsf{F}_1(k^{(1)}, x^*))$. There are two possibilities:

- The adversary makes an (admissible) evaluation query on $x \neq x^*$ where $\mathsf{F}_1(k^{(1)}, x) = \mathsf{F}_1(k^{(1)}, x^*)$.
- The adversary makes an inversion query on (y_1, y_2) where $y_1 = \mathsf{F}_1(k^{(1)}, x^*)$.

Suppose there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{Hyb}_{4,1}$ and $\mathsf{Hyb}_{4,2}$ with non-negligible probability ε. This means that with probability at least ε, adversary $\mathcal{A}$ makes a query satisfying one of the above two conditions in $\mathsf{Hyb}_{4,1}$. 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 $y^* \xleftarrow{\text{R}} \mathcal{V} \times \mathcal{X}$, computes $(k_{v^*}^{(2)}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$, 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 $(z, \mathsf{st}_\mathcal{S}) \leftarrow \mathcal{S}_2(v, 1, \mathsf{st}_\mathcal{S})$, and returns $(v, z \oplus x)$, exactly as in $\mathsf{Hyb}_{4,1}$. 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 $(x', \mathsf{st}_\mathcal{S}) \leftarrow \mathcal{S}_2(y_1, 1, \mathsf{st}_\mathcal{S})$ 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 $\mathsf{Hyb}_{4,1}$. 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 if set. Note that the "good" events precisely correspond to the types of queries $\mathcal{A}$ makes with probability ε in $\mathsf{Hyb}_{4,1}$. We now compute the probability that $\mathcal{B}$ outputs 1 in the pseudorandom setting and the truly random setting:

- If $\hat{v}^* = \mathsf{F}_1(k^{(1)}, x^*)$, we show that the Good flag is set with probability at least ε. There are two cases:
  - Suppose $\mathcal{A}$ queries the inversion oracle on the point (y_1, y_2) where $y_1 = \mathsf{F}_1(k^{(1)}, x^*)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$. 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 $y_1 = \mathsf{F}_1(k^{(1)}, x^*)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$. Then, by construction, $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in $\mathsf{Hyb}_{4,1}$. In this case, with probability ε, $\mathcal{A}$ makes a query in the query phase that causes algorithm $\mathcal{B}$ to set the Good flag.

  We conclude that with probability at most ε, $\mathcal{B}$ outputs 1 when $\hat{v}^* = \mathsf{F}_1(k^{(1)}, x^*)$.
- If $\hat{v}^*$ is uniformly random over $\mathcal{V}$, then adversary $\mathcal{B}$ outputs 1 with probability at most $Q/\left|\mathcal{V}\right| = \text{negl}(\lambda)$, 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 ε is the advantage $\mathcal{A}$ has in distinguishing hybrids $\mathsf{Hyb}_{4,1}$ from $\mathsf{Hyb}_{4,2}$. The claim follows. □

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

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 $\mathsf{Hyb}_{4,2}$ and Hyb_5. We use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ that can distinguish between $\mathsf{Real}_{\mathcal{B},\mathsf{F}_2}(\lambda)$ and $\mathsf{Ideal}_{\mathcal{B},\mathcal{S},\mathsf{F}_2}(\lambda)$. 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 $\mathsf{F}_1(k^{(1)}, x^*)$ in the privacy game (instead of v^*).

- **Query phase.** Algorithm $\mathcal{B}$ simulates the evaluation and inversion queries exactly as in $\mathsf{Hyb}_{4,2}$ and Hyb_5, except whenever it needs to compute $\mathsf{F}_2(k^{(2)}, v)$ 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 $\mathsf{Real}_{\mathcal{B},\mathsf{F}_2}$, then the setup and evaluation queries are simulated exactly as in Hyb_5. Conversely, if the key $k_{v^*}^{(2)}$ and PRF evaluations are computed according to $\mathsf{Ideal}_{\mathcal{B},\mathcal{S},\mathsf{F}_2}$, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{4,2}$. The claim follows. $\square$

By Claims A.6 through A.8, we conclude that hybrids Hyb_4 and 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 $k = (k^{(1)}, k^{(2)}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{F.Setup}(1^\lambda)$, $k_f = (k_f^{(1)}, k_F^{(2)}, \mathsf{pk}) \leftarrow \mathsf{F.Constrain}(k, f)$. We now show the two correctness requirements separately:

- Take any $x \in \mathcal{X}$ where $f(x) = 1$. Let $(y_1, y_2) \leftarrow \mathsf{F}(k, x)$. Then, $y_1 = \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$ where $r_x = \mathsf{F}_1(k^{(1)}, x)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x$. By correctness of F_1, it follows that $r_x = \mathsf{F}_1.\mathsf{Eval}(k_f^{(1)}, x)$. Furthermore, by correctness of PKE, $\mathsf{Decrypt}(\mathsf{sk}, y_1) = x$ and so, $F_{\mathsf{sk},f}(y_1) = f(\mathsf{Decrypt}(\mathsf{sk}, y_1)) = f(x) = 1$. By correctness of F_2, $y_2 = \mathsf{F}_2.\mathsf{Eval}(k_F^{(2)}, y_1) \oplus x$.
- Take any input $(y_1, y_2) \in \mathcal{T} \times \mathcal{X}$ where there exists some x such that $f(x) = 1$ and $F(k, x) = (y_1, y_2)$. This means that $y_1 = \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$ for $r_x = \mathsf{F}_1(k^{(1)}, x)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x$. By correctness of PKE, $F_{\mathsf{sk},f}(y_1) = 1$. Then, by correctness of F_2, $\mathsf{F}_2.\mathsf{Eval}(k_F^{(2)}, y_1) \oplus y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x$. By correctness of F_1, $r_x = \mathsf{F}_1(k^{(1)}, x) = \mathsf{F}_1.\mathsf{Eval}(k_f^{(1)}, x)$. In this case, $\mathsf{F.Eval}^{-1}(k_f, (y_1, y_2)) = x = \mathsf{F}^{-1}(k, (y_1, y_2))$. $\square$

