Constraining Pseudorandom Functions Privately

Dan Boneh Kevin Lewi David J. Wu

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

Abstract

In a constrained pseudorandom function (PRF), the holder of the master secret key can derive constrained keys with respect to a boolean circuit C. The constrained key can be used to evaluate the PRF on all inputs x for which $C(x) = 1$. In almost all existing constructions of constrained PRFs, the constrained key itself reveals its constraints. We introduce the concept of *private* constrained PRFs, which are constrained PRFs with the additional property that the constrained keys do not reveal their constraints. Our main notion of privacy captures the intuition that an adversary, given a constrained key for one of two circuits C_0 and C_1, is unable to tell which circuit is associated with its key. As a primitive, private constrained PRFs have many natural applications in searchable symmetric encryption, deniable encryption, and more.

We then show how to instantiate private constrained PRFs. Our first construction uses indistinguishability obfuscation and achieves our strongest notions of functionality and privacy. We also give two constructions based on concrete assumptions on multilinear maps which achieve slightly weaker notions of privacy and for more limited classes of constraints: namely, for the class of bit-fixing constraints and puncturing constraints.

1 Introduction

A pseudorandom function (PRF) [GGM86] is a (keyed) function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ with the property that, for a randomly chosen key $\mathsf{msk} \in \mathcal{K}$, the outputs of $F(\mathsf{msk}, \cdot)$ look indistinguishable from the outputs of a truly random function from $\mathcal{X}$ to $\mathcal{Y}$. Constrained PRFs[1], proposed independently by Boneh and Waters [BW13], Boyle, Goldwasser, and Ivan [BGI14], and Kiayias, Papadopoulos, Triandopoulos and Zacharias [KPTZ13], behave just like standard PRFs, except that the holder of the (master) secret key $\mathsf{msk} \in \mathcal{K}$ for the PRF is also able to produce a constrained key sk_C for a boolean circuit C. This constrained key sk_C can be used to evaluate the PRF $F(\mathsf{msk}, \cdot)$ on all inputs $x \in \mathcal{X}$ where $C(x) = 1$, but sk_C reveals nothing about $F(\mathsf{msk}, x)$ when $C(x) = 0$. Constrained PRFs have found many applications, for example, in broadcast encryption [BW13] and in the "punctured programming" techniques of Sahai and Waters [SW14].

The Goldreich-Goldwasser-Micali (GGM) PRF [GGM86] is a *puncturable* PRF, that is, a constrained PRF for the special class of puncturing constraints. In a puncturable PRF, each constrained key is associated with an input $x \in \mathcal{X}$, and the constrained key allows for evaluation at all points $x' \neq x$ while revealing no information about $F(\mathsf{msk}, x)$. Boneh and Waters [BW13] show how to use multilinear maps [GGH13a, CLT13, GGH14, CLT15] to construct constrained PRFs

[1] They have also been called *functional* PRFs [BGI14] and *delegatable* PRFs [KPTZ13]

for more expressive classes of constraints, including bit-fixing constraints as well as general circuit constraints (of a priori bounded depth). Subsequent works in this area have focused on achieving adaptive notions of security [HKKW14, HKW14], developing schemes with additional properties such as verifiability [CRV14], and constructing circuit-constrained PRFs from standard lattice-based assumptions [BV15].

Constraining privately. In this work, we initiate the study of *private* constrained PRFs, which can be viewed as a natural extension of constrained PRFs with the additional property that the constrained keys should not reveal their constraints.

Our definition of privacy requires that an adversary, given a single constrained key sk for one of two possible circuits C_0 and C_1, cannot tell which circuit was used as the constraint for sk. We also generalize this definition to the setting where the adversary obtains multiple constrained keys. Since the adversary can compare the outputs from multiple constrained keys, some information is necessarily leaked about the underlying constraints. In this setting, our privacy property ensures that the adversary learns the minimum possible.

For the special case of a puncturable PRF (where the adversary only has access to a single constrained key), the privacy requirement is that for any two adversarially-chosen points $x_0, x_1 \in \mathcal{X}$, the adversary cannot distinguish a secret key punctured at x_0 from one punctured at x_1. In particular, this means that using a secret key punctured at the input x to evaluate the PRF on x must return a value that is *unpredictable* to the adversary, as opposed to a fixed constant value or $\perp$ as is done in existing (non-private) constrained PRF constructions.

While privacy is a very simple requirement to impose on constrained PRFs, it is not clear how to adapt existing schemes to satisfy this property, even just for puncturing. As a first attempt to constructing private puncturable PRFs, let the PRF input space $\mathcal{X}$ be $\{0,1\}^n$, and consider the GGM tree-based PRF [GGM86], where the outputs are computed as the leaf nodes of a binary tree with the PRF secret key occupying the root node. To puncture the GGM PRF at an input x, the puncturing algorithm reveals the secret keys of all internal nodes that are adjacent[2] to the path from the root to the leaf node corresponding with x. Certainly then, the GGM construction is not private—given the punctured key, an adversary can easily reconstruct the path from the root to the punctured leaf node, and hence, recover the input x.

However, the GGM PRF is a private constrained PRF for the class of length-ℓ prefix constraints, for an integer $\ell \leq n$. This class refers to the family of constraints described by a prefix $s \in \{0,1\}^\ell$, where an input satisfies the constraint if its first ℓ bits match s. To constrain the GGM PRF on a prefix s, the constrain algorithm reveals the secret key for the internal node associated with s in the GGM tree. Then, to evaluate an input x using the constrained key, the evaluator discards the first ℓ bits of x and, beginning with the node associated with the constrained key, uses the remaining bits of x to traverse down the GGM tree, outputting the value associated with the resulting leaf node. Privacy follows from the fact that, without the original root of the GGM tree, the secret key for the internal node for s appears to be distributed uniformly and independently of s.

While the GGM PRF provides an efficient solution to privately constraining PRFs under fixed-length prefix constraints, this is insufficient for the applications we have in mind. Instead, we construct private constrained PRFs for more general classes of constraints: puncturing and general circuit constraints.

[2]Here, an internal node is "adjacent" to a path if it does not lie on the path but its parent does.

1.1 Applications of Private Constrained PRFs

To illustrate the power of private constrained PRFs we first describe a few natural applications, including deniable encryption, watermarkable PRFs, and searchable encryption.

Private constrained MACs. Constrained MACs are the secret-key variant of constrained signatures, which were first introduced by Boyle et al. [BGI14]. In a constrained MAC, the holder of the master secret key can issue constrained secret keys to users. Given a constrained key, a user can only generate MACs for messages that conform to some pre-specified constraint. Here, we consider private constrained MACs, where the constraint is also hidden from the user. Just as a secure PRF implies a secure MAC, a private constrained PRF yields a private constrained MAC.

As a concrete example, suppose a company would like to enforce spending limits on its employees. For business reasons, they do not want employees to be able to learn their precise spending limit, which might reveal confidential information about their position and rank within the company. For example, an employee Alice might only be allowed to create spending requests for at most \$500. In this case, Alice's company could issue a constrained key to Alice that restricts her to only being able to compute MACs for messages which contain her name and whose spending requests do not exceed \$500. If Alice attempts to create a MAC for a spending request that either exceeds \$500 or is not bound to her name, then the computed MAC will not pass verification. Moreover, privacy of the constrained key ensures that Alice cannot tell if the MAC she constructed is valid or not with respect to the master verification key. Hence, without interacting with the verifier, Alice learns nothing about her exact spending limit. A key advantage in this scenario is that the purchase authorizer who holds the master verification key does not need to keep track of the spending limits for each employee, since they are already encoded and enforced in the constrained MAC keys.

Symmetric deniable encryption. The notion of deniable encryption was first introduced by Canetti, Dwork, Naor, and Ostrovsky [CDNO97]. Informally speaking, a deniable encryption scheme allows a sender and receiver, after exchanging encrypted messages, to later on produce either fake randomness (in the public-key setting), or a fake decryption key (in the symmetric-key setting) that opens a ciphertext to another message of their choosing. Of course, the fake randomness or decryption key that is constructed by this "deny" algorithm should look like legitimately-sampled randomness or an honestly-generated decryption key.

Recently, Sahai and Waters [SW14] used indistinguishability obfuscation [BGI$^+$12, GGH$^+$13b, BGK$^+$14, SW14, GLSW14, Zim15] to give the first construction of public-key deniable encryption that achieves the security notions put forth by Canetti et al.[3] In all prior constructions of deniable encryption, the adversary is able to distinguish real randomness from fake randomness with advantage $1/n$, where n roughly corresponds to the length of a ciphertext in the scheme [CDNO97].

Surprisingly, the machinery of private puncturable PRFs provides a direct solution to a variant of symmetric deniable encryption. In the symmetric setting, we assume that an adversary has intercepted a collection of ciphertexts $c_1, \ldots, c_n$ and asks the sender to produce the secret key to decrypt this collection of messages. The deniable encryption scheme that we construct enables the sender to produce a fake secret key sk that looks indistinguishable from an honestly generated

[3] In fact, their construction achieves the stronger notion of publicly deniable encryption where the sender does not have to remember the randomness it used to construct a particular ciphertext when producing fake randomness.

encryption key, and yet, will only correctly decrypt all but one of the intercepted ciphertexts.[4] In our particular construction, the sender (or receiver) has a trapdoor that can be used to deny messages. Our framework is similar to the *flexibly* deniable framework where there are separate key-generation and encryption algorithms [CDNO97, OPW11] for so-called "honest" encryption and "dishonest" encryption. A second difference in our setting is that we only support denying to a random message rather than an arbitrary message of the sender's choosing. Thus, our scheme is better-suited for scenarios where the messages being encrypted have high entropy (e.g., cryptographic keys).

Our construction of a symmetric deniable encryption scheme uses a private puncturable PRF. Suppose F is a privately puncturable PRF with output space $\{0,1\}^m$. We use F to construct a symmetric deniable encryption scheme with message space $\{0,1\}^m$. The symmetric keys sk in our deniable encryption scheme consists of constrained keys punctured at a random point $x \in \mathcal{X}$. The denying trapdoor is the master secret key msk. An encryption of a message y is just the tuple $(r, F(\mathsf{msk}, r) \oplus y)$, where r is a randomly sampled nonce. With overwhelming probability, the key sk is not punctured at r, so evaluating sk at r yields $F(\mathsf{msk}, r)$. Later, to deny a particular ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$, the sender (or receiver) takes msk and derives a key sk' punctured at ct_1. Privacy for the puncturable PRF ensures that sk' looks indistinguishable from the real key sk. Security of the puncturable PRF ensures that the blinding factor $F(\mathsf{msk}, \mathsf{ct}_1)$ looks random to an adversary who only has sk'. Thus, under sk', the ciphertext ct looks like the encryption of a random message. Finally, by correctness of the punctured encryption scheme, the "fake" key sk' still correctly decrypts the other ciphertexts. We give a complete description of our model and construction, as well as a survey of additional related work in Section 6.1.

Watermarking PRFs. A watermarking scheme for programs [HMW07, BGI+12, CHV15, NW15, CHN+15] consists of a marking algorithm, which takes as input a program and embeds a "mark" in it, and a verification algorithm that takes an arbitrary program and determines whether it has been marked. The requirement is that a marked program should preserve the functionality of the original program on almost all inputs, but still be difficult for an adversary to remove the watermark without destroying the functionality. As discussed in [HMW07, BGI+12, CHN+15], the marking algorithm can be extended to embed a string into the program; correspondingly, the verification algorithm would extract the embedded string when run on a watermarked program. We say such schemes are message embedding [CHN+15].

Hopper, Molnar, and Wagner [HMW07] first introduced the formal notion of a secretly-verifiable watermarking scheme, which was then discussed and adapted to the setting of watermarking cryptographic programs in Barak et al. [BGI+12]. In a secretly-verifiable scheme, only the holder of a secret key can test if a program is watermarked. More recently, Cohen et al. [CHN+15] showed how to construct publicly-verifiable watermarking for puncturable PRFs from indistinguishability obfuscation. In the publicly-verifiable setting, anyone with the public parameters is able to test whether a program is watermarked or not. Moreover, Cohen et al. noted that watermarkable PRFs have applications in challenge-response authentication and traitor tracing. We survey more related work in Section 6.2.

In our work, we show that starting with a private *programmable* PRF, we obtain a watermarkable family of PRFs, where the associated watermarking scheme is secretly-verifiable and supports message embedding. Intuitively, a programmable PRF is a puncturable PRF, except with the

[4] It is important to define our notions with respect to multiple intercepted messages. Otherwise, the one-time-pad is a trivial symmetric deniable encryption scheme.

property that the holder of the master secret key can additionally specify the value the constrained key evaluates to at the punctured point. The privacy requirement stipulates that a programmed key hides the point which was "reprogrammed." We give the formal definitions of this concept in Appendix C, and a concrete construction based on indistinguishability obfuscation in Appendix E.

We now give an overview of our construction of a watermarkable PRF. For simplicity, we describe our construction without message embedding. To mark a key msk for a private programmable PRF, the marking algorithm first evaluates $F(\mathsf{msk}, \cdot)$ at several (secret) points $z_1, \ldots, z_d \in \mathcal{X}$ to obtain values $t_1, \ldots, t_d$. The marking algorithm then derives a pseudorandom pair (x, y) from the values $t_1, \ldots, t_d$, and outputs a programmed key for msk with the value at x replaced by y. To test whether a circuit C is marked or not, the verification algorithm applies the same procedure as the marking algorithm to obtain a test point (x', y'). The test algorithm then outputs "marked" if $C(x') = y'$ and "unmarked" otherwise. Privacy is crucial here because if the adversary knew the "reprogrammed" point x, it can trivially remove the watermark by producing a circuit that simply changes the value at x. We show more formally in Section 6.2 that this simple construction not only satisfies our notion of secretly-verifiable watermarking, but can be easily extended to support watermarking of arbitrary messages.

Although our current constructions of private programmable PRFs rely on indistinguishability obfuscation, we stress that advances in constructing private programmable PRFs from weaker assumptions or with improved efficiency would have implications in constructing watermarkable PRFs as well.

Searchable encryption. In searchable symmetric encryption (SSE) [SWP00, Goh03, CGKO06, CK10, BCLO09], a server holds a set of encrypted documents and a client wants to retrieve all documents that match its query. For simplicity, suppose each document is tagged, and the client wants to retrieve all documents with a particular tag. One of the simplest SSE approaches is to compute and store an encrypted index on the server. Specifically, fix a PRF F and a key msk. For each tag t, the encrypted index maps the token $F(\mathsf{msk}, t)$ onto an encrypted list of document indices that match the tag. To search for a tag t, a user who holds the PRF key msk can issue a query $F(\mathsf{msk}, t)$. The server returns the encrypted list of matching documents.

We consider a new notion called restrictable SSE, where multiple parties can search the database, and the database owner wants to prevent some users from searching for certain tags. For example, suppose a company hosts all of its documents in a central database and tags each document with the name of its associated project. Moreover, suppose the company is developing a top-secret project and wants to restrict access so that only employees working on the project are able to search for documents related to the project. Using restrictable SSE, the company can issue restricted search keys to all employees not working on the project. Security of the constrained PRF ensures that an employee is unable to search for documents pertaining to the secret project. If we moreover assume that the tags are drawn from a small (polynomially-sized) domain (e.g., the English dictionary), privacy ensures that an employee cannot tell if a search came back empty because she was not allowed to search for a particular tag, or if there are actually no documents that match the tag. Privacy also ensures that unauthorized employees cannot infer the name of the secret project from their search keys.

By instantiating F with a private constrained PRF, we easily obtain a restrictable SSE system. The construction is collusion resistant: if several employees who individually cannot search for the tag t combine their search keys, they still cannot search for t. However, it does become possible for

them to test whether a certain tag is in the intersection of their restricted sets.

Online/offline 2-server private keyword search. In private keyword search [CGN98, FIPR05, OI05], a server holds a database $D = \{w_1, \ldots, w_n\}$ of keywords, and a client wants to determine whether a specific keyword is in the database without revealing the keyword to the server. This setting differs from searchable encryption in that the server learns nothing about the client's query, whereas in the searchable encryption framework, information about the client's query (such as whether or not there are any matching results) could be leaked.

In the 2-server variant of this problem [GI14], the database is shared among two servers. The client can send queries to each server independently, and then combine the results of the queries to obtain the answer. We assume moreover that the two servers are non-colluding. Recently, Gilboa and Ishai [GI14] gave a secure solution for the the 2-server variant of the problem that is more efficient than the solutions for 1-server private keyword search, and relies on weaker cryptographic assumptions.

Using a private puncturable PRF, we can construct an online/offline version of the 2-server keyword-search protocol. In an online/offline 2-server private keyword search protocol, there is an "offline" server and an "online" server. The offline server can process the search query before the client has decided its query (for instance, the offline computation can be preformed in a separate setup phase). When the client issues a search query, it only communicates with the online server. The client then combines the response from both servers to learn the result of the query. Our protocol can be seen as a hybrid between the 1-server and 2-server protocols. In the 1-server setting, there is no offline setup component in the protocol, while in the 2-server setting, we require both servers to be online during the query phase.

To implement online/offline 2-server private keyword search using private puncturable PRFs, during the offline (setup) phase, the client generates a master secret key msk for the private puncturable PRF, and sends msk to the offline server. Let $\{0,1\}^m$ be the range of the PRF. For each word $w_i \in D$, the offline server computes $s_i = F(\mathsf{msk}, w_i)$, and returns $s = \bigoplus_{i=1}^{n} s_i$ to the client. Note that all computation in the offline phase is *independent* of the client's search query. In the online phase, after the client has determined its search query w^*, she sends a key sk_{w^*} punctured at w^* to the online server. For each word $w_i \in D$, the online server evaluates sk_{w^*} on w_i to obtain a value t_i. Finally, the online server returns the value $t = \bigoplus_{i=1}^{n} t_i$. To learn the result of the keyword search, the client tests whether $z = s \oplus t$ is the all-zeros string 0^m or not. If $z = 0^m$, then the client concludes $w^* \notin D$; otherwise, the client concludes that $w^* \in D$. To see why, consider the case where $w^* \notin D$, so $w^* \neq w_i$ for all i. By correctness of the punctured PRF, $s_i = t_i$ for all i, in which case $z = 0^m$. Conversely, if $w^* = w_{i^*}$ for some i^*, then for all $i \neq i^*$, $s_i = t_i$. Moreover, security of the PRF implies that s_{i^*} is independent of t_{i^*}. Thus, with overwhelming probability, $s_{i^*} \neq t_{i^*}$ and $z \neq 0^m$.

For the security parameter λ and a dictionary of n keywords, the size of the search tokens sent to the online and offline servers is $O(\lambda \log N)$. The size of the responses from each server is $O(\lambda)$ bits. For single-server private keyword search, Ostrovsky and Skeith [OI05] show how to construct a private keyword search protocol, using homomorphic encryption and a private information retrieval (PIR) protocol. Instantiating the PIR protocol with the scheme of Gentry and Ramzan [GR05] results in a 1-server private keyword search with $O(\lambda + \log N)$ communication, which is optimal. We remark that although our current constructions do not result in a more efficient private keyword search protocol, improved constructions of private puncturable PRFs would have direct implications

for the online/offline 2-server variant of private keyword search.

1.2 Constructing Private Constrained PRFs

In this section, we describe our constructions of private constrained PRFs.

A construction from indistinguishability obfuscation. Indistinguishability obfuscation (iO) [BGI+12, GGH+13b, BGK+14, SW14, GLSW14, Zim15, AB15] is a powerful primitive that has enabled a number of new constructions in cryptography [SW14, BZ14, GGH+13b]. Informally, an indistinguishability obfuscator is a machine that takes as input a program and outputs a second program with the identical functionality, but at the same time, hides some details on how the original program works.

We first show how indistinguishability obfuscation can be used to construct a private constrained PRF for general circuit constraints. Suppose $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a PRF with master secret key $\mathsf{msk} \in \mathcal{K}$. We use F in conjunction with iO to construct a private circuit-constrained PRF. We describe the constrain algorithm. On input a circuit C, the constrain algorithm samples another secret key $\mathsf{sk} \in \mathcal{K}$ and outputs the obfuscation of the following program P:

"On input x, if $C(x) = 1$, output $F(\mathsf{msk}, x)$. Otherwise, output $F(\mathsf{sk}, x)$."

In the above program, note that C, msk, and sk are all hard-coded into the program. Let $\hat{P}$ be the obfuscated program. Evaluation of the PRF using the constrained key corresponds to evaluating the program $\hat{P}(x)$. We see that on all inputs x where $C(x) = 1$, $\hat{P}(x) = F(\mathsf{msk}, x)$, so correctness is immediate.

At a high level, the constrain algorithm generates a "fake" PRF key sk, and the constrained key is just a program that either evaluates the "real" PRF or the fake PRF, depending on the value of $C(x)$. Since the adversary cannot distinguish between the outputs under the real PRF key from those under the fake PRF key, the adversary cannot simply use the input-output behavior of the obfuscated program to learn anything about C. Moreover, in Section 3, we show that if the underlying PRF F is puncturable (not necessarily privately), the indistinguishability obfuscation of the program does in fact hide the constraining circuit C. We note though that for general circuits, our security reduction requires subexponential hardness of iO (and one-way functions). For restricted classes of circuits, such as puncturing, however, we can obtain security from polynomially-hard iO (and one-way functions).

Multilinear maps. Although our construction from indistinguishability obfuscation is clean and simple, it is still interesting to try and construct private constrained PRFs from weaker assumptions. In particular, we give two constructions of private constrained PRFs for more restrictive classes of constraints based on standard assumptions over multilinear maps.

Multilinear maps [BS03, GGH13a, CLT13, GGH14, CLT15] have been successfully applied to many problems in cryptography, most notably in constructing indistinguishability obfuscation [GGH+13b, BGK+14, AGIS14, GLSW14, Zim15, AB15]. Unfortunately, a number of recent attacks [CHL+15, BWZ14, HJ15, CGH+15, CLR15, MF15, Cor15] have invalidated many of the basic assumptions on multilinear maps. However, indistinguishability obfuscation is an example of a setting where the adversary does not have the necessary information to carry out these attacks, and so most of the existing constructions are not known to be broken [CGH+15]. In our first

construction from multilinear maps, we rely on the Multilinear Diffie-Hellman (MDH) assumption [BS03, GGH13a] over prime-order multilinear maps. In our second construction, we rely on the Subgroup Decision assumption [BGN05, GGH13a] as well as a generalization which we call the Multilinear Diffie-Hellman Subgroup Decision (MDHSD) assumption over composite-order multilinear maps.[5] In both cases, our constructions do not require publishing the rerandomization parameters for the multilinear map, nor do they provide enough encodings for the adversary to construct sufficiently many top-level encodings of zero.[6] Thus, our assumptions plausibly hold in existing multilinear map candidates, notably the Garg et al. construction in the prime-order setting [GGH13a], and the Coron et al. construction for the composite-order setting [CLT13]. We also note that starting from iO, it is also possible to construct multilinear maps where the MDH assumption holds [AFH+15].

Two constructions from multilinear maps. Using multilinear maps, we give two constructions of private constrained PRFs: one for the class of bit-fixing constraints, and the other for puncturing. A bit-fixing constraint is described by a pattern $s \in \{0, 1, ?\}^n$. An input $x \in \{0, 1\}^n$ satisfies the constraint if it matches the pattern—that is, for each coordinate i, either $s_i = ?$ or $s_i = x_i$. Our private bit-fixing PRF builds off of the Boneh-Waters bit-fixing PRF [BW13] based on prime-order multilinear maps [BS03, GGH13a]. We give the full construction in Section 4. In Section 5, we give the full construction of our privately puncturable PRF from composite-order multilinear maps. Here, security and privacy are based on the n-MDHSD and Subgroup Decision assumptions.

1.3 Related Work

Recently, Gilboa and Ishai introduce the notion of distributed point functions (DPFs) [GI14, BGI15], which are closely related to private puncturable PRFs. In a DPF, a function Gen takes as input a pair $x, y \in \{0, 1\}^*$ and outputs two keys k_0 and k_1, and a function Eval is such that $\mathsf{Eval}(k_0, x') \oplus \mathsf{Eval}(k_1, x') = 0^{|y|}$ if $x' \neq x$, and $\mathsf{Eval}(k_0, x) \oplus \mathsf{Eval}(k_1, x) = y$. The security of the DPF stipulates that each of the keys, by themselves, appear to be distributed independently of x and y. A DPF is similar to a private puncturable PRF if we view k_0 as the master secret key and k_1 as a secret key punctured at x. However, there are two significant differences: first, the keys k_0 and k_1 need not be PRF keys (in the sense that $F(k_0, \cdot)$ and $F(k_1, \cdot)$ need to be indistinguishable from a truly random function), and second, the keys k_0 and k_1 are generated *together* depending on x, whereas in a puncturable PRF, the master secret key is generated *independently* of x.

Also, Kiayias et al. [KPTZ13] put forth the notion of policy privacy for delegatable PRFs. In a delegatable PRF, a proxy is able to evaluate a PRF on a subset of its domain by using a trapdoor derived from the master secret key, where the trapdoor (constrained key) is constructed based on a policy predicate (circuit constraint) which determines which values in the domain the proxy is able to compute the PRF on. Here, policy privacy refers to the security property that the trapdoor does not reveal the underlying policy predicate. The notion of policy privacy is conceptually similar to our notion of privacy for constrained PRFs, except that the delegatable PRFs which they construct are for policy predicates that describe a consecutive range of PRF inputs. Moreover, this restriction is reflected in their definition of policy privacy, and hence, their notion of privacy is incomparable to ours. However, we note that their delegatable PRF constructions are GGM-based and, thus, more

[5] In Appendix B, we prove this assumption in the generic multilinear map model.

[6] To our knowledge, the existing attacks require one of these two conditions to be true.

efficient than our PRF constructions.

Finally, as discussed earlier, Boyle et al. [BGI14] proposed the notion of constrained signatures (which they call functional signatures). Here, in addition to the master signing key, there are secondary signing keys for functions f which restrict the signer to only being able to construct valid signatures for a range of messages determined by f. They also proposed the notion of function privacy, which intuitively states that a signature constructed from a secondary signing key should not reveal the function associated with the signing key, nor the message that the function was applied to. However, critically, this notion of privacy does not prevent the secondary signing key itself from revealing the function it corresponds to; in this respect, their notion of function privacy is incomparable to our notion of privacy for constrained PRFs.

In Sections 6.1 and 6.2, we also survey the related work on deniable encryption and cryptographic watermarking, respectively.

2 Private Constrained PRFs

In this section, we first review some notational conventions that we use throughout the work, along with the definition of a pseudorandom function (PRF). Then, we define constrained PRFs and the notion of privacy.

2.1 Conventions

For an integer n, we write $[n]$ to denote the set $\{1, \ldots, n\}$. For a finite set S, we write $x \xleftarrow{R} S$ to denote that x is drawn uniformly at random from S. For two finite sets S and T, we write $\mathsf{Funs}(S, T)$ to denote the set of all (well-defined) functions $f : S \to T$. Hence, if $f \xleftarrow{R} \mathsf{Funs}(S, T)$, then for every distinct input $a \in S$, the value $f(a)$ is distributed uniformly and independently in T. We say a function $f(\lambda)$ is negligible in the parameter λ, denoted as $\mathrm{negl}(\lambda)$, if $f(\lambda) = o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. For two families of distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, we write $\mathcal{D}_1 \equiv \mathcal{D}_2$ if the two distributions are identical. We write $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$ if the two distributions are computationally indistinguishable, that is, no efficient algorithm can distinguish $\mathcal{D}_1$ from $\mathcal{D}_2$, except perhaps with negligible probability.

2.2 Pseudorandom Functions

We first review the definition of a pseudorandom function (PRF) [GGM86]. Unless otherwise noted, we will specialize the domain of our PRFs to $\{0,1\}^n$ and the range to $\{0,1\}^m$.

Definition 2.1 (Pseudorandom Function [GGM86])**.** Fix the security parameter λ. A PRF $F : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^m$ with key space $\mathcal{K}$, domain $\{0,1\}^n$, and range $\{0,1\}^m$ is secure if for all efficient algorithms $\mathcal{A}$,

$$\left| \Pr\left[k \xleftarrow{R} \mathcal{K} : \mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1\right] - \Pr\left[f \xleftarrow{R} \mathsf{Funs}(\{0,1\}^n, \{0,1\}^m) : \mathcal{A}^{f(\cdot)}(1^\lambda) = 1\right]\right| = \mathrm{negl}(\lambda).$$

We also review the definition of a constrained PRF [BW13, KPTZ13, BGI14]. Consider a PRF $F : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^m$, and let msk be the master secret key for F. In a constrained PRF, the holder of msk can derive keys sk for some circuit $C : \{0,1\}^n \to \{0,1\}$, such that given sk, the evaluator can compute the PRF on all inputs $x \in \{0,1\}^n$ where $C(x) = 1$. More precisely, we have the following definition.

Definition 2.2 (Constrained PRF [BW13, KPTZ13, BGI14])**.** A constrained PRF for a circuit class $\mathcal{C}$ is a tuple of algorithms $\Pi = (\mathsf{cPRF.Setup}, \mathsf{cPRF.Constrain}, \mathsf{cPRF.ConstrainEval}, \mathsf{cPRF.Eval})$ over the input space $\{0,1\}^n$ and output space $\{0,1\}^m$, with the following properties:

- $\mathsf{cPRF.Setup}(1^\lambda) \to \mathsf{msk}$. On input the security parameter λ, the setup algorithm $\mathsf{cPRF.Setup}$ outputs the master secret key msk.
- $\mathsf{cPRF.Constrain}(\mathsf{msk}, C) \to \mathsf{sk}$. On input the master secret key msk and a circuit $C \in \mathcal{C}$, the constrain algorithm $\mathsf{cPRF.Constrain}$ outputs a secret key sk for the circuit C.
- $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, x) \to y$. On input a secret key sk, and an input $x \in \{0,1\}^n$, the constrained evaluation algorithm $\mathsf{cPRF.ConstrainEval}$ outputs an element $y \in \{0,1\}^m$.
- $\mathsf{cPRF.Eval}(\mathsf{msk}, x) \to y$. On input the master secret key msk and an input $x \in \{0,1\}^n$, the evaluation algorithm $\mathsf{cPRF.Eval}$ outputs an element $y \in \{0,1\}^m$.

Correctness. A constrained PRF is correct for a circuit class $\mathcal{C}$ if $\mathsf{msk} \leftarrow \mathsf{cPRF.Setup}(1^\lambda)$, for every circuit $C \in \mathcal{C}$ and input $x \in \{0,1\}^n$ such that $C(x) = 1$, it is the case that

$$\mathsf{cPRF.ConstrainEval}(\mathsf{cPRF.Constrain}(\mathsf{msk}, C), x) = \mathsf{cPRF.Eval}(\mathsf{msk}, x).$$

Security. In what follows, we describe two security properties for a constrained PRF. The first property is the basic security notion for a constrained PRF and is adapted from the definitions of Boneh and Waters [BW13]. At a high level, this first notion captures the property that given several constrained keys as well as PRF evaluations at points of the adversary's choosing, the output of the PRF on points the adversary cannot compute itself looks random. The second property, which we call privacy, captures the notion that a constrained key does not reveal the associated constraining function. Each security definition is accompanied by an experiment between a challenger and an adversary, along with admissibility restrictions on the power of the adversary.

Definition 2.3 (Experiment $\mathsf{Expt}_b^{\mathsf{cPRF}}$)**.** For the security parameter $\lambda \in \mathbb{N}$, a family of circuits $\mathcal{C}$, and a bit $b \in \{0,1\}$, we define the experiment $\mathsf{Expt}_b^{\mathsf{cPRF}}$ between a challenger and an adversary $\mathcal{A}$, which can make oracle queries of the following types: constrain, evaluation, and challenge. First, the challenger sets $\mathsf{msk} \leftarrow \mathsf{cPRF.Setup}(1^\lambda)$ and samples a function $f \xleftarrow{R} \mathsf{Funs}(\{0,1\}^n, \{0,1\}^m)$ uniformly at random. For $b \in \{0,1\}$, the challenger responds to each oracle query made by $\mathcal{A}$ in the following manner.

- **Constrain oracle.** On input a circuit $C \in \mathcal{C}$, the challenger returns a constrained key $\mathsf{sk} \leftarrow \mathsf{cPRF.Constrain}(\mathsf{msk}, C)$ to $\mathcal{A}$.
- **Evaluation oracle.** On input $x \in \{0,1\}^n$, the challenger returns $y \leftarrow \mathsf{cPRF.Eval}(\mathsf{msk}, x)$.
- **Challenge oracle.** On input $x \in \{0,1\}^n$, the challenger returns $y \leftarrow \mathsf{cPRF.Eval}(\mathsf{msk}, x)$ to $\mathcal{A}$ if $b = 0$, and $y \leftarrow f(x)$ if $b = 1$.

Eventually, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also output by $\mathsf{Expt}_b^{\mathsf{cPRF}}$. Let $\mathsf{Pr}[\mathsf{Expt}_b^{\mathsf{cPRF}}(\mathcal{A}) = 1]$ denote the probability that $\mathsf{Expt}_b^{\mathsf{cPRF}}$ outputs 1 with $\mathcal{A}$.

At a high level, we say that a constrained PRF is secure if no efficient adversaries can distinguish $\mathsf{Expt}_0^{\mathsf{cPRF}}$ from $\mathsf{Expt}_1^{\mathsf{cPRF}}$. However, we must first restrict the set of allowable adversaries. For example, an adversary that makes a constrain query for a circuit $C \in \mathcal{C}$ and a challenge query for a point $x \in \{0,1\}^n$ where $C(x) = 1$ can trivially distinguish the two experiments. Hence, we first define an admissibility criterion that precludes such adversaries.

Definition 2.4 (Admissible Constraining)**.** We say an adversary is *admissible* if the following conditions hold:

- For each constrain query $C \in \mathcal{C}$ and each challenge query $y \in \{0,1\}^n$, $C(y) = 0$.
- For each evaluation query $x \in \{0,1\}^n$ and each challenge query $y \in \{0,1\}^n$, $x \neq y$.

Definition 2.5 (Constrained Security)**.** A constrained PRF Π is secure if for all efficient and admissible adversaries $\mathcal{A}$, the following quantity is negligible:

$$\mathsf{Adv}^{\mathsf{cPRF}}[\Pi, \mathcal{A}] \stackrel{\mathsf{def}}{=} \left| \mathsf{Pr}[\mathsf{Expt}_0^{\mathsf{cPRF}}(\mathcal{A}) = 1] - \mathsf{Pr}[\mathsf{Expt}_1^{\mathsf{cPRF}}(\mathcal{A}) = 1] \right|.$$

Remark 2.6 (Multiple Challenge Queries)**.** In our constructions of constrained PRFs, it will be convenient to restrict the adversary's power and assume that the adversary makes at most one challenge query. As was noted by Boneh and Waters [BW13], a standard hybrid argument shows that any constrained PRF secure against adversaries that make a single challenge oracle query is also secure against adversaries that make Q challenge oracle queries while only incurring a $1/Q$ loss in advantage. Thus, this restricted definition is equivalent to Definition 2.5.

Remark 2.7 (Adaptive Security)**.** We say that a constrained PRF Π is *selectively* secure if for all efficient adversaries $\mathcal{A}$, the same quantity $\mathsf{Adv}^{\mathsf{cPRF}}[\Pi, \mathcal{A}]$ is negligible, but in the security game, the adversary first commits to its challenge query $x \in \{0,1\}^n$ at the start of the experiment. If we do not require the adversary to first commit to its challenge query, then we say that the scheme is *adaptively* (or *fully*) secure. A selectively-secure scheme can be shown to be fully secure using a standard technique called complexity leveraging [BB04] (at the expense of a super-polynomial loss in the security reduction).

Privacy. Next, we give an indistinguishability-based definition for privacy. In the privacy game, the adversary is allowed to submit two circuits C_0, C_1 to the challenger. On each such query, it receives a PRF key constrained to C_b for some fixed $b \in \{0,1\}$. The adversary can also query the PRF at points of its choosing, and its goal is to guess the bit b. We now give the formal definitions.

Definition 2.8 (Experiment $\mathsf{Expt}_b^{\mathsf{cpriv}}$)**.** For the security parameter $\lambda \in \mathbb{N}$, a family of circuits $\mathcal{C}$, and a bit $b \in \{0,1\}$, we define the experiment $\mathsf{Expt}_b^{\mathsf{cpriv}}$ between a challenger and an adversary $\mathcal{A}$, which can make evaluation and challenge queries. First, the challenger obtains $\mathsf{msk} \leftarrow \mathsf{cPRF.Setup}(1^\lambda)$. For $b \in \{0,1\}$, the challenger responds to each oracle query type made by $\mathcal{A}$ in the following manner.

- **Evaluation oracle.** On input $x \in \{0,1\}^n$, the challenger returns $y \leftarrow \mathsf{cPRF.Eval}(\mathsf{msk}, x)$.
- **Challenge oracle.** On input a pair of circuits $C_0, C_1 \in \mathcal{C}$, the challenger returns $\mathsf{sk} \leftarrow \mathsf{cPRF.Constrain}(\mathsf{msk}, C_b)$.

Eventually, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also output by $\mathsf{Expt}_b^{\mathsf{cPRF}}$. Let $\mathsf{Pr}[\mathsf{Expt}_b^{\mathsf{cpriv}}(\mathcal{A}) = 1]$ denote the probability that $\mathsf{Expt}_b^{\mathsf{cpriv}}$ outputs 1.

Roughly speaking, we say that a constrained PRF is private if no efficient adversary can distinguish $\mathsf{Expt}_0^{\mathsf{cpriv}}$ from $\mathsf{Expt}_1^{\mathsf{cpriv}}$. As was the case with constraining security, when formulating the exact definition, we must preclude adversaries that can trivially distinguish the two experiments.

Definition 2.9 (Admissible Privacy)**.** Let $C_0^{(i)}, C_1^{(i)} \in \mathcal{C}$ be the pair of circuits submitted by the adversary on the i^{th} challenge oracle query, and let d be the total number of challenge oracle queries made by the adversary. For a circuit $C \in \mathcal{C}$, define $S(C) \subseteq \{0,1\}^n$ where $S(C) = \{x \in \{0,1\}^n : C(x) = 1\}$. Then, an adversary is *admissible* if:

1. For each evaluation oracle query with input x, and for each $i \in [d]$, it is the case that $C_0^{(i)}(x) = C_1^{(i)}(x)$.
2. For every pair of distinct indices $i, j \in [d]$,

$$S\left(C_0^{(i)}\right) \cap S\left(C_0^{(j)}\right) = S\left(C_1^{(i)}\right) \cap S\left(C_1^{(j)}\right). \tag{2.1}$$

Definition 2.10 (d-Key Privacy)**.** A constrained PRF Π is (adaptively) *d-key private* if for all efficient and admissible adversaries $\mathcal{A}$ that make d challenge oracle queries, the following quantity is negligible:

$$\mathsf{Adv}^{\mathsf{cpriv}}[\Pi, \mathcal{A}] \stackrel{\mathsf{def}}{=} \left|\mathsf{Pr}[\mathsf{Expt}_0^{\mathsf{cpriv}}(\mathcal{A}) = 1] - \mathsf{Pr}[\mathsf{Expt}_1^{\mathsf{cpriv}}(\mathcal{A}) = 1]\right|.$$

Furthermore, we say a constrained PRF is *multi-key* private if it is d-key private for all $d \in \mathbb{N}$.

Remark 2.11 (Admissibility Requirement)**.** We remark that any non-admissible adversary (Definition 2.9) can trivially win the privacy game if the constrained PRF is secure (Definition 2.5). Thus, Definition 2.9 gives the minimal requirements for a satisfiable notion of multi-key privacy for constrained PRFs. Take an adversary $\mathcal{A}$ that makes two challenge queries $(C_0^{(1)}, C_1^{(1)})$ and $(C_0^{(2)}, C_1^{(2)})$. Suppose that for some x, $C_0^{(1)}(x) = 1 = C_0^{(2)}(x)$, but $C_0^{(1)}(x) = 1$ and $C_1^{(2)}(x) = 0$. Let sk_1 and sk_2 be the keys $\mathcal{A}$ receives from the challenger in $\mathsf{Expt}_b^{\mathsf{cpriv}}$. For $i \in \{1,2\}$, the adversary computes $z_i = \mathsf{cPRF.ConstrainEval}(\mathsf{sk}_i, x)$. When $b = 0$, correctness implies that $z_1 = z_2$. When $b = 1$, security of the constrained PRF stipulates that z_2 be independent of z_1, so $z_2 \neq z_1$ with overwhelming probability. The claim follows.

Remark 2.12 (Weaker Notions of Privacy)**.** In some cases, we also consider a weaker notion of privacy where the adversary is not given access to an evaluation oracle in experiment $\mathsf{Expt}_b^{\mathsf{cpriv}}$. While this is a weaker notion of privacy, in all of our candidate applications, a scheme that satisfies this weaker notion suffices.

Puncturable PRFs. A *puncturable* PRF [SW14, KPTZ13, BW13, BGI14] is a special case of a constrained PRF, where the constraining circuit describes a point function, that is, each constraining circuit C_{x^*} is associated with a point $x^* \in \{0,1\}^n$, and $C_{x^*}(x) = 1$ if and only if $x \neq x^*$. More concretely, a puncturable PRF is specified by a tuple of algorithms $\Pi = (\mathsf{cPRF.Setup}, \mathsf{cPRF.Puncture}, \mathsf{cPRF.ConstrainEval}, \mathsf{cPRF.Eval})$, which is identical to the syntax of a constrained PRF with the exception that the algorithm $\mathsf{cPRF.Constrain}$ is replaced with the algorithm $\mathsf{cPRF.Puncture}$.

- $\mathsf{cPRF.Puncture}(\mathsf{msk}, x) \to \mathsf{sk}$. On input the master secret key msk and an input $x \in \{0,1\}^n$, the puncture algorithm $\mathsf{cPRF.Puncture}$ outputs a secret key sk.

The correctness and security definitions (for constrained security and privacy) are analogous to those for private constrained PRFs. We note that no puncturable PRF can be secure or private against an adversary making multiple challenge queries, so only 1-key privacy is achievable.

3 Private Circuit Constrained PRFs from Obfuscation

In this section, we show how to construct multi-key private circuit-constrained PRFs from indistinguishability obfuscation and puncturable PRFs (implied by one-way functions [GGM86, BW13, KPTZ13, BGI14]). First, we review the notion of indistinguishability obfuscation introduced by Barak et al. [BGI+12].

Definition 3.1 (Indistinguishability Obfuscation (iO) [BGI+12, GGH+13b])**.** An indistinguishability obfuscator iO for a circuit class $\{\mathcal{C}_\lambda\}$ is a uniform and efficient algorithm satisfying the following requirements:

- **Correctness.** For all security parameters $\lambda \in \mathbb{N}$, all circuits $C \in \mathcal{C}_\lambda$, and all inputs x, we have that
$$\Pr[C' \leftarrow \mathsf{iO}(C) : C'(x) = C(x)] = 1.$$
- **Indistinguishability.** For all security parameters λ, 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
$$|\Pr[\mathcal{A}(\mathsf{iO}(C_0)) = 1] - \Pr[\mathcal{A}(\mathsf{iO}(C_1)) = 1]| = \mathrm{negl}(\lambda).$$

For general circuit constraints, our construction will require the stronger assumption that the indistinguishability obfuscator and puncturable PRF be secure against subexponential-time adversaries. However, for more restrictive circuit families, such as puncturing, our construction can be shown to be secure assuming the more standard polynomial hardness of iO and the puncturable PRF (Remark D.11). In addition, in Appendix E, we show how to adapt our private circuit-constrained PRF to also obtain a private programmable PRF (formally defined in Appendix C) from (polynomially-hard) iO and one-way functions.

Construction overview. Our starting point is the circuit-constrained PRF by Boneh and Zhandry [BZ14, Construction 9.1]. In the Boneh-Zhandry construction, the master secret key msk is a key for a puncturable PRF, and a constrained key for a circuit $C : \{0,1\}^n \to \{0,1\}$ is an obfuscation of the program that outputs $\mathsf{cPRF.Eval}(\mathsf{msk}, x)$ if $C(x) = 1$ and $\perp$ otherwise. Because the program outputs $\perp$ on inputs x where $C(x) = 0$, simply evaluating the PRF at different points x reveals information about the underlying constraint. In our construction, we structure the program so that on an input x where $C(x) = 0$, the program's output is the output of a different PRF. Intuitively, just by looking at the outputs of the program, it is difficult to distinguish between the output of the real PRF and the output of the other PRF. In Theorem 3.3, we formalize this intuition by showing that our construction provides multi-key privacy.