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_0: This is the constrained IPF security experiment $\mathsf{Expt}^{(\mathsf{IPF})}_{\mathcal{A},\mathsf{F}}(\lambda, 0)$ from Definition 3.11. During the setup phase, after the adversary commits to a function $f^* \in \mathcal{F}$, the challenger samples PRF keys $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$, $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$, as well as a PKE public/secret key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE}.\mathsf{Setup}(1^\lambda)$ and sets $k = (k^{(1)}, k^{(2)}, \mathsf{pk}, \mathsf{sk})$. Next, it computes $k^{(1)}_{f^*} \leftarrow \mathsf{F}_1.\mathsf{Constrain}(k^{(1)}, f^*)$, $k^{(2)}_{F^*} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(2)}, F_{\mathsf{sk},f^*})$, and gives $k_{f^*} = (k^{(1)}_{f^*}, k^{(2)}_{F^*}, \mathsf{pk})$ to the adversary. The challenger answers the evaluation and inversion queries by computing $\mathsf{F}(k, \cdot)$ and $\mathsf{F}^{-1}(k, \cdot)$, respectively. On a challenge query $x^* \in \mathcal{X}$, the challenger computes $r^* \leftarrow \mathsf{F}_1(k^{(1)}, x^*)$, $\mathsf{ct}^* \leftarrow \mathsf{PKE}.\mathsf{Encrypt}(\mathsf{pk}, x^*; r^*)$, and returns $y^* = (\mathsf{ct}^*, \mathsf{F}_2(k^{(2)}, \mathsf{ct}^*) \oplus x^*)$.
- Hyb_1: Same as Hyb_0, except whenever the adversary makes a pre-challenge inversion query $(y_1, y_2) \in \mathcal{Y}$, if $\mathsf{F}^{-1}(k, (y_1, y_2)) = x$ 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_0. 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 $x \leftarrow \mathsf{PKE}.\mathsf{Decrypt}(\mathsf{sk}, y_1)$. If $f^*(x) = 1$ or $\mathcal{A}$ has previously made an evaluation query on x, then the challenger checks to see if $y_1 = \mathsf{PKE}.\mathsf{Encrypt}(\mathsf{pk}, x; r_x)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x$, where $r_x \leftarrow \mathsf{F}_1(k^{(1)}, x)$. 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_2: Same as Hyb_1, except when responding to the challenge query during the query phase, the challenger samples $r^* \xleftarrow{\text{R}} \mathcal{V}$. Moreover, when responding to the (subsequent) evaluation and inversion queries, the challenger always uses r^* in place of the value $\mathsf{F}_1(k^{(1)}, x^*)$.
- Hyb_3: Same as Hyb_2, except when responding to the evaluation and (post-challenge) inversion queries, the challenger always evaluates $\mathsf{F}_1(k^{(1)}, \cdot)$ instead of substituting the value r^* for $\mathsf{F}_1(k^{(1)}, x^*)$.
- Hyb_4: Same as Hyb_3, except when responding to the challenge query, the challenger samples $z^* \xleftarrow{\text{R}} \mathcal{X}$ and returns (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 $\mathsf{F}_2(k^{(2)}, \mathsf{ct}^*)$.

- Hyb_5: Same as Hyb_4, except when responding to the evaluation and (post-challenge) inversion queries, the challenger always evaluates $\mathsf{F}_2(k^{(2)}, \cdot)$ instead of substituting the value $z^* \oplus x^*$ for $\mathsf{F}_2(k^{(2)}, \mathsf{ct}^*)$.

- Hyb_6: Same as Hyb_5, 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_5 except the challenge computes $(k^{(2)}_{F^*}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$ instead of setting $k^{(2)}_{F^*} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(2)}, F_{\mathsf{sk}, f^*})$.

  – **Query phase.** Same as Hyb_5 except whenever the challenger needs to evaluate $\mathsf{F}_2(k^{(2)}, \mathsf{ct})$ on some $\mathsf{ct} \in \mathcal{T}$, it instead computes $(z, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(\mathsf{ct}, F_{\mathsf{sk}, f^*}(\mathsf{ct}), \mathsf{st}_{\mathcal{S}})$ and uses z in place of $\mathsf{F}_2(k^{(2)}, \mathsf{ct})$.

- Hyb_7: Same as Hyb_6, except the challenger answers the challenge query by choosing a random value $\hat{x}^* \xleftarrow{\text{R}} \mathcal{X}$ and returning $(\mathsf{PKE.Encrypt}(\mathsf{pk}, \hat{x}^*; r^*), z^*)$.

- Hyb_8: Same as Hyb_7, except the challenger sets $k^{(2)}_{F^*} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(2)}, F_{\mathsf{sk}, f^*})$ in the setup phase, and uses the real evaluations $\mathsf{F}_2(k^{(2)}, \cdot)$ rather than the simulated ones in the query phase.

- Hyb_9: Same as Hyb_8, except the pre-challenge inversion queries are now handled as in the real scheme. This is the constrained IPF security experiment $\mathsf{Expt}^{(\mathsf{IPF})}_{\mathcal{A}, \mathsf{F}}(\lambda, 1)$.

As in the proof of Theorem 4.3, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of experiment Hyb_i 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 both correct and smooth (Definition 2.2).

Lemma B.1. *If F_1 is a single-key, selective-function-secure constrained PRF, then for all efficient adversaries $\mathcal{A}$, it follows that* $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \text{negl}(\lambda)$.