Construction. Let iO be an indistinguishability obfuscator, and let $\Pi_\mathsf{F} = (\mathsf{F.Setup}, \mathsf{F.Puncture}, \mathsf{F.ConstrainEval}, \mathsf{F.Eval})$ be any puncturable (but not necessarily private) PRF. Our multi-key private circuit-constrained PRF $\Pi_{\mathsf{ioPRF}} = (\mathsf{cPRF.Setup}, \mathsf{cPRF.Constrain}, \mathsf{cPRF.ConstrainEval}, \mathsf{cPRF.Eval})$ is given as follows:

- $\mathsf{cPRF.Setup}(1^\lambda)$. The setup algorithm outputs $\mathsf{msk} \leftarrow \mathsf{F.Setup}(1^\lambda)$.
- $\mathsf{cPRF.Constrain}(\mathsf{msk}, C)$. First, the constrain algorithm computes $\mathsf{msk}' \leftarrow \mathsf{F.Setup}(1^\lambda)$. Then, it outputs an obfuscated program $\mathsf{iO}\left(P\left[C, \mathsf{msk}', \mathsf{msk}\right]\right)$, where $P\left[C, \mathsf{msk}', \mathsf{msk}\right]$ is the following program[7]:

> **Constants**: a circuit $C : \{0,1\}^n \to \{0,1\}$, and master secret keys msk_0, msk_1 for the puncturable PRF $\Pi_\mathsf{F} = (\mathsf{F.Setup}, \mathsf{F.Puncture}, \mathsf{F.ConstrainEval}, \mathsf{F.Eval})$.
>
> On input $x \in \{0,1\}^n$:
>
> 1. Let $b = C(x)$. Output $\mathsf{F.Eval}(\mathsf{msk}_b, x)$.

Figure 1: The program $P_1\left[C, \mathsf{msk}_0, \mathsf{msk}_1\right]$

- $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, x)$. The constrained evaluation algorithm outputs the evaluation of the obfuscated program sk on x.
- $\mathsf{cPRF.Eval}(\mathsf{msk}, x)$. The evaluation algorithm outputs $\mathsf{F.Eval}(\mathsf{msk}, x)$.

Correctness. By definition, the program $P_1[C, \mathsf{msk}', \mathsf{msk}]$ outputs $\mathsf{F.Eval}(\mathsf{msk}, x)$ on all $x \in \{0,1\}^n$ where $C(x) = 1$. Correctness of Π_{ioPRF} immediately follows from correctness of the indistinguishability obfuscator.

Security. We now state our security theorems, but defer their formal proofs to Appendix D.

Theorem 3.2. *Suppose* iO *is an indistinguishability obfuscator and* Π_F *is a selectively-secure puncturable PRF. Then,* Π_{ioPRF} *is selectively secure (Definition 2.5).*

Theorem 3.3. *Suppose* iO *is a indistinguishability obfuscator, and* Π_F *is a selectively-secure puncturable PRF, both secure against subexponential adversaries. Then,* Π_{ioPRF} *is multi-key private (Definition 2.10).*

Proof overview. In the multi-key privacy game, the adversary makes challenge queries of the form (C_0, C_1), and the challenger replies with a constrained key for circuit C_b where $b \in \{0,1\}$ is fixed in the experiment. The constrained key C_b is an obfuscated program that evaluates C_b on the input x, and depending on $C_b(x)$, outputs either the real PRF value at x, or that of a different PRF at x. To show privacy, we show that the obfuscated program with C_0 embedded inside is computationally indistinguishable from that with C_1 embedded inside. Our argument consists of a sequence of hybrid arguments $\mathsf{H}_0, \ldots, \mathsf{H}_{2^n+1}$, where in hybrid H_i, the obfuscated program uses $C_1(x)$ to determine which PRF output to use for the first i inputs in the domain (that is, all $x < i$), and $C_0(x)$ for the remaining inputs. The first and last hybrids are identical to the real experiment where the challenger constrains to C_0, and that where the challenger constrains to C_1. The programs

[7] We pad the program $P\left[C, \mathsf{msk}', \mathsf{msk}\right]$ to the maximum size of any program that appears in the hybrid experiments in the proofs of Theorem 3.2 and 3.3.

given out in each adjacent pair of hybrids differ on a single point, so using puncturing security, we can show that each adjacent pair of hybrids are also computationally indistinguishable. We give the full proof in Appendix D.2.

We note that Theorem 3.3 only requires subexponentially-secure iO if the set of challenge circuits $\{C_0^{(j)}\}_{j\in[d]}$ and $\{C_1^{(j)}\}_{j\in[d]}$ the adversary submits differ on a super-polynomial number of points. In particular, this implies that Π_{ioPRF} is a private puncturable PRF assuming only polynomial hardness of iO and selective-security of Π_{F}. For more details, see Remark D.11.

4 A Private Bit-Fixing PRF

In this section, we construct a constrained PRF for the class of bit-fixing circuits, a notion first introduced in [BW13]. First, a bit-fixing string s is an element of $\{0,1,?\}^n$. We say a bit-fixing string s matches $x \in \{0,1\}^n$ if for all $i \in [n]$, either $s_i = x_i$ or $s_i = ?$. We now define the class of bit-fixing circuits.

Definition 4.1 (Bit-Fixing Circuits [BW13])**.** For a circuit $C : \{0,1\}^n \to \{0,1\}$, a string $s \in \{0,1,?\}^n$ is *bit-fixing* for C if $C(x) = 1$ on precisely the inputs $x \in \{0,1\}^n$ that s matches. The *class of bit-fixing circuits* $\mathcal{C}_{\mathsf{bf}}$ is the class of all circuits $C : \{0,1\}^n \to \{0,1\}$ for which there exists a bit-fixing string for C.

Our bit-fixing construction uses multilinear maps [BS03], which are a generalization of bilinear maps [MOV93, Mil04, BF01]. While constructing ideal multilinear maps remains an open problem, there have been several recent candidates of graded encodings schemes [GGH13a, CLT13, GGH14, CLT15], which is often a suitable substitute for ideal multilinear maps. For ease of presentation, we describe our constructions using the simpler abstraction of ideal multilinear maps. However, we note that we can easily map our constructions to the language of graded encodings using the same techniques as in [BW13, Appendix B]. We begin by defining multilinear maps over prime-order groups. In Appendix A, we also recall the standard ℓ-Multilinear Diffie-Hellman assumption [BS03, GGH13a] over prime-order multilinear maps.

Definition 4.2 (Prime-Order Multilinear Map [BS03, GGH13a, CLT13, GGH14, CLT15])**.** We define a *prime-order multilinear map* to consist of a setup algorithm MMGen along with a map function e, defined as follows.

- $\mathsf{MMGen}(1^\lambda, 1^\ell)$. The setup algorithm MMGen takes as input the security parameter λ and a positive integer ℓ, and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_\ell)$ each of prime order p (for a λ-bit prime p). The algorithm also outputs canonical generators $g_i \in \mathbb{G}_i$ for each $i \in [\ell]$, and the group order p.
- $e(g_1^{a_1}, \ldots, g_1^{a_\ell})$. The map function $e : (\mathbb{G}_1)^\ell \to \mathbb{G}_\ell$ takes as input ℓ elements from $\mathbb{G}_1$ and outputs an element in $\mathbb{G}_\ell$ such that, for all $a_1, \ldots, a_\ell \in \mathbb{Z}_p$,

$$e(g_1^{a_1}, \ldots, g_1^{a_\ell}) = g_\ell^{a_1 a_2 \cdots a_\ell}.$$

Construction overview. Our starting point is the bit-fixing PRF by Boneh and Waters [BW13]. The Boneh-Waters bit-fixing PRF uses a symmetric multilinear map. To provide context, we give a brief description of the Boneh-Waters construction. Let $\{0,1\}^n$ be the domain of the PRF, and let $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_{n+1})$ be a sequence of leveled multilinear groups of prime order p. For each $i \in [n+1]$, let g_i be a canonical generator of $\mathbb{G}_i$; for notational convenience, we will often write $g = g_1$. In the Boneh-Waters construction, they define the multilinear map in terms of a collection of bilinear maps $e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j}$ for each $i, j \in [n]$ where $i + j \leq n + 1$. The master secret key in the Boneh-Waters PRF consists of exponents $\alpha, \{d_{i,0}, d_{i,1}\}_{i\in[n]} \in \mathbb{Z}_p$. For an input $x \in \{0,1\}^n$, the value of the PRF at x is $g_{n+1}^{\alpha \prod_{i\in[n]} d_{i,x_i}}$. A constrained key for a pattern $s \in \{0,1,?\}^n$ consists of a "pre-multiplied" element $g_{1+|S|}^{\alpha \prod_{i\in S} d_{i,s_i}}$, where $S \subseteq [n]$ is the subset of indices where $s_i \neq ?$, along with components $g_1^{d_{i,b}}$ for $i \notin S$ and $b \in \{0,1\}$. While this construction is selectively secure [BW13], it does not satisfy our notion of privacy. By simply inspecting the constrained key and seeing which elements $g_1^{d_{i,b}}$ are given out, an adversary can determine the indices s_i in the pattern s where $s_i = ?$.

A first attempt to make the Boneh-Waters construction private is to publish g^α along with a complete set of group elements $\{g^{d^*_{i,0}}, g^{d^*_{i,1}}\}_{i\in[n]}$ where $d^*_{i,b} = d_{i,b}$ if $s_i = ?$ or $s_i = b$, and otherwise, set $d^*_{i,b} \xleftarrow{R} \mathbb{Z}_p$. By construction, this only permits evaluation of the PRF at the points x that match s. However, this does not yield a secure constrained PRF, since an adversary that sees more than one constrained key can mix and match components from different keys, and learn the value of the PRF at points it could not directly evaluate given any of the individual keys. To prevent mixing and matching attacks in our construction, we rerandomize the elements in the constrained key. We give our construction below.

Construction. For simplicity, we describe the algorithm $\mathsf{cPRF.Constrain}$ as taking as input the master secret key msk and a bit-fixing string $s \in \{0,1,?\}^n$ rather than a circuit $C \in \mathcal{C}$. We define $\Pi_{\mathsf{bfPRF}} = (\mathsf{cPRF.Setup}, \mathsf{cPRF.Constrain}, \mathsf{cPRF.ConstrainEval}, \mathsf{cPRF.Eval})$ as follows.

- $\mathsf{cPRF.Setup}(1^\lambda)$. The setup algorithm runs $\mathsf{MMGen}(1^\lambda, 1^{n+1})$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_{n+1})$ each of prime order p, along with generators $g_i \in \mathbb{G}_i$. As usual, we set $g = g_1$. Next, for $i \in [n]$, it samples $(d_{i,0}, d_{i,1}) \xleftarrow{R} \mathbb{Z}_p^2$, along with a random $\alpha \xleftarrow{R} \mathbb{Z}_p$. It outputs
$$\mathsf{msk} = \left(g, g_{n+1}, \alpha, \{d_{i,0}, d_{i,1}\}_{i\in[n]}\right). \tag{4.1}$$
- $\mathsf{cPRF.Constrain}(\mathsf{msk}, s)$. Let msk be defined as in Equation (4.1) and $s = s_1 s_2 \cdots s_n$. For $i \in [n]$ and $b \in \{0,1\}$, the constrain algorithm samples n random elements $\beta_1 \ldots, \beta_n \xleftarrow{R} \mathbb{Z}_p$ uniformly and independently, along with n random elements $r_1, \ldots, r_n \xleftarrow{R} \mathbb{Z}_p$. Define $\beta_0 = (\beta_1 \beta_2 \cdots \beta_n)^{-1}$. For each $i \in [n]$, define
$$(D_{i,0}, D_{i,1}) = \begin{cases} \left(g^{d_{i,0}}, g^{r_i}\right), & \text{if } s_i = 0 \\ \left(g^{r_i}, g^{d_{i,1}}\right), & \text{if } s_i = 1 \\ \left(g^{d_{i,0}}, g^{d_{i,1}}\right), & \text{if } s_i = ? \end{cases}.$$
It outputs
$$\mathsf{sk} = \left((g^\alpha)^{\beta_0}, \left\{(D_{i,0})^{\beta_i}, (D_{i,1})^{\beta_i}\right\}_{i\in[n]}\right). \tag{4.2}$$

- $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, x)$. Write $\mathsf{sk} = \left(g^{\sigma}, \{g^{\mu_{i,0}}, g^{\mu_{i,1}}\}_{i\in[n]}\right)$, and let $x = x_1 x_2 \cdots x_n$. The constrained evaluation algorithm computes and outputs $y = e(g^{\sigma}, g^{\mu_{1,x_1}}, \ldots, g^{\mu_{n,x_n}})$.
- $\mathsf{cPRF.Eval}(\mathsf{msk}, x)$. Let msk be defined as in Equation (4.1), and let $x = x_1 x_2 \cdots x_n$. The evaluation algorithm outputs $y = g_{n+1}^{\alpha \prod_{i\in[n]} d_{i,x_i}}$.

Correctness and security. We now state the correctness and security theorems for Π_{bfPRF}, but defer the formal proofs to Appendix F.

Theorem 4.3. *The bit-fixing PRF Π_{bfPRF} is correct.*

Theorem 4.4. *Under the $(n+1)$-MDH assumption (Definition A.1), the bit-fixing PRF Π_{bfPRF} is selectively secure.*

Theorem 4.5. *The bit-fixing PRF Π_{bfPRF} is (unconditionally) 1-key private in the model where the adversary does not have access to an evaluation oracle.*

5 A Private Puncturable PRF

Recall from Section 2 that a puncturable PRF is a special class of constrained PRFs where the constraint can be described by a point function that is 1 everywhere except at a single point $s \in \{0,1\}^n$. In this section, we give a construction of a private puncturable PRF using multilinear maps over a composite-order ring. We give an adaptation of Definition 4.2 to the composite-order setting. In Appendix A, we review the standard Subgroup Decision assumption [BGN05, GGH13a] over composite-order groups, and a new assumption which we call ℓ-Multilinear Diffie-Hellman Subgroup Decision (MDHSD). Then, in Appendix B, we show that the ℓ-MDHSD assumption holds in a generic model of composite-order multilinear maps, provided that factoring is hard.

Definition 5.1 (Composite-Order Multilinear Map [BS03, CLT13, CLT15])**.** We define a *composite-order multilinear map* to consist of a setup algorithm CMMGen along with a map function e, defined as follows:

- $\mathsf{CMMGen}(1^{\lambda}, 1^{\ell})$. The setup algorithm CMMGen takes as input the security parameter λ and a positive integer ℓ, and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_\ell)$ each of composite order $N = pq$ (where p, q are λ-bit primes). For each $\mathbb{G}_i$, let $\mathbb{G}_{p,i}$ and $\mathbb{G}_{q,i}$ denote the order-p and order-q subgroups of $\mathbb{G}_i$, respectively. Let $g_{p,i}$ be a canonical generator of $\mathbb{G}_i$ and $g_{q,i}$ be a canonical generator of $\mathbb{G}_{q,i}$. In addition to $\vec{\mathbb{G}}$, the algorithm outputs the generators $g_{p,1}, \ldots, g_{p,\ell}, g_{q,1}, \ldots, g_{q,\ell}$, and the primes p, q.
- $e(g_1^{a_1}, \ldots, g_1^{a_\ell})$. The map function $e : (\mathbb{G}_1)^{\ell} \to \mathbb{G}_\ell$ takes as input ℓ elements from $\mathbb{G}_1$ and outputs an element in $\mathbb{G}_\ell$ such that, for all $a_1, \ldots, a_\ell \in \mathbb{Z}_N$,

$$e(g_1^{a_1}, \ldots, g_1^{a_\ell}) = g_\ell^{a_1 a_2 \cdots a_\ell}.$$

Construction overview. Our construction builds on the Naor-Reingold PRF [NR04], and uses composite-order multilinear maps of order $N = pq$ (Definition 5.1). In our description, we use the same notation for group generators as in Definition 5.1. The master secret key in our construction is a collection of exponents $\{d_{i,0}, d_{i,1}\}_{i\in[n]}$ where each $d_{i,b}$ for all $i \in [n]$ and $b \in \{0,1\}$ is random over $\mathbb{Z}_N$. The value of the PRF at a point $x \in \{0,1\}^n$ is the element $g_{p,n}^{\prod_{i\in[n]} d_{i,x_i}} \in \mathbb{G}_{p,n}$.

Suppose we want to puncture at a point $s = s_1 \cdots s_n \in \{0,1\}^n$. Our constrained key consists of a collection of points $\{D_{i,0}, D_{i,1}\}_{i\in[n]}$. For $b \neq s_i$, we set $D_{i,b} = g_{p,1}^{d_{i,b}} \in \mathbb{G}_{p,1}$ to be an element in the order-p subgroup, and for $b = s_i$, we set the element $D_{i,b} = g_{p,1}^{d_{i,b}} g_{q,1}^{d_{i,b}} \in \mathbb{G}_1$ to be an element in the full group. To evaluate the PRF at a point $x \in \{0,1\}^n$ using the constrained key, one applies the multilinear map to the components D_{i,x_i} in the constrained key. By multilinearity and the fact that the order-p and order-q subgroups are orthogonal, if any of the inputs to the multilinear map lie in the $\mathbb{G}_{p,1}$ subgroup, then the output will be an element of the $\mathbb{G}_{p,n}$ subgroup. Thus, as long as there exists some index $i \in [n]$ such that $x_i \neq s_i$, the constrained key will evaluate to the real PRF output. If however $x = s$, then the constrained key on x will evaluate to an element of the full group $\mathbb{G}_n$. We show in Theorem 5.3 that under the n-MDHSD assumption, this element hides the true value of the PRF at x, which gives puncturing security. Moreover, since the constrained key is just a collection of random elements in either $\mathbb{G}_{p,1}$ or in $\mathbb{G}_1$, the scheme is 1-key private under the Subgroup Decision assumption (Theorem 5.4).

Construction. For simplicity in our description, we describe the $\mathsf{cPRF.Constrain}$ algorithm as taking as input the master secret key msk and a point $s \in \{0,1\}$ to puncture rather than a circuit C. We define $\Pi_{\mathsf{puncPRF}} = (\mathsf{cPRF.Setup}, \mathsf{cPRF.Puncture}, \mathsf{cPRF.ConstrainEval}, \mathsf{cPRF.Eval})$ as follows.

- $\mathsf{cPRF.Setup}(1^\lambda)$. The setup algorithm runs $\mathsf{CMMGen}(1^\lambda, 1^n)$ and outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_n)$, each of composite order $N = pq$, along with the factorization of N, and the generators $g_{p,i}, g_{q,i} \in \mathbb{G}_i$ of the order-p and order-q subgroups of $\mathbb{G}_i$, respectively for all $i \in [n]$. Let $g_1 = g_{p,1} g_{q,1}$ be the canonical generator of $\mathbb{G}_1$. Finally, the setup algorithm samples $2n$ random elements $(d_{1,0}, d_{1,1}), \ldots, (d_{n,0}, d_{n,1}) \xleftarrow{R} \mathbb{Z}_N^2$, and outputs the following master secret key msk:
$$\mathsf{msk} = \left(p, q, g_1, g_{p,1}, g_{p,n}, \{d_{i,0}, d_{i,1}\}_{i\in[n]}\right) \tag{5.1}$$

- $\mathsf{cPRF.Puncture}(\mathsf{msk}, s \in \{0,1\}^n)$. Write $s = s_1 s_2 \cdots s_n$. Let $g_1 = g_{p,1} g_{q,1}$. For each $i \in [n]$, define
$$(D_{i,0}, D_{i,1}) = \begin{cases} (g_1^{d_{i,0}},\ g_{p,1}^{d_{i,1}}), & \text{if } s_i = 0 \\ (g_{p,1}^{d_{i,0}},\ g_1^{d_{i,1}}), & \text{if } s_i = 1 \end{cases}.$$
The algorithm then outputs the constrained key $\mathsf{sk} = \{D_{i,0}, D_{i,1}\}_{i\in[n]}$.

- $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, x)$. Write sk as $\{D_{i,0}, D_{i,1}\}_{i\in[n]}$, and $x = x_1 x_2 \cdots x_n$. The constrained evaluation algorithm outputs $y = e(D_{1,x_1}, \ldots, D_{n,x_n})$.

- $\mathsf{cPRF.Eval}(\mathsf{msk}, x)$. Let msk be defined as in Equation (5.1), and $x = x_1 x_2 \cdots x_n$. The evaluation algorithm outputs $y = g_{p,n}^{\prod_{i\in[n]} d_{i,x_i}}$.

Correctness and security. We now state the correctness and security theorems, but defer the formal analysis to Appendix G.

Theorem 5.2. *The puncturable PRF* Π_{puncPRF} *is correct.*

Theorem 5.3. *Under the n-MDHSD assumption (Definition A.3), the puncturable PRF* Π_{puncPRF} *is selectively secure.*

Theorem 5.4. *Under the Subgroup Decision assumption (Definition A.2), the puncturable PRF* Π_{puncPRF} *is* 1*-key private in the model where the adversary does not have access to an evaluation oracle.*

6 Applications

In Section 1.1, we outlined several applications of private constrained PRFs. Several of our applications (private constrained MACs, restrictable SSE, and online/offline 2-server private keyword search) follow readily from our definitions of private constrained PRFs, and so we do not elaborate further on them. In this section, we give a more formal treatment of how to use private constrained PRFs to construct symmetric deniable encryption and secretly-verifiable message-embedding watermarking of PRFs.

6.1 Symmetric Deniable Encryption

In this section, we give a formal definition of symmetric deniable encryption adapted from those of Canetti et al. [CDNO97]. We then give a construction of our variant of symmetric deniable encryption from private puncturable PRFs. Finally, we conclude with a brief survey of related work in this area.

Definition 6.1 (Symmetric Deniable Encryption [CDNO97, adapted])**.** A symmetric deniable encryption scheme is a tuple of algorithms $\Pi_{\mathsf{DE}} = (\mathsf{DE.Setup}, \mathsf{DE.Encrypt}, \mathsf{DE.Decrypt}, \mathsf{DE.Deny})$ defined over a key space $\mathcal{K}$, a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$ with the following properties:

- $\mathsf{DE.Setup}(1^\lambda) \to (\mathsf{dk}, \mathsf{sk})$. On input the security parameter λ, the setup algorithm outputs a secret key $\mathsf{sk} \in \mathcal{K}$ and a denying key dk.
- $\mathsf{DE.Encrypt}(\mathsf{sk}, m) \to \mathsf{ct}$. On input the secret key $\mathsf{sk} \in \mathcal{K}$ and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext $\mathsf{ct} \in \mathcal{C}$.
- $\mathsf{DE.Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$. On input a secret key $\mathsf{sk} \in \mathcal{K}$ and a ciphertext $\mathsf{ct} \in \mathcal{C}$, the decryption algorithm outputs a message $m \in \mathcal{M}$.
- $\mathsf{DE.Deny}(\mathsf{dk}, \mathsf{ct}) \to \mathsf{sk}'$. On input a denying key dk and a ciphertext ct, the deny algorithm outputs a key $\mathsf{sk}' \in \mathcal{K}$.

The first property we require is that the tuple of algorithms ($\mathsf{DE.Setup}, \mathsf{DE.Encrypt}, \mathsf{DE.Decrypt}$, $\mathsf{DE.Deny}$) should satisfy the usual correctness and semantic security requirements for symmetric encryption schemes [GM82].

Definition 6.2 (Correctness)**.** A symmetric deniable encryption scheme $\Pi_{\mathsf{DE}} = (\mathsf{DE.Setup}, \mathsf{DE.Encrypt}, \mathsf{DE.Decrypt}, \mathsf{DE.Deny})$ is correct if for all messages $m \in \mathcal{M}$, with $(\mathsf{sk}, \mathsf{dk}) \leftarrow \mathsf{DE.Setup}(1^\lambda)$, we have that

$$\Pr\left[\mathsf{DE.Decrypt}(\mathsf{sk}, \mathsf{DE.Encrypt}(\mathsf{sk}, m)) \neq m\right] = \mathrm{negl}(\lambda),$$

where the probability is taken oven the randomness of $\mathsf{DE.Setup}$ and $\mathsf{DE.Encrypt}$.

Definition 6.3 (Semantic Security [GM82, adapted])**.** A symmetric deniable encryption scheme $\Pi_{\mathsf{DE}} = (\mathsf{DE.Setup}, \mathsf{DE.Encrypt}, \mathsf{DE.Decrypt}, \mathsf{DE.Deny})$ is semantically secure if for all efficient adversaries $\mathcal{A}$ and $(\mathsf{sk}, \mathsf{dk}) \leftarrow \mathsf{DE.Setup}(1^\lambda)$,

$$\left|\Pr\left[\mathcal{A}^{\mathcal{O}_0(\mathsf{sk},\cdot,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_1(\mathsf{sk},\cdot,\cdot)}(1^\lambda)\right]\right| = \mathrm{negl}(\lambda),$$

where for $b \in \{0,1\}$, $\mathcal{O}_b(\mathsf{sk}, \cdot, \cdot)$ is an encryption oracle that takes as input two messages $m_0, m_1 \in \mathcal{M}$ and outputs the ciphertext $\mathsf{DE.Encrypt}(\mathsf{sk}, m_b)$.

Finally, we define the notion of deniability for a symmetric deniable encryption scheme. Our notion is similar to that defined in Canetti et al. [CDNO97, Definition 4]. Let $m_1, \ldots, m_n$ be a collection of messages, and let $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ be encryptions of these messages under a symmetric key sk. Suppose without loss of generality that the sender wants to deny to message m_n. Then, the fake secret key sk' output by $\mathsf{DE.Deny}$ should be such that the joint distribution $(\mathsf{sk}', \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ of the fake secret key and the real ciphertexts should look indistinguishable from the joint distribution $(\mathsf{sk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_{n-1}, \mathsf{ct}^*)$ of the real secret key and the real ciphertexts with ct_n substituted for an encryption ct^* of a random message. Our definition captures both the property that the fake secret key looks indistinguishable from a legitimately-generated secret key and that the fake secret key does not reveal any additional information about the denied message m_n beyond what the adversary could already infer. We now proceed with the formal security definition.

Definition 6.4 (Experiment $\mathsf{Expt}_b^{\mathsf{DE}}$)**.** For the security parameter $\lambda \in \mathbb{N}$, we define the experiment $\mathsf{Expt}_b^{\mathsf{DE}}$ between a challenger and an adversary $\mathcal{A}$ as follows:

1. The challenger begins by running $(\mathsf{sk}, \mathsf{dk}) \leftarrow \mathsf{DE.Setup}(1^\lambda)$.
2. The adversary $\mathcal{A}$ chooses a tuple of messages $(m_1, \ldots, m_q) \in \mathcal{M}^q$ and an index $i^* \in [q]$. It gives $(m_1, \ldots, m_q)$ and i^* to the challenger.
3. For each $i \in [q]$, the challenger computes $\mathsf{ct}_i \leftarrow \mathsf{DE.Encrypt}(\mathsf{sk}, m_i)$. Then, depending on the bit b, the challenger does the following:
   - If $b = 0$, the challenger first runs $\mathsf{sk}' \leftarrow \mathsf{DE.Deny}(\mathsf{dk}, \mathsf{ct}_{i^*})$, and then sends $\left(\mathsf{sk}', \{\mathsf{ct}_i\}_{i \in [q]}\right)$ to the adversary.
   - If $b = 1$, the challenger chooses a random message $m^* \xleftarrow{R} \mathcal{M}$, and computes $\mathsf{ct}^* \leftarrow \mathsf{DE.Encrypt}(\mathsf{sk}, m^*)$. It sends $\left(\mathsf{sk}, \{\mathsf{ct}_i\}_{i \neq i^*} \cup \{\mathsf{ct}^*\}\right)$ to the adversary.
4. At the end of the experiment, the adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment. Let $\Pr[\mathsf{Expt}_b^{\mathsf{DE}}(\mathcal{A}) = 1]$ denote the probability that adversary $\mathcal{A}$ outputs 1 in experiment $\mathsf{Expt}_b^{\mathsf{DE}}$.

Definition 6.5. A symmetric deniable encryption scheme $\Pi_{\mathsf{DE}} = (\mathsf{DE.Setup}, \mathsf{DE.Encrypt}, \mathsf{DE.Decrypt}, \mathsf{DE.Deny})$ is deniable if for all efficient adversaries $\mathcal{A}$,

$$\left|\Pr[\mathsf{Expt}_0^{\mathsf{DE}}(\mathcal{A}) = 1] - \Pr[\mathsf{Expt}_1^{\mathsf{DE}}(\mathcal{A}) = 1]\right| = \mathrm{negl}(\lambda).$$

Construction. We now describe our construction of a symmetric deniable encryption scheme from a private puncturable PRF (such as the one from Section 5). Let $\Pi_{\mathsf{cprf}} = (\mathsf{cPRF.Setup}, \mathsf{cPRF.Puncture}, \mathsf{cPRF.ConstrainEval}, \mathsf{cPRF.Eval})$ be a private puncturable PRF with key space $\mathcal{K}$, domain $\{0,1\}^n$ and range $\{0,1\}^\ell$. We use Π_{cprf} to build a symmetric deniable encryption scheme $\Pi_{\mathsf{DE}} = (\mathsf{DE.Setup}, \mathsf{DE.Encrypt}, \mathsf{DE.Decrypt}, \mathsf{DE.Deny})$ with key space $\mathcal{K}$ and message space $\{0,1\}^\ell$ as follows:

- $\mathsf{DE.Setup}(1^\lambda)$. On input the security parameter λ, run $\mathsf{msk} \leftarrow \mathsf{cPRF.Setup}(1^\lambda)$ to obtain the master secret key for the puncturable PRF. Choose a random point $x \xleftarrow{R} \{0,1\}^n$ and run $\mathsf{sk}_x \leftarrow \mathsf{cPRF.Puncture}(\mathsf{msk}, x)$ to obtain a punctured key. Set the symmetric key to $\mathsf{sk} = \mathsf{sk}_x$ and the denying key $\mathsf{dk} = \mathsf{msk}$. Output $(\mathsf{sk}, \mathsf{dk})$.
- $\mathsf{DE.Encrypt}(\mathsf{sk}, m)$. On input the symmetric key sk and a message $m \in \{0,1\}^\ell$, choose a random value $r \xleftarrow{R} \{0,1\}^n$ and output the pair
$$(r, \mathsf{cPRF.ConstrainEval}(\mathsf{sk}, r) \oplus m).$$
- $\mathsf{DE.Decrypt}(\mathsf{sk}, \mathsf{ct})$. On input the symmetric key sk and a ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$, output $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, \mathsf{ct}_0) \oplus \mathsf{ct}_1$.
- $\mathsf{DE.Deny}(\mathsf{dk}, \mathsf{ct})$. On input the denying key $\mathsf{dk} = \mathsf{msk}$ and a ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$, output $\mathsf{cPRF.Puncture}(\mathsf{msk}, \mathsf{ct}_0)$.

Correctness and security. We give the formal proofs for the correctness and security of Π_{DE} in Appendix H.

Theorem 6.6. *The deniable encryption scheme Π_{DE} is correct.*

Theorem 6.7. *If Π_{cprf} is a secure PRF, then Π_{DE} is semantically secure.*

Theorem 6.8. *If Π_{cprf} is a private, selectively-secure PRF, then Π_{DE} is deniable (Definition 6.5).*

Related work. Up until the recent work of Sahai and Waters [SW14], in all previous constructions of deniable encryption, an adversary was able to distinguish real randomness from fake randomness with advantage $1/n$, where n roughly corresponds to the length of a ciphertext in the scheme. Sahai and Waters [SW14] give the first construction of a public-key deniable encryption scheme that achieves the full security notions put forth by Canetti et al.

In their original paper, Canetti et al. also propose a relaxed definition of deniable encryption called *flexibly deniable encryption*. In a flexibly deniable encryption scheme, there are two separate versions of the setup and encryption algorithms: the "honest" version and the "dishonest" version. The guarantee is that if a user encrypts a message m using the dishonest encryption algorithm to obtain a ciphertext ct, it is later able to produce randomness r that makes it look as if ct is an

honest encryption of some arbitrary message m' under randomness r. Using standard assumptions, Canetti et al. give a construction of a sender-deniable flexibly deniable encryption scheme trapdoor permutations: that is, a scheme that gives the sender the ability to later fake the randomness for a particular ciphertext. O'Neill, Peikert, and Waters [OPW11] later extend these ideas to construct a secure flexibly bideniable encryption scheme from lattices. A bideniable encryption scheme is one that allows both the sender and the receiver to fake randomness for a particular message. We note that in a flexibly deniable encryption scheme, only ciphertexts generated via the "dishonest" algorithms can later be opened as honestly-generated ciphertexts of a different message.

Canetti et al. also introduce the notion of deniable encryption with pre-planning. In this setting, the sender can commit ("pre-plan") to deny a message at a later time. The authors show that in the pre-planning model, there are trivial constructions of symmetric deniable encryption schemes if the ciphertext length is allowed to grow with the number of possible openings of a particular message. We note that our construction does not require pre-planning.

Comparison to previous works. There are several differences between our definitions and those of Canetti et al. that we note here. Let c_i be the ciphertext that the sender chooses to deny. First, unlike the definitions proposed in Canetti et al., the sender cannot program the key sk so that c_i decrypts to an arbitrary message of its choosing. Rather, c_i will decrypt to a uniformly random message under the fake key sk'. Thus, our deniable encryption scheme is best suited for scenarios where the messages being encrypted are drawn uniformly from a message space, for instance, when encrypting cryptographic keys. Next, our key generation algorithm outputs a "trapdoor" that the sender (or receiver) uses when generating fake secret keys. This is similar to the flexibly deniable encryption setting when we have two sets of algorithms for key generation and encryption. However, in our construction, there is only one encryption algorithm, and all ciphertexts output by the encryption algorithm can be denied (provided that the sender or receiver has the denying key).

We note also that the Sahai-Waters construction provides strictly stronger guarantees than those achieved by our construction. However, our primary motivation here is to show how private puncturable PRFs can be directly applied to provide a form of symmetric deniable encryption without relying on obfuscation.

6.2 Watermarking PRFs

In this section, we show how to construct watermarkable PRFs from private programmable PRFs.[8] The watermarking scheme we give is secretly-verifiable and supports message embedding [CHN+15], where the marking algorithm can embed a string into the program that can later be extracted by the verification algorithm. We first introduce some definitions for unremovability and unforgeability. The unremovability definitions are adapted from the corresponding definition in [CHN+15] while the unforgeability definitions are adapted from that in [CHV15]. We then show how to construct a watermarkable PRF from any private programmable PRF. Finally, we conclude with a survey of related work.

Definition 6.9 (Watermarkable Family of PRFs [CHN+15, adapted])**.** For the security parameter λ and a message space $\{0,1\}^\ell$, a secretly-verifiable message-embedding watermarking scheme for

[8] Intuitively, a programmable PRF is the same as a puncturable PRF except that the holder of the master secret key can also program the value at the punctured point. We give a formal definition of programmable PRFs in Appendix C.

a PRF with key-space $\mathcal{K}$ is a tuple of algorithms $\Pi = (\mathsf{WM.Setup}, \mathsf{WM.Mark}, \mathsf{WM.Verify})$ with the following properties.

- $\mathsf{WM.Setup}(1^\lambda) \to \mathsf{msk}$. On input the security parameter λ, the setup algorithm outputs the watermarking secret key msk.
- $\mathsf{WM.Mark}(\mathsf{msk}, m) \to (k, C)$. On input the watermarking secret key msk and a message $m \in \{0,1\}^\ell$, the mark algorithm outputs a PRF key $k \in \mathcal{K}$ and a marked circuit C.
- $\mathsf{WM.Verify}(\mathsf{msk}, C') \to m$. On input the master secret key msk and an arbitrary circuit C', the verification algorithm outputs a string $m \in \{0,1\}^\ell \cup \{\perp\}$.

Definition 6.10 (Circuit Similarity)**.** Fix a circuit class $\mathcal{C}$ on n-bit inputs. For two circuits $C, C' \in \mathcal{C}$ and for a non-decreasing function $f : \mathbb{N} \to \mathbb{N}$, we write $C \sim_f C'$ to denote that the two circuits agree on all but an $1/f(n)$ fraction of inputs. More formally, we define

$$C \sim_f C' \qquad \iff \qquad \Pr_{x \xleftarrow{R} \{0,1\}^n} [C(x) \neq C'(x)] \leq 1/f(n).$$

We also write $C \not\sim_f C'$ to denote that C and C' differ on at least a $1/f(n)$ fraction of inputs.

Definition 6.11 (Correctness ([CHN+15, adapted]))**.** Fix the security parameter λ. A watermarking scheme for a PRF with key-space $\mathcal{K}$ and domain $\{0,1\}^n$ is *correct* if for all messages $m \in \{0,1\}^\ell$, $\mathsf{msk} \leftarrow \mathsf{WM.Setup}(1^\lambda)$, $(k, C) \leftarrow \mathsf{WM.Mark}(\mathsf{msk}, m)$, we have that

- The key k is uniformly distributed over the key-space $\mathcal{K}$ of the PRF.
- $C(\cdot) \sim_f F(k, \cdot)$, where $1/f(n) = \mathrm{negl}(\lambda)$.
- $\Pr[\mathsf{WM.Verify}(\mathsf{msk}, C) = m]$ with overwhelming probability.

Watermarking security. We define watermarking security in the context of an experiment $\mathsf{Expt}_{\mathsf{wm}}$ between a challenger and an adversary $\mathcal{A}$, which can make marking oracle and challenge oracle queries.

Definition 6.12 (Experiment $\mathsf{Expt}_{\mathsf{wm}}$)**.** First, the challenger samples $\mathsf{msk} \leftarrow \mathsf{WM.Setup}(1^\lambda)$, and the challenger then responds to each oracle query made by $\mathcal{A}$ in the following manner.

- **Marking oracle.** On input a message $m \in \{0,1\}^\ell$, the challenger returns the pair $(k, C) \leftarrow \mathsf{WM.Mark}(\mathsf{msk}, m)$ to $\mathcal{A}$.
- **Challenge oracle.** On input a message $m \in \{0,1\}^\ell$, the challenger computes $(k, C) \leftarrow \mathsf{WM.Mark}(\mathsf{msk}, m)$ but only returns C to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs a circuit C', and the challenger computes and outputs $\mathsf{WM.Verify}(\mathsf{msk}, C')$, which is also the output of the experiment, denoted as $\mathsf{Expt}_{\mathsf{wm}}(\mathcal{A})$.

Definition 6.13 (Unremoving Admissibility)**.** An adversary $\mathcal{A}$ is *unremoving admissible* if $\mathcal{A}$ only queries the challenge oracle once, and $C(\cdot) \sim_f C'(\cdot)$, where C is the output of the challenge oracle query, C' is the output of $\mathcal{A}$, and $1/f(n) = \mathrm{negl}(\lambda)$.

Definition 6.14 (Unremovability)**.** A watermarking scheme Π is *unremovable* if for all efficient and unremoving admissible adversaries $\mathcal{A}$, if $m \in \{0,1\}^\ell$ is the message submitted by $\mathcal{A}$ to the challenge oracle in $\mathsf{Expt}_{\mathsf{wm}}$, the probability $\mathsf{Pr}[\mathsf{Expt}_{\mathsf{wm}}(\mathcal{A}) \neq m]$ is negligible.

Definition 6.15 (δ-Unforging Admissibility)**.** We say an adversary $\mathcal{A}$ is *δ-unforging admissible* if $\mathcal{A}$ does not make any challenge oracle queries, and for all $i \in [Q]$, $C_i(\cdot) \not\sim_f C'(\cdot)$, where Q is the total number of marking queries the adversary makes, C_i is the output of the marking oracle on the i^{th} query, C' is the circuit output by the adversary, and $1/f(n) \geq \delta$ for all $n \in \mathbb{N}$.

Definition 6.16 (δ-Unforgeability)**.** We say a watermarking scheme Π is *δ-unforgeable* if for all efficient and δ-unforging admissible adversaries $\mathcal{A}$, the probability $\mathsf{Pr}[\mathsf{Expt}_{\mathsf{wm}}(\mathcal{A}) \neq \bot]$ is negligible.

Construction. Fix the security parameter λ, positive integers $n, m, \ell \geq \lambda$, and a positive real value $\delta < 1$, such that $d = \lambda/\delta = \mathrm{poly}(\lambda)$. Let $F : \mathcal{K} \times (\{0,1\}^m)^d \to \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\ell$ be a PRF, and let $\Pi_{\mathsf{pprf}} = (\mathsf{pPRF.Setup}, \mathsf{pPRF.Program}, \mathsf{pPRF.ProgramEval}, \mathsf{pPRF.Eval})$ be a programmable PRF with input space $\{0,1\}^n$ and output space $\{0,1\}^m \times \{0,1\}^\ell$. We construct a watermarking scheme $\Pi_{\mathsf{wm}} = (\mathsf{WM.Setup}, \mathsf{WM.Mark}, \mathsf{WM.Verify})$ for the PRF Π_{pprf} as follows:

- $\mathsf{WM.Setup}(1^\lambda)$. The setup algorithm chooses $\mathsf{k} \xleftarrow{\mathrm{R}} \mathcal{K}$ and $(z_1, \ldots, z_d) \xleftarrow{\mathrm{R}} (\{0,1\}^n)^d$ uniformly at random and outputs $\mathsf{msk} = (\mathsf{k}, z_1, \ldots, z_d)$.
- $\mathsf{WM.Mark}(\mathsf{msk}, m)$. The mark algorithm first parses $\mathsf{msk} = (\mathsf{k}, z_1, \ldots, z_d)$. It generates $k' \leftarrow \mathsf{pPRF.Setup}(1^\lambda)$, and then computes $(x, y, \tau) = F(\mathsf{k}, (\mathsf{pPRF.Eval}(k', z_1), \ldots, \mathsf{pPRF.Eval}(k', z_d)))$ and $v = m \oplus \tau$. Then, it computes $\mathsf{sk}_k \leftarrow \mathsf{pPRF.Program}(k', x, (y, v))$ and outputs (k', C), where $C(\cdot) = \mathsf{pPRF.ProgramEval}(\mathsf{sk}_k, \cdot)$.
- $\mathsf{WM.Verify}(\mathsf{msk}, C, x)$. The verification algorithm first parses $\mathsf{msk} = (\mathsf{k}, z_1, \ldots, z_d)$ and then computes $(x, y, \tau) = F(\mathsf{k}, (C(z_1), \ldots, C(z_d)))$. It then sets $(y', v) = C(x)$ and outputs $v \oplus \tau$ if $y = y'$, and $\bot$ otherwise.

We defer the proofs of correctness and security to Appendix I.1 and I.2.

Theorem 6.17. *If F is a secure PRF and Π_{pprf} is a programmable PRF, then the watermarking scheme Π_{wm} is correct.*

Theorem 6.18. *If F is a secure PRF and Π_{pprf} is a private programmable PRF, then the watermarking scheme Π_{wm} is unremovable.*

Theorem 6.19. *If F is a secure PRF and Π_{pprf} is a programmable PRF, then for $\delta = 1/\mathrm{poly}(\lambda)$, the watermarking scheme Π_{wm} is δ-unforgeable.*

Related work. Recently, Cohen et al. [CHN+15] showed how to construct publicly-verifiable watermarking for puncturable PRFs from indistinguishability obfuscation. They pursue the notion of approximate functionality for watermarking, where the watermarked program is equal to the original program on *most* inputs, as Barak et al. [BGI+12] showed the negative result that the existence of iO implies that watermarking programs for exact functionality is impossible.

Cohen et al. [CHV15] gave a construction from iO which achieves publicly-verifiable watermarking for relaxed notions of unremovability and unforgeability, namely where the adversary can only query

the marking oracle before receiving the challenge program in the unremovability game and moreover, is only allowed to query the challenge oracle once (lunchtime unremovability). In addition, the adversary must submit a forged program which differs on the same set of inputs with respect to all programs submitted to the mark oracle in the unforgeability game.

In a concurrent work to [CHV15], Nishimaki and Wichs [NW15] considered a relaxed notion of watermarking security for message-embedding schemes by considering "selective-message" security, where the adversary must commit to the message to be embedded into the challenge program before interacting with the mark oracle. This limitation is removed in their subsequent work [CHN+15].

Comparison to previous works. In previous constructions of watermarkable PRFs [NW15, CHV15, CHN+15], the authors show how to watermark any family of puncturable PRFs. In contrast, our construction gives a family of watermarkable PRFs from private programmable PRFs. In our construction, we also consider a slightly weaker version of the mark oracle which takes as input a message and outputs a *random* program that embeds the message. This is a weaker notion of security than providing the adversary access to a marking oracle that take as input an (adversarially chosen) program and a message and outputs a watermarked program with the embedded message.[9] In addition, we consider secretly-verifiable watermarking constructions while Cohen et al. and Nishimaki and Wichs focus on publically verifiable constructions. However, despite these limitations, we note that the family of watermarkable PRFs we construct are still sufficient to instantiate the motivating applications for watermarkable PRFs by Nishimaki and Wichs [CHN+15]. In addition, in our model, we achieve full security (as opposed to lunchtime security or selective-message security) for unremovability and strong unforgeability (as opposed to relaxed unforgeability).