Proof. By construction, Hyb_0 and Hyb_1 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_0 from Hyb_1 with non-negligible advantage ε. Then, with probability ε, algorithm $\mathcal{A}$ will make a pre-challenge inversion query (y_1, y_2) such that the challenger's response is $\perp$ in Hyb_1, but not in Hyb_0. 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_0 and Hyb_1 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 $k^{(1)}_{f^*}$. It runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$ and constructs $k^{(2)}_{F^*} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(2)}, F_{\mathsf{sk}, f^*})$. It gives the constrained key $k_{f^*} = (k^{(1)}_{f^*}, k^{(2)}_{F^*}, \mathsf{pk})$ to $\mathcal{A}$. In addition, algorithm $\mathcal{B}$ initializes an empty set $S = \varnothing$ and an empty table of mappings $T = \varnothing$.

- **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 $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$ and $z \leftarrow \mathsf{F}_2(k^{(2)}, \mathsf{ct})$ and replies to $\mathcal{A}$ with $(\mathsf{ct}, z \oplus x)$. In addition, $\mathcal{B}$ adds the mapping $\mathsf{ct} \mapsto (z, x)$ 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 $x \leftarrow \mathsf{PKE.Decrypt}(\mathsf{sk}, y_1)$. 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 $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$ and $z \leftarrow \mathsf{F}_2(k^{(2)}, \mathsf{ct}) \oplus x$. Finally, algorithm $\mathcal{B}$ replies with x if $(\mathsf{ct}, z \oplus x) = (y_1, y_2)$ 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 $(y_1^*, y_2^*) \xleftarrow{\text{R}} S$. It computes $x^* \leftarrow \mathsf{PKE.Decrypt}(\mathsf{sk}, y_1^*)$ 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 $y_1^* = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r^*)$ 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 Hyb_1. By assumption, with probability ε, $\mathcal{A}$ will issue a pre-challenge inversion query (y_1^*, y_2^*) such that the challenger's response is $\perp$ in Hyb_1, but not in 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 Hyb_0 and Hyb_1 is identical.

- Since the challenger's response in Hyb_0 is not $\perp$, it must be the case that $y_1^* = \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$ for some $x \in \mathcal{X}$, $r_x = \mathsf{F}_1(k^{(1)}, x)$, and $y_2^* = \mathsf{F}_2(k^{(2)}, y_1^*) \oplus x$. From the first requirement, we have that the adversary did not previously submit an evaluation query for x. Since the challenger's response in Hyb_1 is $\perp$, it must additionally be the case that $f^*(x) = 0$.

With probability ε, $\mathcal{A}$ 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 ε/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 $r^* = \mathsf{F}_1(k^{(1)}, x^*)$. Then, with probability at least ε/Q, $y_1^* = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r^*)$, in which case, $\mathcal{B}$ outputs 1. Thus, in this case, $\mathcal{B}$ outputs 1 with probability at least ε/Q.

- Suppose r^* is uniform over $\mathcal{V}$ (and independent of all other quantities). Since PKE is smooth, $\Pr[\mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r^*) = y_1^*] = \mathrm{negl}(\lambda)$. In this case, $\mathcal{B}$ outputs 1 with negligible probability.

Since $Q = \mathrm{poly}(\lambda)$ and ε is non-negligible, we have that $\mathcal{B}$ breaks puncturing security of F_1 with probability at least $\varepsilon/Q - \mathrm{negl}(\lambda)$. 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 $\mathcal{A}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathrm{negl}(\lambda)$.*

Proof. Let $\mathcal{A}$ be a distinguisher for Hyb_1 and 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 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$, and $k_{F^*}^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(1)}, F_{\mathsf{sk}, f^*})$ as in Hyb_1 and Hyb_2. It gives the constrained key $k_{f^*} = (k_{f^*}^{(1)}, k_{F^*}^{(2)}, \mathsf{pk})$ to $\mathcal{A}$.

- **Query phase.** Algorithm $\mathcal{B}$ simulates the pre-challenge evaluation and inversion queries exactly as described in Hyb_1 and Hyb_2. Whenever it needs to compute $\mathsf{F}_1(k^{(1)}, x)$, 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 $\mathsf{ct}^* \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r^*)$ and $z^* \leftarrow \mathsf{F}_2(k^{(2)}, \mathsf{ct}^*)$. It gives $(\mathsf{ct}^*, \mathsf{F}_2(k^{(2)}, \mathsf{ct}^*) \oplus x^*)$ to $\mathcal{A}$. The post-challenge queries are handled exactly as in Hyb_1 and Hyb_2, except $\mathcal{B}$ queries its evaluation oracle for F_1 whenever it needs to compute $\mathsf{F}_1(k^{(1)}, x)$ for $x \neq x^*$, and $\mathcal{B}$ uses r^* in place of the value of $\mathsf{F}_1(k^{(1)}, x^*)$.

- **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 $\mathsf{F}_1(k^{(1)}, x^*)$, 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 $r^* = \mathsf{F}_1(k^{(1)}, x^*)$, then $\mathcal{B}$ perfectly simulates Hyb_1. If r^* is uniformly random over $\mathcal{V}$, then $\mathcal{B}$ perfectly simulates 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 F_1 is a single-key, selective-function-secure constrained PRF (Definition 3.7), then for all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \mathrm{negl}(\lambda)$.*

Proof. The only difference between Hyb_2 and Hyb_3 is that the challenger uses the value r^* in place of the value $\mathsf{F}_1(k^{(1)}, x^*)$ to answer the *post-challenge* evaluation and inversion queries in Hyb_2 while the challenger uses the real value $\mathsf{F}_1(k^{(1)}, x^*)$ in 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 Hyb_2 and Hyb_3.

- **Evaluation queries.** On an evaluation query x, the challenger in Hyb_2 and Hyb_3 only needs to evaluate $\mathsf{F}_1(k^{(1)}, \cdot)$ on x. Since Hyb_2 and Hyb_3 only differ in the value the challenger uses for $\mathsf{F}_1(k^{(1)}, x^*)$, the outputs of the evaluation oracle in Hyb_2 and 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 Hyb_2 and Hyb_3 are distributed identically.

- **Post-challenge inversion queries.** On a (post-challenge) inversion query (y_1, y_2), the challenger computes $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$, $r_x \leftarrow \mathsf{F}_1(k^{(1)}, x)$, and $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$. Then, it checks whether $y_1 = \mathsf{ct}$. If $x \neq x^*$, then the output of the inversion oracle in Hyb_2 and Hyb_3 is computed in the same way. It suffices to just consider the case where $x = x^*$, or equivalently, when $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$. Let $\mathsf{ct}^* = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r^*)$ be the ciphertext component in the challenge that is constructed using a uniformly random r^*.

  – In Hyb_2, the challenger uses the value r^* for the value $\mathsf{F}_1(k^{(1)}, x^*)$. For any query (y_1, y_2) where $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$, the challenger in Hyb_2 always responds with $\perp$ unless $y_1 = \mathsf{ct}^*$. But if $y_1 = \mathsf{ct}^*$, then it must be the case that $y_2 = \mathsf{F}_2(k^{(2)}, \mathsf{ct}^*) \oplus x^* = z^*$. This means that $(y_1, y_2) = (\mathsf{ct}^*, z^*)$, which is the challenge query. Therefore, the challenger in Hyb_2 responds with $\perp$ on all admissible inversion queries satisfying $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$.

  – In Hyb_3, on a query (y_1, y_2) where $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$, the challenger in Hyb_2 responds with $\perp$ unless $y_1 = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r_{x^*})$ where $r_{x^*} = \mathsf{F}_1(k^{(1)}, x^*)$. In this case, $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$.

  We conclude that the response of the challenger on an inversion query is identically distributed in Hyb_2 and Hyb_3 unless the adversary makes a (post-challenge) inversion query on the tuple (y_1, y_2) where $y_1 = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r_{x^*})$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$.

Suppose now that an adversary $\mathcal{A}$ is able to distinguish between Hyb_2 and Hyb_3 with some non-negligible probability ε. By the above analysis, the adversary's view in Hyb_2 and Hyb_3 is identically distributed unless the adversary makes a (post-challenge) inversion query on an input (y_1, y_2) where $y_1 = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r_{x^*})$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$. Since $\mathcal{A}$ distinguishes Hyb_2 from Hyb_3 with advantage ε, it will make an inversion query on a pair (y_1, y_2) satisfying the above property with probability at least ε. 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 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 $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$, a public/secret key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, and constructs the constrained key $k_{F^*}^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(2)}, F_{\mathsf{sk}, f^*})$. It sends $k_{f^*} = (k_{f^*}^{(1)}, k_{F^*}^{(2)}, \mathsf{pk})$ to $\mathcal{A}$.

- **Query phase.** Algorithm $\mathcal{B}$ simulates the evaluation, pre-challenge inversion, and challenge queries exactly as described in Hyb_2 and Hyb_3, with the exception that it queries its evaluation oracle for F_1 whenever it needs to evaluate $\mathsf{F}_1(k^{(1)}, \cdot)$. For post-challenge inversion queries (y_1, y_2) where $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$, 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 $\mathsf{F}_1(k^{(1)}, x)$.

- **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 $\widehat{\mathsf{ct}}^* \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; \hat{r}^*)$. Now, $\mathcal{B}$ checks whether $\mathcal{A}$ made an inversion query on the tuple $(\widehat{\mathsf{ct}}^*, \mathsf{F}_2(k^{(2)}, \widehat{\mathsf{ct}}^*) \oplus x^*)$. 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 Hyb_2 for $\mathcal{A}$. This means that with probability at least ε, adversary $\mathcal{A}$ will submit an inversion query on the tuple (y_1, y_2) where $y_1 = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r_{x^*})$, $r_{x^*} = \mathsf{F}_1(k^{(1)}, x^*)$, and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$. We now consider the probability that $\mathcal{B}$ outputs 1.

- Suppose $\hat{r}^* = \mathsf{F}_1(k^{(1)}, x^*)$. By assumption, with probability at least ε, adversary $\mathcal{A}$ will query the inversion oracle on the tuple (y_1, y_2) where $y_1 = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r_{x^*}) = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; \hat{r}^*)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x^*$ at some point in the execution. In this case, algorithm $\mathcal{B}$ outputs 1.

- Suppose $\hat{r}^*$ is uniformly random in $\mathcal{V}$. Since PKE is smooth, for any fixed string $\mathsf{ct} \in \mathcal{T}$, we have that $\Pr[\mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; \hat{r}^*) = \mathsf{ct}] = \operatorname{negl}(\lambda)$. If the adversary makes a total of $Q = \operatorname{poly}(\lambda)$ inversion queries, then by a union bound, the probability that $\widehat{\mathsf{ct}}^* = y_1$ for some y_1 appearing in one of $\mathcal{A}$'s inversion queries is bounded by $Q \cdot \operatorname{negl}(\lambda) = \operatorname{negl}(\lambda)$. Correspondingly, in this case, algorithm $\mathcal{B}$ outputs 1 with $\operatorname{negl}(\lambda)$ probability.

We conclude that if $\mathcal{A}$ is able to distinguish between hybrids Hyb_2 and Hyb_3 with non-negligible advantage ε, then $\mathcal{B}$ is able to break puncturing security of F_1 with probability $\varepsilon - \operatorname{negl}(\lambda)$. □

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

Proof. Suppose that there exists an adversary $\mathcal{A}$ that can distinguish between hybrids Hyb_3 and 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 $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$, the constrained key $k^{(1)}_{f^*} \leftarrow \mathsf{F}_1.\mathsf{Constrain}(k^{(1)}, f^*)$, and the PKE key $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$. It submits F_{sk, f^*} to the constrained PRF challenger for F_2 and receives a constrained key $k^{(2)}_{F^*}$. Algorithm $\mathcal{B}$ gives $k_{f^*} = (k^{(1)}_{f^*}, k^{(2)}_{F^*})$ to $\mathcal{A}$.