7 Conclusions

In this work, we introduce the notion of privacy for constrained PRFs, and give a number of interesting applications including deniable encryption, watermarkable PRFs, and searchable encryption. We also give three constructions of private constrained PRFs: one from indistinguishability obfuscation, and two from concrete assumptions on multilinear maps. Our indistinguishability obfuscation result achieves the strongest notion of privacy for general circuit constraints. Our multilinear map constructions yield private bit-fixing PRFs and private puncturable PRFs.

We leave open the question of constructing private constrained PRFs from simpler and more standard assumptions (such as from lattices or pairing-based cryptography). In particular, is it possible to construct a private puncturable PRF from one-way functions? Currently, our best constructions for private puncturable PRFs require multilinear maps.

Acknowledgments

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

[9] The reason for this stems from the fact that we require PRF security in our security reductions, which cannot be guaranteed when the PRF key is chosen adversarially (as opposed to randomly).

References

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *TCC*, pages 528–556, 2015.

[AFH+15] Martin R. Albrecht, Pooya Farshim, Dennis Hofheinz, Enrique Larraia, and Kenneth G. Paterson. Multilinear maps from obfuscation. *IACR Cryptology ePrint Archive*, 2015:780, 2015.

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. *IACR Cryptology ePrint Archive*, 2014:222, 2014.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

[BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, pages 224–241, 2009.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, pages 213–229, 2001.

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

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519, 2014.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT*, 2015.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, 2014.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[BLR+15] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *EUROCRYPT*, pages 563–594, 2015.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003.

[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*, pages 1–30, 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, pages 280–300. Springer, 2013.

[BWZ14] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. *IACR Cryptology ePrint Archive*, 2014:930, 2014.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, pages 480–499, 2014.

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

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *CRYPTO*, pages 247–266, 2015.

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS*, pages 79–88, 2006.

[CGN98] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. *IACR Cryptology ePrint Archive*, 1998:3, 1998.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, pages 3–12, 2015.

[CHN+15] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. *IACR Cryptology ePrint Archive*, 2015:1096, 2015.

[CHV15] Aloni Cohen, Justin Holmgren, and Vinod Vaikuntanathan. Publicly verifiable software watermarking. *IACR Cryptology ePrint Archive*, 2015:373, 2015.

[CK10] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, pages 577–594, 2010.

[CLR15] Jung Hee Cheon, Changmin Lee, and Hansol Ryu. Cryptanalysis of the new CLT multilinear maps. *IACR Cryptology ePrint Archive*, 2015:934, 2015.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493, 2013.

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *CRYPTO*, pages 267–286, 2015.

[Cor15] Jean-Sébastien Coron. Cryptanalysis of GGH15 multilinear maps, 2015.

[CRV14] Nishanth Chandran, Srinivasan Raghuraman, and Dhinakaran Vinayagamurthy. Constrained pseudorandom functions: Verifiable and delegatable. *IACR Cryptology ePrint Archive*, 2014:522, 2014.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT 2013*. Springer Berlin Heidelberg, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[GGH14] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. *IACR Cryptology ePrint Archive*, 2014:645, 2014.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, 2014.

[GLSW14] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377, 1982.

[Goh03] Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *ICALP*, pages 803–815, 2005.

[HJ15] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. *IACR Cryptology ePrint Archive*, 2015:301, 2015.

[HKKW14] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. *IACR Cryptology ePrint Archive*, 2014:720, 2014.

[HKW14] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. *IACR Cryptology ePrint Archive*, 2014:521, 2014.

[HMW07] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. In *TCC*, pages 362–382, 2007.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *CCS*, pages 669–684, 2013.

[MF15] Brice Minaud and Pierre-Alain Fouque. Cryptanalysis of the new multilinear map over the integers. *IACR Cryptology ePrint Archive*, 2015:941, 2015.

[Mil04] Victor S Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 2004.

[MOV93] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004.

[NW15] Ryo Nishimaki and Daniel Wichs. Watermarking cryptographic programs against arbitrary removal strategies. *IACR Cryptology ePrint Archive*, 2015:344, 2015.

[OI05] Rafail Ostrovsky and William E. Skeith III. Private searching on streaming data. In *CRYPTO*, pages 223–240, 2005.

[OPW11] Adam O'Neill, Chris Peikert, and Brent Waters. Bi-deniable public-key encryption. In *CRYPTO*, pages 525–542, 2011.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In *EUROCRYPT*, pages 439–467, 2015.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, pages 216–226, 1979.

A Hardness Assumptions

We first recall the ℓ-Multilinear Diffie-Hellman assumption, used in proving constrained security of the construction in Section 4.

Definition A.1 (ℓ-Multilinear Diffie-Hellman [BS03, GGH13a])**.** Let $\vec{\mathbb{G}}$ along with generators $g = g_1, \ldots, g_\ell$ and prime p be the output of $\mathsf{MMGen}(1^\lambda, 1^\ell)$, and let $\mathsf{pp} = (\vec{\mathbb{G}}, p, g, g_\ell)$. Let $a_1, \ldots, a_{\ell+1}, r \in \mathbb{Z}_p$ be drawn uniformly and independently at random, and let $z = a_1 a_2 \cdots a_{\ell+1}$. The *$\ell$-Multilinear Diffie-Hellman assumption* states that the following two distributions are computationally indistinguishable:

$$\{\mathsf{pp}, g_\ell^z\} \qquad \text{and} \qquad \{\mathsf{pp}, g_\ell^r\}.$$

We also review the Subgroup Decision assumption over composite-order groups, which we use to prove privacy of the construction in Section 5.

Definition A.2 (Subgroup Decision [BGN05, GGH13a])**.** Let $\vec{\mathbb{G}}$, primes p, q, and generators $g_{p,1}, \ldots, g_{p,\ell},\ g_{q,1}, \ldots, g_{q,\ell}$ be the output of $\mathsf{CMMGen}(1^\lambda, 1^\ell)$. Set $N = pq$ and let $g_1 = g_{p,1}g_{q,1}$ be the canonical generator of $\mathbb{G}_1$. Choose $\gamma \xleftarrow{R} \mathbb{Z}_p$, and let $\mathsf{pp} = (\vec{\mathbb{G}}, N, g_1, g_{p,1}^\gamma)$. Let $r \xleftarrow{R} \mathbb{Z}_N$. The *Subgroup Decision assumption* states that the following two distributions are computationally indistinguishable:

$$\{\mathsf{pp}, g_1^r\} \qquad \text{and} \qquad \{\mathsf{pp}, g_{p,1}^r\}.$$

In addition to the subgroup decision assumption, we require an additional assumption, which we call the ℓ-Multilinear Diffie-Hellman Subgroup Decision (MDHSD) assumption to hold in our composite-order multilinear map. This assumption is used to prove constrained security of our construction in Section 5. In Appendix B, we show that in a generic model of composite-order multilinear maps, the ℓ-MDHSD assumption holds, provided that factoring is hard.

Definition A.3 (ℓ-Multilinear Diffie-Hellman Subgroup Decision)**.** Let $\vec{\mathbb{G}}$, primes p, q, and generators $g_{p,1}, \ldots, g_{p,\ell},\ g_{q,1}, \ldots, g_{q,\ell}$ be the output of running $\mathsf{CMMGen}(1^\lambda, 1^\ell)$. For notational convenience, let $g_1 = g_{p,1}g_{q,1}$, and $g_\ell = g_{p,\ell}g_{q,\ell}$ be canonical generators of $\mathbb{G}_1$ and $\mathbb{G}_\ell$, respectively. Set $N = pq$ and choose $\gamma \xleftarrow{R} \mathbb{Z}_p$. Define $\mathsf{pp} = (\vec{\mathbb{G}}, N, g_1, g_{p,1}^\gamma)$. Choose random exponents $a_1, \ldots, a_\ell \xleftarrow{R} \mathbb{Z}_N$ and $r \xleftarrow{R} \mathbb{Z}_N$. Let $z = \prod_{i \in [\ell]} a_i$. The *$\ell$-Multilinear Diffie-Hellman Subgroup Decision assumption* states that the following two distributions are computationally indistinguishable:

$$\left(\mathsf{pp}, g^{a_1}, \ldots, g^{a_\ell}, g_{p,\ell}^z\right) \qquad \text{and} \qquad \left(\mathsf{pp}, g^{a_1}, \ldots, g^{a_\ell}, g_\ell^r\right).$$

B Generic Security of ℓ-MDHSD Assumption

In this section, we show that the ℓ-Multilinear Diffie-Hellman Subgroup Decision assumption holds in a generic model of composite-order multilinear maps. We adapt the formulation of the generic model of composite-order multilinear maps from [Zim15, BWZ14] to the more restrictive setting of a symmetric multilinear map. This is a natural adaption of the generic multilinear map model for prime-order groups [GGH$^+$13b, BGK$^+$14, BR14, BLR$^+$15], and can be viewed as a generalization of the generic group model [Sho97]. We begin by defining the notion of a generic multilinear map.

B.1 Generic Multilinear Maps

We begin by introducing a generic multilinear map over the composite order ring $\mathbb{Z}_N$ where $N = pq$ is a product of two primes. Roughly speaking, a multilinear map lets us take a scalar $x \in \mathbb{Z}_N$ and produce an encoded version, $\hat{x} = [x]_i$ at some level $i \in \mathbb{N}$. Moreover, the multilinear map supports basic arithmetic (addition and multiplication) on the encodings. We define the notion more precisely below.

Definition B.1 (Multilinear Map ([BS03, GGH13a, CLT13, GGH14, CLT15]))**.** A multilinear map over a ring of composite order $N = pq$, where p and q are prime, supports the following operations. Each operation ($\mathsf{GMM.Setup}$, $\mathsf{GMM.Add}$, $\mathsf{GMM.Mult}$, $\mathsf{GMM.ZeroTest}$, $\mathsf{GMM.Encode}$) is implemented by an efficient randomized algorithm.

- The setup procedure takes as input the degree of multilinearity ℓ, and the security parameter λ (in unary). It first samples two $O(\lambda)$-bit primes p, q and sets $N = pq$. It produces as output the public parameters pp (which include the modulus N) and secret evaluation parameters sk:

$$\mathsf{GMM.Setup}(1^\lambda, 1^\ell) \rightarrow (\mathsf{pp}, \mathsf{sk}).$$

- For each level $i \leq \ell$, and each scalar $x \in \mathbb{Z}_N$, there is a set of strings $[x]_i \subseteq \{0,1\}^*$, i.e., the set of all valid encodings of x at level i.[10] From here on, we will abuse notation to write $[x]_i$ to stand for any element of $[x]_i$ (i.e., any valid encoding of x at level i).

- Elements at the same level $i \leq \ell$ can be added, with the result also encoded at i:

$$\mathsf{GMM.Add}(\mathsf{pp}, [x]_i, [y]_i) \rightarrow [x+y]_i.$$

- Elements at two levels $i_1, i_2 \leq \ell$ can be multiplied, with the result encoded at the sum of the two levels, provided that their sum does not exceed ℓ:

$$\mathsf{GMM.Mult}(\mathsf{pp}, [x]_{i_1}, [y]_{i_2}) \rightarrow \begin{cases} [xy]_{i_1+i_2} & \text{if } i_1 + i_2 \leq \ell \\ \perp & \text{otherwise.} \end{cases}$$

- Elements at the top level ℓ can be *zero-tested*:

$$\mathsf{GMM.ZeroTest}(\mathsf{pp}, [x]_i) \rightarrow \begin{cases} \text{“zero”} & \text{if } i = \ell \text{ and } x = 0 \in \mathbb{Z}_N \\ \text{“nonzero”} & \text{otherwise.} \end{cases}$$

- Using the secret parameters, one can generate a representation of a given scalar $x \in \mathbb{Z}_N$ at any level $i \leq \ell$:

$$\mathsf{GMM.Encode}(\mathsf{sp}, x, i) \rightarrow [x]_i.$$

We briefly describe how the composite-order multilinear maps from Definition 5.1 satisfy this abstract schema. In Definition 5.1, the algorithm CMMGen corresponds to the $\mathsf{GMM.Setup}$ algorithm. The algorithm $\mathsf{CMMGen}(1^\lambda, 1^\ell)$ outputs a sequence of groups $\vec{\mathbb{G}} = (\mathbb{G}_1, \ldots, \mathbb{G}_\ell)$, each of composite order N. For each $i \in [\ell]$, it also outputs a canonical generators g_i for each group $\mathbb{G}_i$, as well as canonical generators of each subgroup $\mathbb{G}_{p,i}$ and $\mathbb{G}_{q,i}$. The public parameters then consist of the descriptions of the groups $\mathbb{G}_1, \ldots, \mathbb{G}_n$, the modulus N, and the description of the multilinear map e. The secret parameters include the canonical generators g_i for each group $\mathbb{G}_i$, the generators for each of the subgroups $\mathbb{G}_{p,i}$ and $\mathbb{G}_{q,i}$, as well as the factors p, q of N.

An encoding of a ring element $x \in \mathbb{Z}_N$ at level $i \in [\ell]$ is just the group element g_i^x. Addition of encodings corresponds to evaluating the group operation. Zero testing corresponds to checking that the group element encodes the identity element. In Definition 5.1, the multilinear map only takes ℓ level-1 encodings and produces a level-ℓ encoding of their product. However, this limitation is just for simplicity of presentation, and we can easily formulate a more general definition where CMMGen outputs a collection of bilinear maps $e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \rightarrow \mathbb{G}_{i+j}$ for all $i, j \in [\ell]$ where $i + j \leq \ell$. Under this definition, multiplication in the generic multilinear map model corresponds to applying the appropriate bilinear map.

[10] To be more precise, we define $[x]_i = \{\chi \in \{0,1\}^* : \mathsf{GMM.IsEncoding}(\mathsf{pp}, \chi, x, i)\}$, where the predicate $\mathsf{GMM.IsEncoding}$ is specified by the concrete instantiation of the multilinear map. In general, the predicate $\mathsf{GMM.IsEncoding}$ is not necessarily efficiently decidable—and indeed, for the security of the multilinear map, it should not be.

B.2 The Generic Multilinear Map Model

To define security for multilinear maps, we now define a generic model, represented by a stateful oracle $\mathcal{M}$, that captures the multilinear map functionality. We say a scheme that uses multilinear maps is "secure in the generic multilinear map model" if, for any concrete adversary breaking the real scheme, there is an ideal adversary breaking a modified scheme in which every access to the multilinear map operations (both by the construction and by the adversary) is replaced by access to a stateful oracle $\mathcal{M}$ which performs the corresponding arithmetic operations internally. Our definition of the oracle $\mathcal{M}$ is essentially the same as in other works which use the generic composite-order multilinear map model [Zim15, BWZ14]). We define the oracle formally as follows.

Definition B.2 (Ideal Multilinear Map Oracle ([GGH$^+$13b, BGK$^+$14, BLR$^+$15, Zim15]))**.** A *generic multilinear map oracle* is a stateful oracle $\mathcal{M}$ that responds to queries as follows.

- On a query $\mathsf{GMM.Setup}(1^\lambda, 1^\ell)$, the oracle will generate two primes p, q and set $N = pq$ as in the real setup procedure. Then, it will generate a value sp as a fresh nonce (i.e., distinct from any previous choices) uniformly at random from $\{0,1\}^\lambda$, set $\mathsf{pp} = N$, and return $(\mathsf{pp}, \mathsf{sp}, p, q)$. It will also store the values generated, initialize an internal table $T \leftarrow \{\}$ (to store "handles", as described below), and set the internal state so that subsequent $\mathsf{GMM.Setup}$ queries fail.

- On a query $\mathsf{GMM.Encode}(k, x, i)$, where $k \in \{0,1\}^\lambda$, $x \in \mathbb{Z}_N$, and $i \in \mathbb{N}$, the oracle will check that $k = \mathsf{sp}$ and $i \leq \ell$ (returning $\perp$ if the check fails). If the check passes, the oracle will generate a fresh nonce ("handle") h uniformly at random from $\{0,1\}^\lambda$, add the entry $h \mapsto (x, i)$ to the table T, and return h.

- On a query $\mathsf{GMM.Add}(k, h_1, h_2)$, where $k, h_1, h_2 \in \{0,1\}^\lambda$, the oracle will check that $k = \mathsf{pp}$, and that the handles h_1, h_2 are present in its internal table T, and are mapped to values, resp., (x_1, i_1) and (x_2, i_2) such that $i_1 = i_2 = i \leq \ell$ (returning $\perp$ if the check fails). If the check passes, the oracle will generate a fresh handle h uniformly at random from $\{0,1\}^\lambda$, set $x \leftarrow x_1 + x_2 \in \mathbb{Z}_N$, add the entry $h \mapsto (x, i)$ to the table T, and return h.

- On a query $\mathsf{GMM.Mult}(k, h_1, h_2)$, where $k, h_1, h_2 \in \{0,1\}^\lambda$, the oracle will check that $k = \mathsf{pp}$, and that the handles h_1, h_2 are present in its internal table T, and are mapped to values, resp., (x_1, i_1) and (x_2, i_2) such that $i_1 + i_2 \leq \ell$ (returning $\perp$ if the check fails). If the check passes, the oracle will set $x \leftarrow x_1 \cdot x_2 \in \mathbb{Z}_N$, generate a fresh handle h uniformly at random from $\{0,1\}^\lambda$, add the entry $h \mapsto (x, i_1 + i_2)$ to the table T, and return h.

- On a query $\mathsf{GMM.ZeroTest}(k, h)$, where $k, h \in \{0,1\}^\lambda$, the oracle will check that $k = \mathsf{pp}$, and that the table T contains an entry $h \mapsto (x, \ell)$ (immediately returning $\perp$ if the check fails). If the check passes, the oracle will return "zero" if $x = 0 \in \mathbb{Z}_N$, and "nonzero" otherwise.

Remark B.3 (Oracle Queries Referring to Formal Polynomials)**.** Although the generic multilinear map oracle is defined formally in terms of "handles" (Definition B.2), it is usually more intuitive to regard each oracle query as *referring* to a formal query polynomial. The formal variables are specified by the expressions initially supplied to the $\mathsf{GMM.Encode}$ procedure (as determined by the details of the construction), and the adversary can construct terms that refer to new polynomials by making oracle queries for the generic-model ring operations $\mathsf{GMM.Add}$, $\mathsf{GMM.Mult}$. Rather than

operating on a "handle", each valid $\mathsf{GMM.ZeroTest}$ query refers to a formal query polynomial[11] encoded at the top level ℓ. The result of the query is "zero" precisely if the polynomial evaluates to zero, when its variables are instantiated with the joint distribution over their values in $\mathbb{Z}_N$, generated as in the real security game. For the full formal description, we refer the reader to [Zim15, Appendix B].

B.3 Generic Security of the ℓ-MDHSD Assumption

In this section, we show that assuming the hardness of factoring, the ℓ-Multilinear Diffie-Hellman Subgroup Decision assumption holds in the generic multilinear map model (Definition B.2). We begin by stating the hardness of factoring assumption.

Definition B.4 (Factoring Assumption)**.** For $a, b \in \mathbb{N}$, let $\mathsf{Primes}[a, b]$ be the set of prime numbers in the interval $[a, b]$. Fix the security parameter $\lambda \in \mathbb{N}$. The *factoring assumption* states that for all efficient adversaries $\mathcal{A}$,

$$\Pr\left[p, q \xleftarrow{R} \mathsf{Primes}[2^\lambda, 2^{\lambda+1}] : \mathcal{A}(pq) \in \{p, q\}\right] = \mathrm{negl}(\lambda).$$

In our analysis, we will also make use of the Schwartz-Zippel algorithm [Sch80, Zip79] for polynomial identity testing. In particular, we will require the following simple corollary to the Schwartz-Zippel lemma:

Fact B.5. Let $N = pq$ be a product of two distinct primes p, q. Then a multivariate polynomial of total degree d has at most d^2 roots over $\mathbb{Z}_N$.

Proof. Fix a multivariate polynomial p of degree d over $\mathbb{Z}_N$. By the Chinese Remainder Theorem, each root of p must be a root modulo p and modulo q. By the Schwartz-Zippel lemma, p has at most d roots over $\mathbb{Z}_p$ and at most d roots over $\mathbb{Z}_q$. Thus, there are at most d^2 tuples that are roots over both $\mathbb{Z}_p$ and $\mathbb{Z}_q$. □

Next, we formulate the ℓ-Multilinear Diffie-Hellman Subgroup Decision assumption in the generic model of multilinear maps. For $b \in \{0, 1\}$, we define the experiment $\mathsf{Expt}_b^{\mathsf{MSD}}$ as follows.

Definition B.6 (Experiment $\mathsf{Expt}_b^{\mathsf{MSD}}$)**.** Fix the security parameter λ. For $b \in \{0, 1\}$, we define an experiment $\mathsf{Expt}_b^{\mathsf{MSD}}$ between a challenger and an adversary $\mathcal{A}$. For notational convenience, we write $[x]_k$ to denote a level-k encoding of an element $x \in \mathbb{Z}_N$. Then, the challenger proceeds as follows:

1. Initialize the generic multilinear map oracle by invoking $\mathsf{GMM.Setup}(1^\lambda, 1^\ell)$ to obtain the public parameters $\mathsf{pp} = N$, the secret parameters sp, and the factors p, q of N.
2. Choose random values $a_1, \ldots, a_\ell, r \xleftarrow{R} \mathbb{Z}_N$, and $\gamma \xleftarrow{R} \mathbb{Z}_p$. Let $z \in \mathbb{Z}_N$ be such that $z = \prod_{i=1}^{\ell} a_i \pmod p$ and $z = 0 \pmod q$. Set $g = 1$ and let $g_p \in \mathbb{Z}_N$ be the unique value where $g_p = \gamma \pmod p$ and $g_p = 0 \pmod q$. For each $i \in [\ell]$, invoke $\mathsf{GMM.Encode}(\mathsf{sp}, a_i, 1)$ to obtain a level-1 encoding $\hat{a}_i = [a_i]_1$. Similarly, construct level-1 encodings $\hat{g} = [g]_1$ and $\hat{g}_p = [g_p]_1$. Finally, invoke $\mathsf{GMM.Encode}(\mathsf{sp}, z, \ell)$ and $\mathsf{GMM.Encode}(\mathsf{sp}, r, \ell)$ to obtain top-level encodings $\hat{z} = [z]_\ell$ and $\hat{r} = [r]_\ell$.

[11] To represent a query polynomial concretely, we can use an arithmetic circuit—and thus, for instance, we can still perform efficient manipulations on query polynomials that have been subjected to repeated squaring.

3. If $b = 0$, set $\hat{T} = \hat{z}$. Otherwise, set $\hat{T} = \hat{r}$. Send $(\mathsf{pp}, \hat{g}, \hat{g}_p, \hat{a}_1, \ldots, \hat{a}_\ell, \hat{T})$ to $\mathcal{A}$.