- **Query phase.** Algorithm $\mathcal{B}$ simulates the adversary's evaluation and inversion queries exactly as in Hyb_3 and Hyb_4, except whenever it needs to compute $\mathsf{F}_2(k^{(2)}, \cdot)$, it queries its evaluation oracle for F_2. For the challenge query x^*, $\mathcal{B}$ samples a random element $r^* \xleftarrow{\text{R}} \mathcal{V}$, computes $\mathsf{ct}^* \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r^*)$, and submits ct^* as the challenge query to the constrained

PRF challenger of F_2 to receives z^*. It returns $(\mathsf{ct}^*, z^* \oplus x^*)$ to the adversary. Subsequently, whenever it needs to compute $\mathsf{F}_2(k^{(2)}, \mathsf{ct}^*)$, 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, $F_{\mathsf{sk}, f^*}(\mathsf{ct}^*) = 0$ since $f^*(x^*) = 0$ and 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 ct encrypting x. Since $x \neq x^*$ and ct^* is an encryption of x^*, correctness of PKE implies that $\mathsf{ct} \neq \mathsf{ct}^*$. 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 $\mathsf{ct} \neq \mathsf{ct}^*$. In the post-challenge phase, $\mathcal{B}$ substitutes z^* in place of $\mathsf{F}_2(k^{(2)}, \mathsf{ct}^*)$, so we conclude that $\mathcal{B}$ never needs to make an evaluation query for ct^*.

To conclude the proof, we note that if $z^* = \mathsf{F}_2(k^{(2)}, \mathsf{ct}^*)$, then $\mathcal{B}$ perfectly simulates Hyb_3 for $\mathcal{A}$. If z^* is uniformly random over $\mathcal{X}$, $\mathcal{B}$ perfectly simulates 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 Hyb_3 and Hyb_4. □

Lemma B.5. *For all adversaries $\mathcal{A}$, we have that* $|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]| = \mathrm{negl}(\lambda)$.

Proof. The only difference between Hyb_4 and 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 $\mathsf{F}_2(k^{(2)}, \mathsf{ct}^*)$ in Hyb_4, while the challenger uses the real value $\mathsf{F}_2(k^{(2)}, \mathsf{ct}^*)$ in Hyb_5. Importantly, we will use the fact that in hybrids Hyb_4 and Hyb_5, the ciphertext 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 Hyb_4 and Hyb_5 are statistically indistinguishable.

- **Evaluation queries.** On an evaluation query x, the challenger in Hyb_4 and Hyb_5 first computes $r_x \leftarrow \mathsf{F}_1(k^{(1)}, x)$, $\mathsf{ct} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$, and then evaluates $\mathsf{F}_2(k^{(2)}, \mathsf{ct})$. Therefore, the response to the evaluation query in Hyb_4 and Hyb_5 differ only on points $x \in \mathcal{X}$ where $\mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x) = \mathsf{ct}^*$. By construction, ct^* is an encryption of x^* under pk, so by correctness of PKE, for all $x \neq x^*$, $\mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x) \neq \mathsf{ct}^*$. This means that the outputs of the evaluation oracle (on all admissible queries) in Hyb_4 is the same as that in Hyb_5.

- **Inversion queries.** On an inversion query (y_1, y_2), the challenger in Hyb_4 and Hyb_5 first computes $x \leftarrow \mathsf{F}_2(k^{(2)}, y_1) \oplus y_2$ and then checks whether $y_1 = \mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x)$, where $r_x = \mathsf{F}_1(k^{(1)}, x)$. On all queries where $y_1 \neq \mathsf{ct}^*$, the challenger's behavior in Hyb_4 and Hyb_5 is identical. On queries (ct^*, y_2), the only instance where the adversary does not output $\perp$ is if $\mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x) = \mathsf{ct}^*$. By correctness of PKE, if $x \neq x^*$, then $\mathsf{PKE.Encrypt}(\mathsf{pk}, x; r_x) \neq \mathsf{ct}^*$. Finally, since PKE is smooth, over the randomness used to construct ct^* (more specifically, over the randomness used to sample r^*), $\Pr[\mathsf{ct}^* = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r_{x^*})] = \mathrm{negl}(\lambda)$, where $r_{x^*} = \mathsf{F}_1(k^{(1)}, x^*)$. This means that with overwhelming probability, the response to all of the inversion queries of the form (ct^*, y_2) in Hyb_4 and Hyb_5 will be $\perp$. The claim follows. □

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* $|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]| = \mathrm{negl}(\lambda)$

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between hybrids Hyb_5 and Hyb_6. We construct an adversary $\mathcal{B}$ to distinguish the two experiments $\mathsf{Real}_{\mathcal{B},\mathsf{F}_2}$ and $\mathsf{Ideal}_{\mathcal{B},\mathcal{S},\mathsf{F}_2}$ 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 $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$, the constrained key $k^{(1)}_{f^*} \leftarrow \mathsf{F}_1.\mathsf{Constrain}(k^{(1)}, f^*)$, and a public/secret key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE}.\mathsf{Setup}(1^\lambda)$. It commits to the function F_{sk,f^*} in the privacy game, and receives a constrained key $k^{(2)}_{F^*}$ from the challenger. Finally, it gives $(k^{(1)}_{f^*}, k^{(2)}_{F^*}, \mathsf{pk})$ to $\mathcal{A}$.
- **Query phase.** Algorithm $\mathcal{B}$ answers all of the queries exactly as described in Hyb_5, except whenever it needs to evaluate $\mathsf{F}_2(k^{(2)}, \mathsf{ct})$, it instead queries its evaluation oracle on ct, and uses the response of its evaluation oracle in place of $\mathsf{F}_2(k^{(2)}, \mathsf{ct})$.