The adversary $\mathcal{A}$ is then given oracle access to the algorithms $\mathsf{GMM.Add}$, $\mathsf{GMM.Mult}$, and $\mathsf{GMM.ZeroTest}$. At the end of the experiment, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. Let $\mathsf{Expt}_b^{\mathsf{MSD}}(\mathcal{A})$ be the random variable corresponding to the output of $\mathcal{A}$.

Theorem B.7. *If factoring is hard (Definition B.4), then for all efficient adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathsf{Expt}_0^{\mathsf{MSD}}(\mathcal{A}) = 1] - \Pr[\mathsf{Expt}_1^{\mathsf{MSD}}(\mathcal{A}) = 1]\right| = \mathrm{negl}(\lambda).$$

Proof. We begin by charactering the formal polynomials $\mathcal{A}$ is able to submit to the zero-test oracle in each experiment $\mathsf{Expt}_b^{\mathsf{MSD}}$ for $b \in \{0,1\}$.

Lemma B.8. *Fix $b \in \{0,1\}$ and let $\mathcal{A}$ be an efficient adversary in the generic model of multilinear maps (Definition B.2). Consider a formal polynomial z (Remark B.3) produced by $\mathcal{A}$ in $\mathsf{Expt}_b^{\mathsf{MSD}}(\mathcal{A})$ at the top-level ℓ, over the formal variables $\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{g}, \hat{T}$ in $\mathsf{Expt}_b^{\mathsf{MSD}}(\mathcal{A})$. Then, $z \equiv f_\ell(\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{g}) + \alpha\hat{T}$, where f_ℓ is a formal polynomial of degree exactly ℓ and $\alpha \in \mathbb{Z}$ is a scalar.*

Proof. In $\mathsf{Expt}_b^{\mathsf{MSD}}$, the only encodings available to the adversary are level-1 encodings $\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{g}$, and a single level-$\ell$ encoding $\hat{T}$. Consider a monomial v encoded at the top level. By construction, v is either a scalar multiple of $\hat{T}$, or it is a product of exactly ℓ level-1 encodings. The lemma follows. □

We now proceed via a hybrid argument. For each $b \in \{0,1\}$, we define the following three hybrid experiments:

- $\mathsf{H}_0^{(b)}$: This is the real experiment $\mathsf{Expt}_b^{\mathsf{MSD}}$.
- $\mathsf{H}_1^{(b)}$: Same as $\mathsf{H}_0^{(b)}$, except when $\mathcal{A}$ makes a zero-test query $\mathsf{GMM.ZeroTest}(k, h)$ to the generic multilinear map oracle $\mathcal{M}$, if $k = \mathsf{pp}$, and h refers to a formal polynomial $z \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{g}, \hat{T}]$ at the top level ℓ, the challenger first writes z as $f_\ell(\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{g}) + \alpha\hat{T}$, where f_ℓ is a formal polynomial of degree ℓ (Lemma B.8). Then, the challenger checks if any of the following conditions are satisfied:

  1. If $\gcd(N, \alpha) \in \{p, q\}$.
  2. If $\gcd(N, z'(a_1, \ldots, a_\ell, g_p, T)) \in \{p, q\}$ where $z' \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{T}]$ is the polynomial z with the substitution $\hat{g} \mapsto 1$, and $a_1, \ldots, a_\ell, g_p, T \xleftarrow{\mathrm{R}} \mathbb{Z}_N$.
  3. If $\gcd(N, f'_\ell(a_1, \ldots, a_\ell, g_p)) \in \{p, q\}$, where $f'_\ell \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p]$ is the polynomial f_ℓ with the substitution $\hat{g} \mapsto 1$, and $a_1, \ldots, a_\ell, g_p \xleftarrow{\mathrm{R}} \mathbb{Z}_N$.
  4. If $\gcd(N, f''_\ell(a_1, \ldots, a_\ell)) \in \{p, q\}$, where $f''_\ell \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell]$ is the polynomial f'_ℓ with the substitution $\hat{g}_p \mapsto 0$, and $a_1, \ldots, a_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_N$.

  If any of these conditions are satisfied, then the challenger responds with $\mathsf{GMM.ZeroTest}(k, h)$. Otherwise, it uses the following decision procedure:

  1. Let $z' \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{T}]$ be the query polynomial z with the substitution $\hat{g} \mapsto 1$. Run a Schwartz-Zippel test to determine if $z' \equiv 0 \pmod{N}$. If so, the challenger responds with "zero." Otherwise, continue.

2. Respond with "not-zero."

- $\mathsf{H}_2^{(b)}$: Same as $\mathsf{H}_1^{(b)}$, except if any of the conditions are satisfied, the challenger aborts the experiments and outputs Bad.

As usual, we write $\mathsf{H}_i^{(b)}(\mathcal{A})$ to denote the output of $\mathcal{A}$ when interacting in experiment $\mathsf{H}_i^{(b)}$.

Lemma B.9. *For all adversaries $\mathcal{A}$, $\left|\Pr[\mathsf{H}_0^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{H}_1^{(0)}(\mathcal{A}) = 1]\right|$ is negligible.*

Proof. By construction, the setup procedure, as well as the responses to oracle queries $\mathsf{GMM.Add}$, $\mathsf{GMM.Mult}$ are handled identically in $\mathsf{H}_0^{(0)}$ and $\mathsf{H}_1^{(0)}$. Thus, it suffices to argue that the distribution of zero-test responses are statistically indistinguishable between hybrids $\mathsf{H}_0^{(0)}$ and $\mathsf{H}_1^{(0)}$. Consider a valid zero-test query for a polynomial $z \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{g}, \hat{T}]$ at the top-level. If any of the conditions enumerated in the description of hybrid $\mathsf{H}_1^{(0)}$ are met, then the challenger's response to the zero-test query is identical in hybrids $\mathsf{H}_0^{(0)}$ and $\mathsf{H}_1^{(0)}$. We consider three cases. In the following analysis, define the polynomials z', f', and f'' as in the description of hybrid $\mathsf{H}_1^{(b)}$. We show that in each case, either one of the conditions in $\mathsf{H}_1^{(0)}$ holds, or the distribution of the response to the zero-test query is statistically indistinguishable between hybrids $\mathsf{H}_0^{(0)}$ and $\mathsf{H}_1^{(0)}$.

- Suppose $z' \equiv 0 \pmod{N}$. Since z' is identically zero over $\mathbb{Z}_N$, when its formal variables are instantiated with the real values in $\mathsf{H}_0^{(0)}$, z' will evaluate to 0. In hybrid $\mathsf{H}_1^{(0)}$, on input the polynomial z', the Schwartz-Zippel test will also output "zero," so the distribution of responses in $\mathsf{H}_0^{(0)}$ and $\mathsf{H}_1^{(0)}$ are identically distributed in this case.

- Suppose $z' \not\equiv 0 \pmod{p}$. Consider the distribution of values in $\mathsf{H}_0^{(0)}$. In hybrid $\mathsf{H}_0^{(0)}$, $\hat{T} = \prod_{i=1}^{\ell} \hat{a}_i \pmod{p}$ and $\hat{T} = 0 \pmod{q}$. Let $z'' \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p]$ be the polynomial z with $\hat{T}$ substituted in terms of its definition in $\mathsf{H}_0^{(0)}$. Consider the distribution of responses in the simulation. We consider three possibilities:

  - Suppose $z'' \not\equiv 0 \pmod{p}$. Since the variables $\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p$ in the real distribution are uniform over $\mathbb{Z}_p$, we conclude by the Schwartz-Zippel lemma that the polynomial z'' evaluates to non-zero when instantiated with the values in $\mathsf{H}_0^{(0)}$ with overwhelming probability. Thu the zero-test oracle in $\mathsf{H}_0^{(0)}$ responds with "non-zero" with overwhelming probability. In $\mathsf{H}_1^{(0)}$, the challenger always responds "non-zero," so the distribution of responses between hybrids $\mathsf{H}_0^{(0)}$ and $\mathsf{H}_1^{(0)}$ are statistically indistinguishable in this case.
  - Suppose $z'' \equiv 0 \pmod{p}$ and $f'_\ell \equiv 0 \pmod{p}$. We consider two possibilities:
    * Suppose $f'_\ell \equiv 0 \pmod{q}$. In hybrid $\mathsf{H}_0^{(0)}$ and $\mathsf{H}_1^{(0)}$, $z'' = f'_\ell + \alpha \prod_{i=1}^{\ell} \hat{a}_i \pmod{p}$, and by assumption, $z'' \equiv 0 \pmod{p}$ and $f'_\ell \equiv 0 \pmod{p}$. Thus, $\alpha = 0 \pmod{p}$. If $\alpha \neq 0 \pmod{N}$, then $\gcd(N, \alpha) = p$, and condition 1 holds. Otherwise, if $\alpha = 0 \pmod{N}$ and $f'_\ell \equiv 0 \pmod{q}$, then $f'_\ell \equiv 0 \pmod{N}$. In this case, $z' \equiv f'_\ell + \alpha\hat{T} \equiv 0 \pmod{N}$, which is covered by the first case above.
    * Suppose $f'_\ell \not\equiv 0 \pmod{q}$. By the Schwartz-Zippel lemma, with overwhelming probability, $f'_\ell(a_1, \ldots, a_\ell, g_p) \neq 0 \pmod{q}$ for a random choice of $a_1, \ldots, a_\ell, g_p$ in $\mathbb{Z}_N$. Since $f'_\ell \equiv 0 \pmod{p}$, we conclude that $\gcd(N, f'_\ell(a_1, \ldots, a_\ell, g_p)) = p$ with overwhelming probability. Thus, condition 3 holds.

– Suppose $z'' \equiv 0 \pmod{p}$ and $f'_\ell \not\equiv 0 \pmod{p}$. In hybrid $\mathsf{H}_0^{(0)}$ and $\mathsf{H}_1^{(0)}$, this means that $f'_\ell \equiv -\alpha \prod_{i=1}^{\ell} \hat{a}_i \pmod{p}$. Let $f''_\ell \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell]$ be f'_ℓ with the substitution $\hat{g}_p \mapsto 0$. Since $f'_\ell \pmod{p}$ is independent of $\hat{g}_p$, it must be the case that $f''_\ell \equiv f'_\ell \pmod{p}$. We now consider two possibilities:

  * Suppose $f''_\ell \equiv 0 \pmod{q}$. Consider the value of $f''_\ell(a_1, \ldots, a_\ell)$ where $a_1, \ldots, a_\ell$ are uniform in $\mathbb{Z}_N$. Since $f''_\ell \not\equiv 0 \pmod{p}$, by the Schwartz-Zippel lemma, with overwhelming probability, $f''_\ell(a_1, \ldots, a_\ell) \neq 0 \pmod{p}$. Then, $\gcd(N, f''_\ell(a_1, \ldots, a_\ell)) = q$, and condition 4 holds.
  * Suppose $f''_\ell \not\equiv 0 \pmod{q}$. In the real distribution, $\hat{a}_1, \ldots, \hat{a}_\ell$ are sampled uniformly from $\mathbb{Z}_N$. By the Schwartz-Zippel lemma, with overwhelming probability $f''_\ell(\hat{a}_1, \ldots, \hat{a}_\ell) \neq 0 \pmod{q}$. In hybrid $\mathsf{H}_1^{(0)}$, the challenger always replies with "non-zero," so the two distributions are statistically indistinguishable.

- Suppose $z' \equiv 0 \pmod{p}$ and $z' \not\equiv 0 \pmod{q}$. Consider the value $z'(a_1, \ldots, a_\ell, g_p, T)$ where $a_1, \ldots, a_\ell, g_p, T \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. Let $y = z'(a_1, \ldots, a_\ell, g_p, T)$. Since $z' \not\equiv 0 \pmod{q}$, by the Schwartz-Zippel lemma over $\mathbb{Z}_q$, $y \neq 0 \pmod{q}$ with overwhelming probability. However, since $z' \equiv 0 \pmod{p}$, $y = 0 \pmod{p}$. In this case, $\gcd(N, y) = p$, and condition 2 holds. □

Lemma B.10. *If factoring is hard (Definition B.4), then for all efficient adversaries $\mathcal{A}$ and all $b \in \{0,1\}$, $\left|\Pr[\mathsf{H}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{H}_2^{(b)}(\mathcal{A}) = 1]\right|$ is negligible.*

Proof. Suppose $\mathcal{A}$ is able to distinguish hybrids $\mathsf{H}_1^{(b)}$ from $\mathsf{H}_2^{(b)}$ with advantage ε. The only difference between $\mathsf{H}_1^{(b)}$ and $\mathsf{H}_2^{(b)}$ is that in $\mathsf{H}_2^{(b)}$, the challenger aborts the experiments if one of the conditions are satisfied when the adversary makes a zero-test query. Thus, if $\mathcal{A}$ is able to distinguish $\mathsf{H}_1^{(b)}$ from $\mathsf{H}_2^{(b)}$, it must be able to produce a zero-test query that causes the challenger to output Bad in hybrid $\mathsf{H}_2^{(b)}$ with probability ε. We use $\mathcal{A}$ to construct a factoring adversary $\mathcal{B}$ that achieves the same advantage ε. Algorithm $\mathcal{B}$ is given a factoring challenge N, where N is a product of two primes p, q, and its goal is to output p or q. In the reduction, algorithm $\mathcal{B}$ will invoke $\mathcal{A}$ and simulate for it the generic multilinear map oracle. Algorithm $\mathcal{B}$ operates as follows:

1. At the beginning of the game $\mathcal{B}$ sets $\mathsf{pp} = N$. It also initializes an empty table $T = \{\}$ to store handles to formal variables. Then, for each formal variable $\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{g}, \hat{T}$, it generates a random nonce $h_{a_1}, \ldots, h_{a_\ell}, h_{g_p}, h_g, h_T \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$, and adds the associated mappings to T. It gives the public parameters pp and the encodings $(h_{a_1}, \ldots, h_{a_\ell}, h_{g_p}, h_g, h_T)$ to $\mathcal{A}$.

2. Algorithm $\mathcal{B}$ responds to oracle queries as follows:

   - $\mathsf{GMM.Add}(k, h_1, h_2)$. Algorithm $\mathcal{B}$ first checks that $k = \mathsf{pp}$ and that h_1, h_2 map to valid formal polynomials z_1 and z_2 at levels $i_1, i_2 \in \mathbb{N}$, respectively. If $k \neq \mathsf{pp}$ or h_1 and h_2 do not both map to valid formal polynomials, or $i_1 \neq i_2$, $\mathcal{B}$ responds with $\perp$. Otherwise, $\mathcal{B}$ chooses a fresh handle $h \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$, and adds the entry $h \mapsto (z_1 + z_2, i_1)$ to T. It replies to $\mathcal{A}$ with h.
   - $\mathsf{GMM.Mult}(k, h_1, h_2)$. Algorithm $\mathcal{B}$ first checks that $k = \mathsf{pp}$ and that h_1, h_2 map to valid formal polynomials z_1 and z_2 at levels $i_1, i_2 \in \mathbb{N}$, respectively. If $k \neq \mathsf{pp}$ or h_1 and h_2 do not both map to valid formal polynomials, or $i_1 + i_2 > \ell$, $\mathcal{B}$ responds with $\perp$. Otherwise,

$\mathcal{B}$ chooses a fresh handle $h \xleftarrow{R} \{0,1\}^\lambda$, and adds the entry $h \mapsto (z_1 z_2, i_1 + i_2)$ to T. It replies to $\mathcal{A}$ with h.

- $\mathsf{GMM.ZeroTest}(k, h)$. Algorithm $\mathcal{B}$ first checks that $k = \mathsf{pp}$ and that h maps to a formal polynomial z encoded at level ℓ in T. If these checks do not pass, then $\mathcal{B}$ responds with $\perp$. Otherwise, $\mathcal{B}$ does the following:
  (a) By Lemma B.8, z can be expressed as $f_\ell(\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}, \hat{g}_p) + \alpha\hat{T}$, where f_ℓ is a polynomial of degree ℓ and $\alpha \in \mathbb{Z}$ is a scalar. Then, compute each of the following quantities:
     - Set $y_1 = \gcd(N, \alpha)$.
     - Set $y_2 = \gcd(N, z'(a_1, \ldots, a_\ell, g_p, T))$ where $z' \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{T}]$ is the polynomial z with the substitution $\hat{g} \mapsto 1$ and the values $a_1, \ldots, a_\ell, g_p, T$ are drawn uniformly from $\mathbb{Z}_N$.
     - Set $y_3 = \gcd(N, f'_\ell(a_1, \ldots, a_\ell, g_p))$, where $f'_\ell \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p]$ is the polynomial f_ℓ with the substitution $\hat{g} \mapsto 1$ and $a_1, \ldots, a_\ell, g_p \xleftarrow{R} \mathbb{Z}_N$.
     - Set $y_4 = \gcd(N, f''_\ell(a_1, \ldots, a_\ell))$, where $f''_\ell \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell]$ is the polynomial f'_ℓ with the substitution $\hat{g}_p \mapsto 0$, and $a_1, \ldots, a_\ell \xleftarrow{R} \mathbb{Z}_N$.

     If for any $i \in \{1, 2, 3, 4\}$, y_i is a non-trivial factor of N ($1 < y_i < N$), then abort the experiment, and output y_i. Otherwise, continue.
  (b) Let $z' \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{T}]$ be the polynomial z with the substitution $\hat{g} \mapsto 1$. Perform a Schwartz-Zippel test to determine if $z' \equiv 0 \pmod{N}$. If so, then respond with "zero." Otherwise continue.
  (c) Output "non-zero."

3. If the experiment has not aborted yet, output $\perp$.

We argue that $\mathcal{B}$ correctly simulates the view of $\mathcal{A}$ in $\mathsf{H}_1^{(b)}$ and $\mathsf{H}_2^{(b)}$. Since $\mathcal{B}$ samples the nonces for the challenge from the same distribution as $\mathsf{GMM.Encode}$ in the real scheme, we conclude that the challenge terms are identically distributed. Moreover, $\mathcal{B}$ answers the $\mathsf{GMM.Add}$ and $\mathsf{GMM.Mult}$ oracles in exactly the same manner as the generic multilinear map oracle. Finally, $\mathcal{B}$ responds to the zero-test queries exactly as prescribed in hybrid $\mathsf{H}_1^{(b)}$. By assumption then, with probability at least ε, algorithm $\mathcal{A}$ will produce a query polynomial $z \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{g}, \hat{T}]$ such that one of the conditions outlined in $\mathsf{H}_1^{(b)}$ will hold. But in this case, $\mathcal{B}$ is able to compute a non-trivial factor of N. The lemma follows. □

Lemma B.11. *For all adversaries $\mathcal{A}$, $\left|\Pr[\mathsf{H}_0^{(1)}(\mathcal{A}) = 1] - \Pr[\mathsf{H}_1^{(1)}(\mathcal{A}) = 1]\right|$ is negligible.*

Proof. This proof is very similar to that of Lemma B.9. Again, it suffices to argue that the distribution of zero-test responses are statistically indistinguishable between hybrids $\mathsf{H}_0^{(1)}$ and $\mathsf{H}_1^{(1)}$. Consider a valid zero-test query for a polynomial $z \in \mathbb{Z}[\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{g}, \hat{T}]$ at the top-level. If any of the conditions enumerated in the description of hybrid $\mathsf{H}_1^{(1)}$ are met, then the challenger's response to the zero-test query is identical in hybrids $\mathsf{H}_0^{(1)}$ and $\mathsf{H}_1^{(1)}$. We consider three cases. In the following analysis, define the polynomials z', f', and f'' as in the description of hybrid $\mathsf{H}_1^{(1)}$. We show that in each case, either one of the conditions in $\mathsf{H}_1^{(1)}$ holds, or the distribution of the response to the zero-test query is statistically indistinguishable between hybrids $\mathsf{H}_0^{(1)}$ and $\mathsf{H}_1^{(1)}$.

- Suppose $z' \equiv 0 \pmod{N}$. The analysis of this case is identical to that in Lemma B.9.
- Suppose $z' \not\equiv 0 \pmod{p}$. In hybrid H_2, the value of each formal variable $\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{T}$ is uniform and independent over $\mathbb{Z}_N$. By the Schwartz-Zippel lemma, $z'(\hat{a}_1, \ldots, \hat{a}_\ell, \hat{g}_p, \hat{T}) \neq 0 \pmod{p}$ with overwhelming probability. Thus, in hybrid $\mathsf{H}_0^{(1)}$, the zero-test oracle will output "non-zero" on query z. In hybrid $\mathsf{H}_1^{(1)}$, the challenger always replies "non-zero," so the two distributions are statistically indistinguishable.
- Suppose that $z' \equiv 0 \pmod{p}$ and $z' \not\equiv 0 \pmod{q}$. The analysis of this case is also identical to that in Lemma B.9.

We conclude that hybrids $\mathsf{H}_0^{(1)}$ and $\mathsf{H}_1^{(1)}$ are statistically indistinguishable. □

To complete the proof of Theorem B.7, it suffices to observe that hybrids $\mathsf{H}_2^{(0)}$ and $\mathsf{H}_2^{(1)}$ are identical distributions. In particular, the challenge values, and outputs of the GMM.Add and GMM.Mult oracles are identically distributed in the two distributions. Moreover, the decision procedure the challenger uses to implement GMM.ZeroTest is independent of the actual values of the formal variables. Thus, for all adversaries $\mathcal{A}$, $\mathsf{H}_2^{(0)}(\mathcal{A}) \equiv \mathsf{H}_2^{(1)}(\mathcal{A})$.

Combining Lemmas B.9 through B.11 together with the fact that $\mathsf{H}_2^{(0)}$ and $\mathsf{H}_2^{(1)}$ are identical experiments, we conclude that assuming hardness of factoring, hybrids $\mathsf{H}_0^{(0)}$ is computationally indistinguishable from $\mathsf{H}_0^{(1)}$, or equivalently, that the ℓ-Multilinear Diffie-Hellman Subgroup Decision assumption holds in the generic model of composite-order multilinear maps. □

C Programmable PRFs

In this section, we introduce the notion of a private programmable PRF, which can be viewed as an extension of a private puncturable PRF where the holder of the master secret key can additionally specify the value of the PRF at the punctured point. We show how private programmable PRFs can be used to construct a watermarkable family of PRFs in Section 6.2. We now describe the syntax of a programmable PRF, and give the security notions. We define a programmable PRF as a tuple of algorithms $\Pi = (\mathsf{pPRF.Setup}, \mathsf{pPRF.Program}, \mathsf{pPRF.ProgramEval}, \mathsf{pPRF.Eval})$, over a key space $\mathcal{K}$, input space $\{0,1\}^n$, and output space $\{0,1\}^m$. The syntax for a programmable PRF is essentially identical to the syntax of a constrained PRF, except that the algorithms cPRF.Constrain and cPRF.ConstrainEval are replaced by pPRF.Program and pPRF.ProgramEval.