In $\mathsf{Real}_{\mathcal{B},\mathsf{F}_2}$, the constrained key $k^{(1)}_{f^*}$ is constructed by first computing $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$ and then $k^{(2)}_{F^*} \leftarrow \mathsf{F}_2.\mathsf{Constrain}(k^{(2)}, F_{\mathsf{sk},f^*})$. The evaluation oracle is implemented by $\mathsf{F}_2(k^{(2)}, \cdot)$. This precisely corresponds to Hyb_5. In $\mathsf{Ideal}_{\mathcal{B},\mathcal{S},\mathsf{F}_2}$, the constrained key is constructed by computing by invoking $\mathcal{S}_1(1^\lambda)$, and the evaluations $\mathsf{F}_2(k^{(2)}, \mathsf{ct})$ is implemented by $\mathcal{S}_2(\mathsf{ct}, F_{\mathsf{sk},f^*}(\mathsf{ct}), \mathsf{st}_{\mathcal{S}})$, which precisely corresponds to the distribution in Hyb_6. Thus, if $\mathcal{A}$ is able to distinguish between hybrids Hyb_5 and Hyb_6 with non-negligible advantage, then $\mathcal{B}$ is able to break privacy of F_2 with the same probability. ☐

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* $|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]| = \operatorname{negl}(\lambda)$.

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between hybrids Hyb_6 and 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 $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$, $k^{(1)}_{f^*} \leftarrow \mathsf{F}_1.\mathsf{Constrain}(k^{(1)}, f)$, and $(k^{(2)}_{F^*}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$. It receives pk from the CCA-security challenger, and gives the constrained key $(k^{(1)}_{f^*}, k^{(2)}_{F^*}, \mathsf{pk})$ 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 $r_x \leftarrow \mathsf{F}_1(k^{(1)}, x)$, and $\mathsf{ct} \leftarrow \mathsf{PKE}.\mathsf{Encrypt}(\mathsf{pk}, x; r_x)$. Then, it computes $(z, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_2(\mathsf{ct}, f^*(x), \mathsf{st}_{\mathcal{S}})$ and returns (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 Hyb_6 and Hyb_7.
  - **Challenge query.** On input $x^* \in \mathcal{X}$, $\mathcal{B}$ samples a random point $\hat{x}^* \xleftarrow{\text{R}} \mathcal{X}$. It submits the pair $(x^*, \hat{x}^*)$ as its challenge in the CCA-security game, and receives a challenger ciphertext ct^*. Next, $\mathcal{B}$ samples a random point $z^* \xleftarrow{\text{R}} \mathcal{X}$ and gives (ct^*, z^*) to $\mathcal{A}$.

- **Post-challenge inversion query.** On input $(y_1, y_2) \in \mathcal{Y}$, if $y_1 = \mathsf{ct}^*$, 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 Hyb_6 and Hyb_7.

By construction, $\mathcal{B}$ never queries the decryption oracle on the challenge ciphertext ct^*, and thus, is admissible. To conclude the proof, we show that if the challenge ciphertext ct^* is an encryption of x^*, then $\mathcal{B}$ correctly simulates the distribution in Hyb_6 and if ct^* is an encryption of $\hat{x}^*$, then $\mathcal{B}$ correctly simulates the distribution in 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 ct^* is an encryption of x^*. Then, the challenge query is simulated exactly as in Hyb_6 (recall that in Hyb_6, the ciphertext ct^* is an encryption of x^* with uniform randomness). For the (post-challenge) inversion queries, it suffices to argue that in Hyb_6, the challenger's response to any (post-challenge) inversion query of the form (ct^*, z) for any $z \in \mathcal{X}$ is $\perp$. In Hyb_6, ct^* is an encryption of x^* using randomness that is sampled uniformly at random. Since PKE is smooth, $\Pr[\mathsf{ct}^* = \mathsf{PKE.Encrypt}(\mathsf{pk}, x^*; r_{x^*})] = \operatorname{negl}(\lambda)$ where $r_{x^*} = \mathsf{F}_1(k^{(1)}, x^*)$. Thus, with overwhelming probability, the output on the inversion queries of the form (ct^*, z) in Hyb_6 is $\perp$, and so the simulation is correct.

- Suppose ct^* is an encryption of $\hat{x}^*$. Then, the challenge query is simulated exactly as in Hyb_7. For the (post-challenge) inversion queries, it suffices to argue that in Hyb_7, the challenger's response to any (post-challenge) inversion query of the form (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 (ct^*, z) is not $\perp$ only if $\mathsf{ct}^* = \mathsf{PKE.Encrypt}(\mathsf{pk}, \hat{x}^*; r_{\hat{x}^*})$ where $r_{\hat{x}^*} = \mathsf{F}_1(k^{(1)}, \hat{x}^*)$. Since PKE is smooth, over the randomness used to sample ct^*, this occurs with negligible probability.

We conclude that if $\mathcal{A}$ is able to distinguish between hybrids Hyb_6 and Hyb_7 with non-negligible probability, then $\mathcal{B}$ breaks CCA-security of PKE with the same probability. □

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* $|\Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1]| = \operatorname{negl}(\lambda)$.

Proof. Follows by an analogous argument as that in the proof of Lemma B.6. □

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

Proof. Follows by an analogous argument as that in the proof of Lemma B.1. □

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

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 $k = (k^{(1)}, k^{(2)}) \leftarrow \mathsf{F.Setup}(1^\lambda)$. Let $k_f = (P_1, P_2) \leftarrow \mathsf{F.Constrain}(k, f)$. We consider each of the two correctness requirements separately.

- By correctness of $i\mathcal{O}$, $P_1(x) = P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}](x)$ for all $x \in \mathcal{X}$. Since $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}](x) = \mathsf{F}(k, x)$ for all x where $f(x) = 1$, $\mathsf{F.Eval}(k_f, x) = P_1(x) = \mathsf{F}(k, x)$ for all x where $f(x) = 1$.
- By correctness of $i\mathcal{O}$, $\mathsf{F.Eval}^{-1}(k_f, y) = P_2(y) = P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}](y)$ 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 $\mathsf{F}(k, x) = (y_1, y_2)$. This means that $y_1 = \mathsf{F}_1(k^{(1)}, x)$ and $y_2 = \mathsf{F}_2(k^{(2)}, y_1) \oplus x$. Then, by construction, $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}](y) = x$. $\square$

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_0: This is the constrained IPF security experiment $\mathsf{Expt}^{(\mathsf{IPF})}_{\mathcal{A},\mathsf{F}}(\lambda, 0)$ from Definition 3.11. During the setup phase, after the adversary commits to a point $x^* \in \mathcal{X}$, the challenger samples PRF keys $k^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Setup}(1^\lambda)$, $k^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Setup}(1^\lambda)$ and sets $k = (k^{(1)}, k^{(2)})$. Next, it computes $v^* \leftarrow \mathsf{F}_1(k^{(1)}, x^*)$ and $z^* \leftarrow \mathsf{F}_2(k^{(2)}, v^*)$. It then gives the challenge $y^* = (v^*, z^* \oplus x^*)$ to the adversary. The challenger answers the constrain, evaluation, and inversion queries using $\mathsf{F.Constrain}(k, \cdot)$, $\mathsf{F}(k, \cdot)$, and $\mathsf{F}^{-1}(k, \cdot)$, respectively.

- Hyb_1: Same as Hyb_0, except the challenger substitutes a punctured key $k_{x^*}^{(1)}$ for $k^{(1)}$ and a punctured evaluation algorithm $\mathsf{F}_1.\mathsf{Eval}(k_{x^*}^{(1)}, \cdot)$ for $\mathsf{F}_1(k^{(1)}, \cdot)$ when constructing the obfuscated programs P_1 and P_2 in the constrain queries. More concretely, during the setup phase, the challenger sets $k_{x^*}^{(1)} \leftarrow \mathsf{F}_1.\mathsf{Puncture}(k^{(1)}, x^*)$. When responding to the constrain queries, the challenger constructs $P_1 \leftarrow i\mathcal{O}(P^{\mathsf{Eval}}[f, k_{x^*}^{(1)}, k^{(2)}])$ and $P_2 \leftarrow i\mathcal{O}(P^{\mathsf{Inv}}[f, k_{x^*}^{(1)}, k^{(2)}])$, where we write $P^{\mathsf{Eval}}[f, k_{x^*}^{(1)}, k^{(2)}]$ and $P^{\mathsf{Inv}}[f, k_{x^*}^{(1)}, k^{(2)}]$ to denote the programs in Figures 1 and 2, respectively, where $\mathsf{F}_1(k^{(1)}, \cdot)$ is replaced by $\mathsf{F}_1.\mathsf{Eval}(k_{x^*}^{(1)}, \cdot)$.
- Hyb_2: Same as Hyb_1, except the challenger samples $v^* \xleftarrow{\mathrm{R}} \mathcal{V}$ 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 $\mathsf{F}_1(k^{(1)}, x^*)$.
- Hyb_3: Same as Hyb_2, except when responding to the evaluation and inversion queries, the challenger always evaluates $\mathsf{F}_1(k^{(1)}, \cdot)$ instead of substituting the value v^* for $\mathsf{F}_1(k^{(1)}, x^*)$.
- Hyb_4: Same as Hyb_3, except the challenger reverts to using the real key $k^{(1)}$ (instead of $k_{x^*}^{(1)}$) and the real evaluation algorithm $\mathsf{F}_1(k^{(1)}, \cdot)$ (instead of $\mathsf{F}_1.\mathsf{Eval}(k_{x^*}^{(1)}, \cdot)$) when constructing the obfuscated programs P_1 and P_2 in the constrain queries.
- Hyb_5: Same as Hyb_4, except the challenger substitutes a punctured key $k_{v^*}^{(2)}$ for $k^{(2)}$ and a punctured evaluation algorithm $\mathsf{F}_2.\mathsf{Eval}(k_{v^*}^{(2)}, \cdot)$ for $\mathsf{F}_2(k^{(2)}, \cdot)$ when constructing the obfuscation programs P_1 and P_2 in the constrain queries. More concretely, during the setup phase, after the challenger samples $v^* \xleftarrow{\mathrm{R}} \mathcal{V}$, the challenger sets $k_{v^*}^{(2)} \leftarrow \mathsf{F}_2.\mathsf{Puncture}(k^{(2)}, v^*)$. When responding to the constrain queries, the challenger constructs $P_1 \leftarrow i\mathcal{O}(P^{\mathsf{Eval}}[f, k^{(1)}, k_{v^*}^{(2)}])$ and $P_2 \leftarrow i\mathcal{O}(P^{\mathsf{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}])$, where we write $P^{\mathsf{Eval}}[f, k^{(1)}, k_{v^*}^{(2)}]$ and $P^{\mathsf{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}]$ to denote the programs in Figures 1 and 2, respectively, where $\mathsf{F}_2(k^{(2)}, \cdot)$ is replaced by $\mathsf{F}_2.\mathsf{Eval}(k_{v^*}^{(2)}, \cdot)$.
- Hyb_6: Same as Hyb_5, except during the setup phase, the challenger samples $z^* \xleftarrow{\mathrm{R}} \mathcal{X}$, 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 $\mathsf{F}_2(k^{(2)}, v^*)$.
- Hyb_7: Same as Hyb_6, except when responding to the evaluation and inversion queries, the challenger always computes $\mathsf{F}_2(k^{(2)}, \cdot)$ instead of substituting the value $(z^* \oplus x^*)$ for $\mathsf{F}_2(k^{(2)}, v^*)$.
- Hyb_8: Same as Hyb_7, except for each constrain query that the adversary makes, instead of using the program $P^{\mathsf{Eval}}[f, k^{(1)}, k_{v^*}^{(2)}]$ and $P^{\mathsf{Inv}}[f, k^{(1)}, k_{v^*}^{(2)}]$, the challenger uses $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}]$ and $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}]$ by replacing $\mathsf{F}_2.\mathsf{Eval}(k_{v^*}^{(2)}, \cdot)$ with $\mathsf{F}_2(k^{(2)}, \cdot)$. It sets $P_1 \leftarrow i\mathcal{O}(P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}])$ and $P_2 \leftarrow i\mathcal{O}(P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}])$ and uses $k_f = (P_1, P_2)$ as the constrained key. This is the constrained IPF security experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{F}}^{(\mathsf{IPF})}(\lambda, 1)$ from Definition 3.11.