- $\mathsf{pPRF.Setup}(1^\lambda) \to \mathsf{msk}$. On input the security parameter λ, the setup algorithm pPRF.Setup outputs the master secret key msk.
- $\mathsf{pPRF.Program}(\mathsf{msk}, x, y) \to \mathsf{sk}$. On input the master secret key msk, an input $x \in \{0,1\}^n$ and an output $y \in \{0,1\}^m$, the program algorithm pPRF.Program outputs a secret key sk.
- $\mathsf{pPRF.ProgramEval}(\mathsf{sk}, x) \to y$. On input a secret key sk, and an input $x \in \{0,1\}^n$, the programmed evaluation algorithm pPRF.ProgramEval outputs an element $y \in \{0,1\}^m$.
- $\mathsf{pPRF.Eval}(\mathsf{msk}, x) \to y$. On input the master secret key msk and an input $x \in \{0,1\}^n$, the evaluation algorithm pPRF.Eval outputs an element $y \in \{0,1\}^m$.

Correctness. A programmable PRF is correct if for $\mathsf{msk} \leftarrow \mathsf{pPRF.Setup}(1^\lambda)$, all inputs $x \in \{0,1\}^n$, setting $\mathsf{sk} = \mathsf{pPRF.Program}(\mathsf{msk}, x^*, y^*)$, it is the case that

$$\mathsf{pPRF.ProgramEval}(\mathsf{sk}, x) = \begin{cases} y^*, & \text{if } x = x^* \\ \mathsf{pPRF.Eval}(\mathsf{msk}, x), & \text{otherwise} \end{cases}.$$

Security. The security definition for the privacy of a programmable PRF is mostly identical to that of a private constrained PRF, except for a few syntactical differences which we explain below.

Definition C.1 (Experiment $\mathsf{Expt}_b^{\mathsf{pPRF}}$)**.** For the security parameter $\lambda \in \mathbb{N}$, integers $n, m > 0$, and a bit $b \in \{0,1\}$, we define the experiment $\mathsf{Expt}_b^{\mathsf{pPRF}}$ between a challenger and an adversary $\mathcal{A}$, which can make oracle queries of the following types: program, evaluation, and challenge. First, the challenger obtains $\mathsf{msk} \leftarrow \mathsf{pPRF.Setup}(1^\lambda)$. The challenger also draws a function $f \xleftarrow{R} \mathsf{Funs}(\{0,1\}^n, \{0,1\}^m)$ uniformly at random. For $b \in \{0,1\}$, the challenger responds to each oracle query type made by $\mathcal{A}$ in the following manner.

- **Program oracle.** On input an $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$, the challenger returns $\mathsf{sk} \leftarrow \mathsf{pPRF.Program}(\mathsf{msk}, x, y)$ to $\mathcal{A}$.
- **Evaluation oracle.** On input $x \in \{0,1\}^n$, the challenger returns $y \leftarrow \mathsf{pPRF.Eval}(\mathsf{msk}, x)$.
- **Challenge oracle.** On input $x \in \{0,1\}^n$, the challenger returns $y \leftarrow \mathsf{pPRF.Eval}(\mathsf{msk}, x)$ to $\mathcal{A}$ if $b = 0$, and $y \leftarrow f(x)$ if $b = 1$.

Eventually, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also output by $\mathsf{Expt}_b^{\mathsf{pPRF}}$. Let $\mathsf{Pr}[\mathsf{Expt}_b^{\mathsf{pPRF}}(\mathcal{A}) = 1]$ denote the probability that $\mathsf{Expt}_b^{\mathsf{pPRF}}$ outputs 1 with $\mathcal{A}$.

Definition C.2 (Admissible Programming)**.** An adversary is *admissible* if it makes at most one query $x \in \{0,1\}^n$ to the challenge oracle, and moreover, for all points $x_i' \in \{0,1\}^n$ it submits to the program oracle, $x_i' = x$.

Definition C.3 (Programming Security)**.** A programmable PRF Π is secure if for all efficient and admissible adversaries $\mathcal{A}$, the following quantity is negligible:

$$\mathsf{Adv}^{\mathsf{pPRF}}[\Pi, \mathcal{A}] \stackrel{\mathsf{def}}{=} \left| \mathsf{Pr}[\mathsf{Expt}_0^{\mathsf{pPRF}}(\mathcal{A}) = 1] - \mathsf{Pr}[\mathsf{Expt}_1^{\mathsf{pPRF}}(\mathcal{A}) = 1] \right|.$$

Definition C.4 (Experiment $\mathsf{Expt}_b^{\mathsf{ppriv}}$)**.** For the security parameter $\lambda \in \mathbb{N}$, integers $n, d \in \mathbb{N}$ and a bit $b \in \{0,1\}$, we define the experiment $\mathsf{Expt}_b^{\mathsf{ppriv}}$ between a challenger and an adversary $\mathcal{A}$, which can make evaluation and challenge queries. First, the challenger obtains $\mathsf{msk} \leftarrow \mathsf{pPRF.Setup}(1^\lambda)$, and samples $y^* \leftarrow \{0,1\}^m$ uniformly at random. For $b \in \{0,1\}$, the challenger responds to each oracle query type made by $\mathcal{A}$ in the following manner.

- **Evaluation oracle.** On input $x \in \{0,1\}^n$, the challenger returns $y \leftarrow \mathsf{pPRF.Eval}(\mathsf{msk}, x)$.
- **Challenge oracle.** For a pair of inputs $x_0, x_1 \in \{0,1\}^n$, the challenger returns $\mathsf{sk} \leftarrow \mathsf{pPRF.Program}(\mathsf{msk}, x_b, y^*)$.

Eventually, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also output by $\mathsf{Expt}_b^{\mathsf{pPRF}}$. Let $\mathsf{Pr}[\mathsf{Expt}_b^{\mathsf{ppriv}}(\mathcal{A}) = 1]$ denote the probability that $\mathsf{Expt}_b^{\mathsf{ppriv}}$ outputs 1.

Definition C.5 (Admissible Privacy)**.** An adversary is *admissible* if it makes at most one challenge oracle query, and it does not query the evaluation oracle on any point that also appears in a challenge oracle query.

Definition C.6 (Privacy)**.** A programmable PRF Π is private if for all efficient and admissible adversaries $\mathcal{A}$ the following quantity is negligible:

$$\mathsf{Adv}^{\mathsf{ppriv}}[\Pi, \mathcal{A}] \stackrel{\mathsf{def}}{=} \left|\Pr[\mathsf{Expt}_0^{\mathsf{ppriv}}(\mathcal{A}) = 1] - \Pr[\mathsf{Expt}_1^{\mathsf{ppriv}}(\mathcal{A}) = 1]\right|.$$

Note that in our game-based definition of privacy for programmable PRFs (Definition C.4), the adversary does not specify the value at the punctured point. Instead, the challenger samples the value uniformly at random from the range of the PRF. This restriction is essential for ensuring an achievable notion of privacy. Indeed, if the adversary was able to specify (or guess) the value at the programmed point, then it can trivially distinguish $\mathsf{Expt}_0^{\mathsf{ppriv}}$ from $\mathsf{Expt}_1^{\mathsf{ppriv}}$ by simply evaluating the programmed key at the two points x_0, x_1 it submitted to the challenge oracle. Thus, hiding the reprogrammed point is only possible if the the reprogrammed value is unknown to the adversary. While this seems like a very limiting restriction, we show in Section 6.2 that a private programmable PRF satisfying Definitions C.3 and C.6 is sufficient for constructing a watermarkable family of PRFs.

D Proofs from Section 3

D.1 Proof of Theorem 3.2

We use a hybrid argument:

- **Hybrid H_0.** This is the real experiment $\mathsf{Expt}_0^{\mathsf{cPRF}}$.
- **Hybrid H_1.** In this experiment, the challenger uses punctured keys when constructing the obfuscated programs in response to constrain queries. At the beginning of the game, the adversary first submits a point $x \in \{0,1\}^n$ on which it would like to be challenged. Next, as in the real experiment, the challenger computes $\mathsf{msk} \leftarrow \mathsf{F.Setup}(1^\lambda)$. It runs $\mathsf{sk} \leftarrow \mathsf{F.Puncture}(\mathsf{msk}, x)$. Finally, it responds to oracle queries as follows:
  - **Constrain oracle.** On input a circuit $C : \{0,1\}^n \to \{0,1\}$, the challenger first obtains $\mathsf{msk}' \leftarrow \mathsf{F.Setup}(1^\lambda)$. The challenger then outputs the obfuscated program $\mathsf{iO}\left(P_2^{(\mathsf{cPRF})}[C, \mathsf{msk}', \mathsf{sk}]\right)$, where $P_2^{(\mathsf{cPRF})}[C, \mathsf{msk}', \mathsf{sk}]$ is the program below:

    Constants: a circuit $C : \{0,1\}^n \to \{0,1\}$, a master secret key msk_0, and a punctured key sk_1 for the puncturable PRF $\Pi_{\mathsf{F}} = (\mathsf{F.Setup}, \mathsf{F.Puncture}, \mathsf{F.ConstrainEval}, \mathsf{F.Eval})$.

    On input $x \in \{0,1\}^n$:

    1. Let $b = C(x)$. If $b = 0$, output $\mathsf{F.Eval}(\mathsf{msk}_0, x)$. Otherwise, output $\mathsf{F.ConstrainEval}(\mathsf{sk}_1, x)$.

Figure 2: The program $P_2^{(\mathsf{cPRF})}[C, \mathsf{msk}_0, \mathsf{sk}_1,]$

– **Evaluation oracle.** On input $x \in \{0,1\}^n$, output $\mathsf{cPRF.Eval}(\mathsf{msk}, x)$.

– **Challenge oracle.** On input $x \in \{0,1\}^n$, output $\mathsf{cPRF.Eval}(\mathsf{msk}, x)$.

- **Hybrid H_2.** Identical to hybrid H_1, except when responding to the challenge query, the challenger responds with a uniformly random value $y \xleftarrow{R} \{0,1\}^m$.
- **Hybrid H_3.** This is the real experiment $\mathsf{Expt}_1^{\mathsf{cPRF}}$.

At a high level, the first two hybrids as well as the last two hybrids are computationally indistinguishable by security of iO. The two intermediate hybrids are computationally indistinguishable by puncturing security of Π_{F}. In the following, we will write $\mathsf{H}_i(\mathcal{A})$ to denote the output of $\mathcal{A}$ when interacting according to the specifications of H_i.

Lemma D.1. *If iO is secure, then for all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_0(\mathcal{A}) = 1] - \Pr[\mathsf{H}_1(\mathcal{A}) = 1]|$ is negligible.*

Proof. First, we note that the challenger performs the setup procedure and responds to the evaluation and challenge oracles identically in H_0 and H_1. Thus, we just consider the constrain queries. In hybrid H_0, on an input C, the challenger gives out an obfuscation of the program $P_1[C, \mathsf{msk}', \mathsf{msk}]$ whereas in hybrid H_1, the challenger gives out an obfuscation of the program $P_2^{(\mathsf{cPRF})}[C, \mathsf{msk}', \mathsf{sk}]$. By correctness of the puncturable PRF, $\mathsf{F.ConstrainEval}(\mathsf{sk}, \cdot)$ and $\mathsf{F.Eval}(\mathsf{msk}, \cdot)$ agree everywhere except perhaps on input x. By admissibility, all queries C the adversary submits to the constrain oracle must satisfy $C(x) = 0$. In H_0, on any input x where $C(x) = 0$, the program $P_1[C, \mathsf{msk}', \mathsf{msk}]$ outputs $\mathsf{F.Eval}(\mathsf{msk}', x)$, which is precisely the behavior of $P_2^{(\mathsf{cPRF})}[C, \mathsf{msk}', \mathsf{sk}]$ at x. We conclude that $P_1[C, \mathsf{msk}', \mathsf{msk}]$ and $P_2^{(\mathsf{cPRF})}[C, \mathsf{msk}', \mathsf{sk}]$ compute identical functions, so by security of iO, hybrids H_0 and H_1 are computationally indistinguishable. The lemma follows. □

Lemma D.2. *If Π_{F} is a selectively-secure puncturable PRF, then for all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_1(\mathcal{A}) = 1] - \Pr[\mathsf{H}_2(\mathcal{A}) = 1]|$ is negligible.*

Proof. Suppose $\mathcal{A}$ is able to distinguish H_1 from H_2. We use $\mathcal{A}$ to build an adversary that can win the puncturing security game for Π_{F}. Algorithm $\mathcal{B}$ begins by running algorithm $\mathcal{A}$, which outputs a point $x^* \in \{0,1\}^n$ as its challenge. Algorithm $\mathcal{B}$ then submits x^* to the puncturing security challenger as its challenge. It receives a value T. In addition, $\mathcal{B}$ issues a puncturing query to obtain a key sk punctured at x^*. Algorithm $\mathcal{B}$ then responds to oracle queries as follows:

- **Constrain oracle.** On input a circuit $C : \{0,1\}^n \to \{0,1\}$, set $\mathsf{msk}' \leftarrow \mathsf{F.Setup}(1^\lambda)$, and output the obfuscated program $\mathsf{iO}\left(P_2^{(\mathsf{cPRF})}[C, \mathsf{msk}', \mathsf{sk}]\right)$.
- **Evaluation oracle.** On input $x \in \{0,1\}^n$, output $\mathsf{F.ConstrainEval}(\mathsf{sk}, x)$.
- **Challenge oracle.** Adversary $\mathcal{A}$ can only query the oracle at x^*. Algorithm $\mathcal{B}$ responds with T.

At the end of the experiment, algorithm $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. In the simulation, the master secret key chosen by the puncturing security challenger plays the role of msk. Thus, the constrain queries are correctly simulated. By the admissibility requirement, algorithm $\mathcal{A}$ can never make an evaluation query at x^*, so by correctness of the puncturing scheme, $\mathcal{B}$ correctly simulates

the evaluation queries. For the challenge query, if $T = \mathsf{F.Eval}(\mathsf{msk}, x)$, then $\mathcal{B}$ correctly simulates hybrid H_1. If T is uniform over $\{0,1\}^m$, then $\mathcal{B}$ simulates hybrid H_2. Thus, if $\mathcal{A}$ can distinguish H_1 from H_2, then $\mathcal{B}$ wins the puncturing security game for Π_{F} with the same advantage. The lemma follows. □

Lemma D.3. *If* iO *is secure, then for all efficient adversaries* $\mathcal{A}$, $|\Pr[\mathsf{H}_2(\mathcal{A}) = 1] - \Pr[\mathsf{H}_3(\mathcal{A}) = 1]|$ *is negligible.*

Proof. Same as the proof of Lemma D.1. □

Combining Lemmas D.1 through D.3, we conclude that experiment $\mathsf{Expt}_0^{\mathsf{cPRF}}$ is computationally indistinguishable from experiment $\mathsf{Expt}_1^{\mathsf{cPRF}}$. We conclude that Π_{ioPRF} is selectively secure. □

D.2 Proof of Theorem 3.3

Fix a constant $d \in \mathbb{N}$. In the d-time privacy game (Definition 2.10), the adversary is given access to an evaluation oracle and a challenge oracle. The adversary is allowed to adaptively make up to d challenge queries. On the i^{th} challenge query, the adversary submits two circuits $C_0^{(i)}$ and $C_1^{(i)}$, and receives back a PRF key constrained to circuit $C_b^{(i)}$ where $b \in \{0,1\}$ is fixed at the start of the experiment. Moreover, the adversary is constrained to making challenge queries that satisfy the admissibility requirement from Definition 2.9: for all $i, j \in [d]$, $S(C_0^{(i)}) \cap S(C_0^{(j)}) = S(C_1^{(i)}) \cap S(C_1^{(j)})$, where for a circuit C on n-bit inputs, $S(C) = \{x \in \{0,1\}^n : C(x) = 1\}$. We now proceed with a hybrid argument. For $0 \le i \le 2^n + 1$, define the hybrid experiment H_i as follows.

Hybrid H_i. At the beginning of the experiment, the challenger runs $\mathsf{F.Setup}(1^\lambda)$ to obtain the master secret key msk. Then, the challenger responds to the oracle queries as follows:

- **Evaluation oracle.** On input $x \in \{0,1\}^n$, the challenger responds with $\mathsf{F.Eval}(\mathsf{msk}, x)$.
- **Challenge oracle.** On the j^{th} challenge query, the challenger receives as input a pair of circuits $C_0^{(j)}$ and $C_1^{(j)}$, and does the following:
  1. Set $\mathsf{msk}_j \leftarrow \mathsf{F.Setup}(1^\lambda)$.
  2. Output the indistinguishability obfuscation $\mathsf{iO}\left(P_2^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{msk}_j, \mathsf{msk}, i]\right)$, where the program $P_2^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{msk}_j, \mathsf{msk}, i]$ is defined as follows:

     Constants: two circuits C_0, C_1 where $C_0, C_1 : \{0,1\}^n \to \{0,1\}$, master secret keys $\mathsf{msk}_0, \mathsf{msk}_1$ for the puncturable PRF $\Pi_{\mathsf{F}} = (\mathsf{F.Setup}, \mathsf{F.Puncture}, \mathsf{F.ConstrainEval}, \mathsf{F.Eval})$, and a threshold $0 \le i \le 2^n + 1$.

     On input $x \in \{0,1\}^n$:

     (a) If $x < i$, then set $b = C_1(x)$. Otherwise, set $b = C_0(x)$.

     (b) Output $\mathsf{F.Eval}(\mathsf{msk}_b, x)$.

Figure 3: The program $P_2^{(\mathsf{cpriv})}[C_0, C_1, \mathsf{msk}_0, \mathsf{msk}_1, i]$

In hybrid H_0, the challenger respond to all evaluation queries exactly as in $\mathsf{Expt}_0^{\mathsf{ppriv}}$. Moreover, the programs $P_1[C_0^{(j)}, \mathsf{msk}_j, \mathsf{msk}]$ and $P_2^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{msk}_j, \mathsf{msk}, 0]$ compute identical functions for all $j \in [d]$. Thus, for each $j \in [d]$, we have that

$$\mathsf{iO}\left(P_1\left[C_0^{(j)}, \mathsf{msk}_j, \mathsf{msk}\right]\right) \overset{c}{\approx} \mathsf{iO}\left(P_2^{(\mathsf{cpriv})}\left[C_0^{(j)}, C_1^{(j)}, \mathsf{msk}_j, \mathsf{msk}, 0\right]\right).$$

A standard hybrid argument then shows that experiments $\mathsf{Expt}_0^{\mathsf{cpriv}}$ and H_0 are computationally indistinguishable. Similarly, we have that hybrids H_{2^n+1} and $\mathsf{Expt}_1^{\mathsf{ppriv}}$ are computationally indistinguishable. To conclude the proof, we show that for all $0 \leq i \leq 2^n + 1$, H_i and H_{i+1} are computationally indistinguishable. Intuitively, claim follows from the fact that the obfuscated programs given out in hybrids H_i and H_{i+1} only differ on a single point (when $x = i$). By appealing to the puncturing security of Π_{F}, we can show that the two experiments are in fact indistinguishable. We now state and prove the claim. As usual, for an adversary $\mathcal{A}$, we write $\mathsf{H}_i(\mathcal{A})$ to denote the output of $\mathcal{A}$ when interacting in experiment H_i.

Lemma D.4. *If* iO *is an indistinguishability obfuscator and* Π_{F} *is a selectively-secure puncturable PRF, then for all* $0 \leq i \leq 2^n$*, and all efficient adversaries* $\mathcal{A}$*,* $|\Pr[\mathsf{H}_i(\mathcal{A}) = 1] - \Pr[\mathsf{H}_{i+1}(\mathcal{A}) = 1]|$ *is negligible.*

Proof. We again use a hybrid argument. First, we describe our sequence of hybrid experiments. At a high level, in the first hybrid, we replace the real PRF keys in each program with punctured PRF keys. In each of the subsequent hybrids, we slowly transition the programs to compute the output at i based on the value of $C_1^{(j)}(i)$ rather than $C_0^{(j)}(i)$. Each step in these intermediate hybrids follows by puncturing security of Π_{F}. We describe our hybrids below:

- **Hybrid** $\mathsf{H}_{i,1}$**:** The challenger begins by computing $\mathsf{msk} \leftarrow \mathsf{F.Setup}(1^\lambda)$. Next, it sets $\mathsf{sk} \leftarrow \mathsf{F.Puncture}(\mathsf{msk}, i)$. The challenger responds to oracle queries as follows:
  - **Evaluation oracle.** On input $x \in \{0,1\}^n$, the challenger replies with $\mathsf{F.Eval}(\mathsf{msk}, x)$.
  - **Challenge oracle.** On the j^{th} challenge query, the challenger receives as input a pair of circuits $C_0^{(j)}$ and $C_1^{(j)}$, and does the following:
    1. Set $\mathsf{msk}_j \xleftarrow{\text{R}} \mathsf{F.Setup}(1^\lambda)$ and construct the punctured key $\mathsf{sk}_j = \mathsf{F.Puncture}(\mathsf{msk}_j, i)$.
    2. If $C_0^{(j)}(i) = 0$, then set $r_j = \mathsf{F.Eval}(\mathsf{msk}_j, i)$. Otherwise, set $r_j = \mathsf{F.Eval}(\mathsf{msk}, i)$.
    3. Output the indistinguishability obfuscation $\mathsf{iO}\left(P_3^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{sk}_j, \mathsf{sk}, r_j, i]\right)$, where the program $P_3^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{sk}_j, \mathsf{sk}, r_j, i]$ is defined as follows:

Constants: two circuits C_0, C_1 where $C_0, C_1 : \{0,1\}^n \to \{0,1\}$, punctured keys $\mathsf{sk}_0, \mathsf{sk}_1$ for the puncturable PRF $\Pi_{\mathsf{F}} = (\mathsf{F.Setup}, \mathsf{F.Puncture}, \mathsf{F.ConstrainEval}, \mathsf{F.Eval})$, a string $r \in \{0,1\}^m$, and a threshold $0 \leq i \leq 2^n + 1$.

On input $x \in \{0,1\}^n$:

(a) If $x = i$, output r. Otherwise continue.

(b) If $x < i$, then set $b = C_1(x)$. Otherwise, set $b = C_0(x)$.

(c) Output $\mathsf{F.ConstrainEval}(\mathsf{sk}_b, x)$.