As in the proofs of Theorems 4.3 and 4.7, we write $\mathsf{Hyb}(\mathcal{A})$ to denote the output of experiment 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 $\mathcal{A}$,* $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathrm{negl}(\lambda)$.

Proof. The two experiments are identical except the challenger uses the programs $P^{\mathsf{Eval}}[f, k^{(1)}_{x^*}, k^{(2)}]$ and $P^{\mathsf{Inv}}[f, k^{(1)}_{x^*}, k^{(2)}]$ in place of $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}]$ and $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}]$, respectively, in Hyb_1 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_0 and Hyb_1. 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_0 and Hyb_1 is computationally indistinguishable. It suffices to show that for all $x \in \mathcal{X}$, $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}](x) = P^{\mathsf{Eval}}[f, k^{(1)}_{x^*}, k^{(2)}](x)$. Take $x \in \mathcal{X}$. By correctness of F_1, we have that $\mathsf{F}_1(k^{(1)}, x) = \mathsf{F}_1.\mathsf{Eval}(k^{(1)}_{x^*}, x)$ for all $x \neq x^*$. Thus, $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}](x) = P^{\mathsf{Eval}}[f, k^{(1)}_{x^*}, k^{(2)}](x)$ for all $x \neq x^*$. When $x = x^*$, we have that $f(x^*) = 0$ by admissibility. Then, $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}](x^*) = \perp = P^{\mathsf{Eval}}[f, k^{(1)}_{x^*}, k^{(2)}](x^*)$. The claim follows by security of $i\mathcal{O}$.

- Next, we show that the distribution of P_2 in Hyb_0 and Hyb_1 is computationally indistinguishable. As above, we show that for all $y \in \mathcal{Y}$, $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}](y) = P^{\mathsf{Inv}}[f, k^{(1)}_{x^*}, k^{(2)}](y)$. Take any $y = (y_1, y_2) \in \mathcal{Y}$. By construction of P^{Inv} and correctness of F_1, $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}](y) = P^{\mathsf{Inv}}[f, k^{(1)}_{x^*}, k^{(2)}](y)$ whenever $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 \neq x^*$. When $\mathsf{F}_2(k^{(2)}, y_1) \oplus y_2 = x^*$, we have that $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}](y) = \perp = P^{\mathsf{Inv}}[f, k^{(1)}_{x^*}, k^{(2)}](x^*)$ because $f(x^*) = 0$ by admissibility. The claim follows by security of $i\mathcal{O}$. □

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

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 Hyb_1 and Hyb_2 (using the punctured key $k^{(1)}_{x^*}$). □

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

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 Hyb_2 and Hyb_3, so they do not complicate the proof. □

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

Proof. Follows by the same argument as that used in the proof of Lemma C.1. □

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

Proof. The two experiments are identical except the challenger uses the programs $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}_{v^*}]$ and $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}_{v^*}]$ in place of $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}]$ and $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}]$, respectively, in 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 Hyb_4 and Hyb_5 is computationally indistinguishable. It suffices to show that for all $x \in \mathcal{X}$, $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}](x) = P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}_{v^*}](x)$. Take $x \in \mathcal{X}$. Since v^* is uniform over $\mathcal{V}$ (and sampled independently of $k^{(1)}$), with probability $1 - |\mathcal{X}| / |\mathcal{V}| = 1 - \mathrm{negl}(\lambda)$, there does not exist any $x \in \mathcal{X}$ such that $\mathsf{F}_1(k^{(1)}, x) = v^*$. Thus, with overwhelming probability, $\mathsf{F}_1(k^{(1)}, x) \neq v^*$. By correctness of F_2, $\mathsf{F}_2(v) = \mathsf{F}_2.\mathsf{Eval}(k^{(2)}, v)$ for all $v \neq v^*$. This means that with overwhelming probability, $P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}](x) = P^{\mathsf{Eval}}[f, k^{(1)}, k^{(2)}_{v^*}](x)$ 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 Hyb_4 and Hyb_5 is computationally indistinguishable. It suffices to show that for all $y \in \mathcal{Y}$, $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}](y) = P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}_{v^*}](y)$ for all $y \in \mathcal{Y}$. Take any $y = (y_1, y_2) \in \mathcal{Y}$. By correctness of F_2, we have that $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}](y_1, y_2) = P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}_{v^*}](y_1, y_2)$ 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 $\mathsf{F}_1(k^{(1)}, x) = v^*$. Thus, with overwhelming probability (over the choice of v^*), the programs $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}]$ and $P^{\mathsf{Inv}}[f, k^{(1)}, k^{(2)}_{v^*}]$ both output $\perp$ whenever $y_1 = v^*$. The claim then follows by $i\mathcal{O}$ security. ☐

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

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 Hyb_5 and Hyb_6 (using the punctured key $k^{(2)}_{v^*}$). ☐

Lemma C.7. *If $|\mathcal{X}| / |\mathcal{V}| = \mathrm{negl}(\lambda)$, then for all adversaries $\mathcal{A}$,*

$$|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]| = \mathrm{negl}(\lambda).$$

Proof. Follows by an analogous argument as that in the proof of Lemma A.4. ☐

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

Proof. Follows by an analogous argument as that in the proof of Lemma C.5. ☐

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