Figure 4: The program $P_3^{(\mathsf{cpriv})}[C_0, C_1, \mathsf{sk}_0, \mathsf{sk}_1, r, i]$

- **Hybrid $\mathsf{H}_{i,2}$.** Same as $\mathsf{H}_{i,1}$, except when responding to a challenge query, if $C_0^{(j)}(i) = 1$ and $C_1^{(j)}(i) = 0$, the challenger sets $r_j \xleftarrow{\mathrm{R}} \{0,1\}^m$.
- **Hybrid $\mathsf{H}_{i,3}$.** Same as $\mathsf{H}_{i,2}$, except when responding to a challenge query, if $C_0^{(j)}(i) = 1$ and $C_1^{(j)}(i) = 0$, the challenger sets $r_j = \mathsf{F.Eval}(\mathsf{msk}_j, i)$.
- **Hybrid $\mathsf{H}_{i,4}$.** Same as $\mathsf{H}_{i,3}$, except when responding to a challenge query, if $C_0^{(j)}(i) = 0$ and $C_1^{(j)}(i) = 1$, the challenger sets $r_j \xleftarrow{\mathrm{R}} \{0,1\}^m$.
- **Hybrid $\mathsf{H}_{i,5}$.** Same as $\mathsf{H}_{i,4}$, except when responding to a challenge query, if $C_0^{(j)}(i) = 0$ and $C_1^{(j)}(i) = 1$, the challenger sets $r_j = \mathsf{F.Eval}(\mathsf{msk}, i)$

Now, we demonstrate that hybrid H_i is computationally indistinguishable from hybrid $\mathsf{H}_{i,1}$. Then, we demonstrate that each successive pair of hybrids $\mathsf{H}_{i,j}, \mathsf{H}_{i,j+1}$ for $j \in \{1,2,3,4\}$ are computationally indistinguishable. Finally, we show that $\mathsf{H}_{i,5}$ and H_{i+1} are computationally indistinguishable.

Lemma D.5. *If iO is an indistinguishability obfuscator, then for all $0 \leq i \leq 2^n$, and all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_i(\mathcal{A}) = 1] - \Pr[\mathsf{H}_{i,1}(\mathcal{A}) = 1]|$ is negligible.*

Proof. The challenger samples msk and responds to evaluation queries in exactly the same manner in hybrids H_i and $\mathsf{H}_{i,1}$. Moreover, by correctness of Π_{F}, programs $P_2^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{msk}_j, \mathsf{msk}, i]$ and $P_3^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{sk}_j, \mathsf{sk}, r_j, i]$ compute identical programs. The lemma follows from security of iO. □

Lemma D.6. *If Π_{F} is selectively secure, then for all $0 \leq i \leq 2^n$, and all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_{i,1}(\mathcal{A}) = 1] - \Pr[\mathsf{H}_{i,2}(\mathcal{A}) = 1]|$ is negligible.*

Proof. We consider two possibilities. Suppose that for all $j \in [d]$, either $C_0^{(j)}(i) = C_1^{(j)}(i)$ or $C_0^{(j)}(i) = 0$ and $C_1^{(j)}(i) = 1$. In this case, hybrids $\mathsf{H}_{i,1}$ and $\mathsf{H}_{i,2}$ are identical experiments, and the lemma holds.

Suppose instead that there exists some $j^* \in [d]$ such that $C_0^{(j^*)}(i) = 1$ and $C_1^{(j^*)}(i) = 0$. By the admissibility requirement (Definition 2.9), for all $j \neq j^*$, $C_0^{(j)}(i) = 0$. Otherwise, $i \in S(C_0^{(j^*)}) \cap S(C_0^{(j)})$, but $i \notin S(C_1^{(j^*)}) \cap S(C_1^{(j)})$. We show that if there exists a distinguisher $\mathcal{A}$ that can distinguish $\mathsf{H}_{i,1}$ from $\mathsf{H}_{i,2}$ with advantage ε, there exists an adversary $\mathcal{B}$ that can break the puncturing security of Π_{F} with the same advantage. Algorithm $\mathcal{B}$ works as follows. At the beginning of the puncturing security game, $\mathcal{B}$ first commits to the point i. Then, $\mathcal{B}$ makes a puncturing query at i to obtain a punctured key sk', and a challenger query at i to obtain a value T. Algorithm $\mathcal{B}$ responds to oracle queries as follows:

- **Evaluation oracle.** On input a point $x \in \{0,1\}^n$, algorithm $\mathcal{B}$ responds with the value $\mathsf{F.ConstrainEval}(\mathsf{sk}', x)$.
- **Challenge oracle.** When $\mathcal{A}$ issues its j^{th} challenge query on circuits $C_0^{(j)}, C_1^{(j)}$, algorithm $\mathcal{B}$ responds as follows:
  1. Set $\mathsf{msk}_j \leftarrow \mathsf{F.Setup}(1^\lambda)$, and set $\mathsf{sk}_j = \mathsf{F.Puncture}(\mathsf{msk}_j, i)$.
  2. If $C_0^{(j)}(i) = 1$ and $C_1^{(j)}(i) = 0$, then set $r_j = T$. Otherwise, set $r_j = \mathsf{F.Eval}(\mathsf{msk}_j, i)$.
  3. Output the obfuscated program $\mathsf{iO}\left(P_3^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{sk}_j, \mathsf{sk}', r_j, i]\right)$.

Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. We now show that if $T = \mathsf{F.Eval}(\mathsf{msk}, i)$, then $\mathcal{B}$ has perfectly simulated hybrid $\mathsf{H}_{i,1}$ for $\mathcal{B}$. Otherwise, if T is uniformly random, then $\mathcal{B}$ has perfectly simulated hybrid $\mathsf{H}_{i,2}$ for $\mathcal{B}$. In the reduction, the master secret key msk is the key chosen by the challenger in the puncturing security game. By assumption, there is some $j^* \in [d]$ where $C_0^{(j^*)}(i) = 1$ and $C_1^{(j^*)}(i) = 0$. By admissibility, $\mathcal{A}$ is not allowed to make an evaluation query at i. By correctness of Π_{F}, $\mathsf{F.ConstrainEval}(\mathsf{sk}', \cdot)$ agrees with $\mathsf{F.Eval}(\mathsf{msk}, \cdot)$ on all points $x \neq i$, so $\mathcal{B}$ correctly simulates the evaluation queries. As noted above, if $C_0^{(j^*)}(i) = 1$ and $C_1^{(j^*)}(i) = 0$ for some j^*, then for all other $j \neq j^*$, it must be the case that $C_0^{(j)}(i) = 0$. Algorithm $\mathcal{B}$ chooses the values r_j for $j \neq j^*$ exactly as in $\mathsf{H}_{i,1}$ and $\mathsf{H}_{i,2}$. When $j = j^*$ and $T = \mathsf{F.Eval}(\mathsf{msk}, i)$, then $\mathcal{B}$ correctly simulates $\mathsf{H}_{i,1}$. Conversely, if T is uniformly random, then $\mathcal{B}$ correctly simulates $\mathsf{H}_{i,2}$. We conclude that if $\mathcal{A}$ can distinguish $\mathsf{H}_{i,1}$ from H_2 with advantage ε, then $\mathsf{Adv}^{\mathsf{cPRF}}[\Pi_{\mathsf{F}}, \mathcal{B}] = \varepsilon$. The lemma follows. □

Lemma D.7. *If Π_{F} is selectively secure, then for all $0 \leq i \leq 2^n$, and all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_{i,2}(\mathcal{A}) = 1] - \Pr[\mathsf{H}_{i,3}(\mathcal{A}) = 1]|$ is negligible.*

Proof. This proof is similar to the proof of Lemma D.6. We again consider two cases. If for all $j \in [d]$, either $C_0^{(j)}(i) = C_1^{(j)}(i)$ or $C_0^{(j)}(i) = 0$ and $C_1^{(j)}(i) = 1$. Then, hybrids $\mathsf{H}_{i,1}$ and hybrids $\mathsf{H}_{i,2}$ are identical experiments, and the lemma holds.

Suppose instead that there exists some $j^* \in [d]$ such that $C_0^{(j^*)} = 1$ and $C_1^{(j^*)} = 0$. We show then that if there exists a distinguisher $\mathcal{A}$ that can distinguish $\mathsf{H}_{i,2}$ from $\mathsf{H}_{i,3}$ with advantage ε, then there exists an adversary $\mathcal{B}$ that can break the puncturing security of Π_{F}. Algorithm $\mathcal{B}$ works as follows. At the beginning of the puncturing security game, $\mathcal{B}$ commits to the point i. It then makes a puncturing query at i to obtain a punctured key sk', and a challenge query to obtain a value T. It also constructs $\mathsf{msk} \leftarrow \mathsf{F.Setup}(1^\lambda)$ and sets $\mathsf{sk} = \mathsf{F.Puncture}(\mathsf{msk}, i)$. Then, algorithm $\mathcal{B}$ responds to oracle queries as follows:

- **Evaluation oracle.** On input a point $x \in \{0,1\}^n$, algorithm $\mathcal{B}$ responds with the value $\mathsf{F.Eval}(\mathsf{msk}, x)$.

- **Challenge oracle.** When $\mathcal{A}$ issues its j^{th} challenge query on circuits $C_0^{(j)}$ and $C_1^{(j)}$, algorithm $\mathcal{B}$ responds as follows:

  1. If $C_0^{(j)}(i) = 1$ and $C_1^{(j)}(i) = 0$, then it sets $\mathsf{sk}_j = \mathsf{sk}'$ and $r_j = T$. Otherwise, compute $\mathsf{msk}_j \leftarrow \mathsf{F.Setup}(1^\lambda)$, and set $\mathsf{sk}_j \leftarrow \mathsf{F.Puncture}(\mathsf{msk}_j, i)$, $r_j = \mathsf{F.Eval}(\mathsf{msk}_j, i)$.
  2. Output the obfuscated program $\mathsf{iO}\left(P_3^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{sk}_j, \mathsf{sk}, r_j, i]\right)$.

Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. In the simulation, the master secret key msk' chosen by the puncturing challenger plays the role of msk_{j^*}. We see that if T is uniform over $\{0,1\}^m$, then r_{j^*} is uniform over $\{0,1\}^m$, which is precisely the distribution in $\mathsf{H}_{i,2}$. Conversely, if $r_{j^*} = T = \mathsf{F.Eval}(\mathsf{msk}', i)$, then we are in $\mathsf{H}_{i,3}$. By the admissibility requirement, for all $j \neq j^*$, we have that $C_0^{(j)}(i) = 0$. Thus, everything else is constructed as in hybrids $\mathsf{H}_{i,2}$ and $\mathsf{H}_{i,3}$. We conclude that if $\mathcal{A}$ can distinguish $\mathsf{H}_{i,2}$ from $\mathsf{H}_{i,3}$ with advantage ε, then $\mathsf{Adv}^{\mathsf{cPRF}}[\Pi_\mathsf{F}, \mathcal{B}] = \varepsilon$. The lemma follows. □

Lemma D.8. *If Π_F is selectively secure, then for all $0 \leq i \leq 2^n$, and all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_{i,3}(\mathcal{A}) = 1] - \Pr[\mathsf{H}_{i,4}(\mathcal{A}) = 1]|$ is negligible.*

Proof. As usual, we consider two possibilities. If for all $j \in [d]$, either $C_0^{(j)}(i) = C_1^{(j)}(i)$ or $C_0^{(j)}(i) = 1$ and $C_1^{(j)}(i) = 0$. In this case, hybrids $\mathsf{H}_{i,3}$ and $\mathsf{H}_{i,4}$ are identical experiments, and the lemma holds.

Alternatively, suppose that there exists $j^* \in [d]$ such that $C_0^{(j^*)}(i) = 0$ and $C_1^{(j^*)}(i) = 1$. By the admissibility criterion, this means that for all $j \neq j^*$, $C_1^{(j)}(i) = 0$. This means that for all $j \neq j^*$, in hybrids $\mathsf{H}_{i,3}$ and $\mathsf{H}_{i,4}$, the challenger sets $r_j = \mathsf{F.Eval}(\mathsf{msk}_j, i)$. The claim now follows by a puncturing security argument similar to that used in the proof of Lemma D.7. In particular, given a distinguisher $\mathcal{A}$ for $\mathsf{H}_{i,3}$ and $\mathsf{H}_{i,4}$, we can build an adversary $\mathcal{B}$ for the puncturing security game for Π_F. In the reduction, the master secret key msk' chosen by the puncturing security challenger plays the role of msk_{j^*}. Then, $\mathcal{B}$ sets r_{j^*} to the challenge value T. If $T = \mathsf{F.Eval}(\mathsf{msk}', i)$, then $\mathcal{B}$ has perfectly simulated $\mathsf{H}_{i,3}$, and if T is uniform over $\{0,1\}^m$, then $\mathcal{B}$ has perfectly simulated $\mathsf{H}_{i,4}$. The lemma follows by puncturing security of Π_F. □

Lemma D.9. *If Π_F is selectively secure, then for all $0 \leq i \leq 2^n$, and all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_{i,4}(\mathcal{A}) = 1] - \Pr[\mathsf{H}_{i,5}(\mathcal{A}) = 1]|$ is negligible.*

Proof. We consider two cases. If for all $j \in [d]$, either $C_0^{(j)}(i) = C_1^{(j)}(i)$ or $C_0^{(j)}(i) = 1$ and $C_1^{(j)}(i) = 0$. In this case, hybrids $\mathsf{H}_{i,3}$ and $\mathsf{H}_{i,4}$ are identical experiments, and the lemma holds.

Alternatively, suppose that there exists $j^* \in [d]$ such that $C_0^{(j^*)}(i) = 0$ and $C_1^{(j^*)}(i) = 1$. By the admissibility criterion, this means that for all $j \neq j^*$, $C_1^{(j)}(i) = 0$. The claim now follows by a puncturing security argument similar to that used in the proof of Lemma D.6. In particular, given a distinguisher $\mathcal{A}$ for $\mathsf{H}_{i,4}$ and $\mathsf{H}_{i,5}$, we can build an adversary $\mathcal{B}$ for the puncturing security game for Π_F. In the reduction, the key chosen by the puncturing security challenger plays the role of msk. As in the proof of Lemma D.6, evaluation queries can be answered using only the punctured key since the adversary is not allowed to query for the value at the punctured point i. In the reduction, $\mathcal{B}$ sets r_{j^*} to the challenge value T. Since for all other $j \neq j^*$, we have $C_1^{(j)}(i) = 0$, so $r_j = \mathsf{F.Eval}(\mathsf{msk}_j, i)$

in hybrids $\mathsf{H}_{i,4}$ and $\mathsf{H}_{i,5}$. Algorithm $\mathcal{B}$ can simply choose the msk_j for itself, exactly as in the real scheme. Finally, we note that if the challenge T is a uniformly random value, then $\mathcal{B}$ has perfectly simulated $\mathsf{H}_{i,4}$. Otherwise, if $T = \mathsf{F.Eval}(\mathsf{msk}, i)$, then $\mathcal{B}$ has perfectly simulated $\mathsf{H}_{i,5}$. The lemma follows. □

Lemma D.10. *If* iO *is an indistinguishability obfuscator, then for all* $0 \leq i \leq 2^n$*, and all efficient adversaries* $\mathcal{A}$*,* $|\Pr[\mathsf{H}_{i,5}(\mathcal{A}) = 1] - \Pr[\mathsf{H}_{i+1}(\mathcal{A}) = 1]|$ *is negligible.*

Proof. By construction, in hybrid $\mathsf{H}_{i,5}$, the challenger sets $r_j = \mathsf{F.Eval}(\mathsf{msk}, i)$ if $C_1^{(j)}(i) = 1$ and $r_j = \mathsf{F.Eval}(\mathsf{msk}_j, i)$ if $C_1^{(j)}(i) = 0$. Together with correctness of Π_{F}, we have for all $j \in [d]$ that the program $P_3^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{sk}_j, \mathsf{sk}, r_j, i]$ in $\mathsf{H}_{i,5}$ computes the exact same functionality as $P_2^{(\mathsf{cpriv})}[C_0^{(j)}, C_1^{(j)}, \mathsf{msk}_j, \mathsf{msk}, i+1]$. Indistinguishability then follows by security of iO. □

Combining Lemmas D.5 through D.10, we conclude that hybrid experiments H_i and H_{i+1} are computationally indistinguishable for all $1 \leq i \leq 2^n$. This proves Lemma D.4. □

By Lemma D.4, we have that hybrids H_i and H_{i+1} are computationally indistinguishable for all $1 \leq i \leq 2^n$. Thus, as long as the underlying indistinguishability obfuscator and puncturable PRF is secure against subexponential time adversaries, Theorem 3.3 holds. □

Remark D.11 (Security from Polynomial Hardness)**.** We note that Theorem 3.3 only requires subexponentially-secure iO (and one-way functions) if the set of challenge circuits $\{C_0^{(j)}\}_{j\in[d]}$ and $\{C_1^{(j)}\}_{j\in[d]}$ the adversary submits differ on a super-polynomial number of points. In particular, let $S = \left\{x \in \{0,1\}^n \mid \exists j \in [d] : C_0^{(j)}(x) \neq C_1^{(j)}(x)\right\}$ be the set of points on which each pair of circuits differ. In the proof of Theorem 3.3, hybrid experiments H_i and H_{i+1} are identical experiments if $i \notin S$. Thus, we only need to introduce a hybrid for each $x \in S$. If we restrict the class of circuits on which the adversary can query such that $|S| = \mathrm{poly}(n)$, Theorem 3.3 would hold assuming polynomial hardness of iO and one-way functions. This implies, for example, that Π_{ioPRF} is a private puncturable PRF assuming only polynomial hardness of iO and selective-security of Π_{F}.

E A Private Programmable PRF from Obfuscation

We note that the construction of a private circuit-constrained PRF from Section 3 can be easily adapted to obtain a private programmable PRF (defined in Appendix C). In this section, we describe our iO-based construction of a private programmable PRF. As before, we require a puncturable PRF $\Pi_{\mathsf{F}} = (\mathsf{F.Setup}, \mathsf{F.Puncture}, \mathsf{F.ConstrainEval}, \mathsf{F.Eval})$. We define our private programmable PRF $\Pi_{\mathsf{iopPRF}} = (\mathsf{cPRF.Setup}, \mathsf{cPRF.Program}, \mathsf{cPRF.ConstrainEval}, \mathsf{cPRF.Eval})$ as follows:

- $\mathsf{cPRF.Setup}(1^\lambda)$. The setup algorithm computes $\mathsf{msk} \leftarrow \mathsf{F.Setup}(1^\lambda)$.
- $\mathsf{cPRF.Program}(\mathsf{msk}, x, y)$. The constrain algorithm outputs an obfuscation $\mathsf{iO}(P[\mathsf{msk}, x, y])$, where $P[\mathsf{msk}, x, y]$ is the following program[12]:

[12] Note that we pad the program $\bar{P}_1[\mathsf{msk}, x^*, y]$ to the maximum size of any program that appears in the hybrid experiments in the proofs of Theorem E.1 and E.2.

Constants: a master secret key msk for the puncturable PRF scheme $\Pi_{\mathsf{F}} = (\mathsf{F.Setup}, \mathsf{F.Puncture}, \mathsf{F.ConstrainEval}, \mathsf{F.Eval})$, a point $x^* \in \{0,1\}^n$, and a value $y \in \{0,1\}^m$.

On input $x \in \{0,1\}^n$:

1. If $x = x^*$, output y. Otherwise, output $\mathsf{F.Eval}(\mathsf{msk}, x)$.

Figure 5: The program $\bar{P}_1\,[\mathsf{msk}, x^*, y]$

- $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, x)$. The constrained evaluation algorithm outputs the evaluation of the obfuscated program sk on x.
- $\mathsf{cPRF.Eval}(\mathsf{msk}, x)$. The evaluation algorithm outputs $\mathsf{F.Eval}(\mathsf{msk}, x)$.

Correctness. By construction, the program $\bar{P}_1[\mathsf{msk}, x, y]$ outputs y on input x and $\mathsf{F.Eval}(\mathsf{msk}, \cdot)$ on all other inputs. Correctness of the scheme then follows from correctness of iO.

Security. We now state the security and privacy theorems for Π_{iopPRF}. The security and privacy proofs are similar to those of Theorems 3.2 and 3.3.

Theorem E.1. *If* iO *is an indistinguishability obfuscator and* Π_{F} *is a selectively-secure puncturable PRF, then* Π_{iopPRF} *is selectively-secure.*

Proof. We describe the hybrids below.

- **Hybrid** H_0**.** This is the real experiment $\mathsf{Expt}_0^{\mathsf{cPRF}}$ (Definition C.1).
- **Hybrid** H_1**.** This is identical to H_0, except the challenger substitutes $\mathsf{sk} \leftarrow \mathsf{F.Puncture}(\mathsf{msk}, x)$ for msk in the program $\bar{P}_1[\mathsf{msk}, x, y]$. Everything else remains unchanged.
- **Hybrid** H_2**.** This is identical to H_1, except when responding to the challenge query, the challenger returns a uniform random value $r \xleftarrow{R} \{0,1\}^m$.
- **Hybrid** H_3**.** This is the real experiment $\mathsf{Expt}_1^{\mathsf{cPRF}}$ (Definition C.1).

As in the proof of Theorem 3.2, hybrids H_0 and H_1 are computationally indistinguishable by correctness of the puncturable PRF Π_{F} and iO security. The same argument shows that hybrids H_2 and H_3 are computationally indistinguishable. Hybrids H_1 and H_2 are computationally indistinguishable by selective security of Π_{F} (analogous to the argument in the proof of Lemma D.2). The theorem follows. □

Theorem E.2. *If* iO *is an indistinguishability obfuscator and* Π_{F} *is a selectively-secure puncturable PRF, then* Π_{iopPRF} *is private.*

Proof. In the privacy with programmability experiment $\mathsf{Expt}_b^{\mathsf{ppriv}}$, recall that the challenger first computes $\mathsf{msk} \leftarrow \mathsf{cPRF.Setup}(1^\lambda)$. The adversary can then make evaluation queries; at some point, it makes a challenge query by sending two points $x_0, x_1 \in \{0,1\}^n$. The challenger responds with $\mathsf{cPRF.Program}(\mathsf{msk}, x_b, y)$, where y is chosen uniformly at random. In the first hybrid, we will have the challenger choose y to be the output of a puncturable PRF. In doing so, we arrive at the construction of Π_{ioPRF}, which we know to be private (Theorem 3.3). This suffices to prove the claim. We now describe the hybrids more concretely.

- **Hybrid H_0.** This is the real experiment $\mathsf{Expt}_0^{\mathsf{ppriv}}$.
- **Hybrid H_1.** Same as H_0, except the challenger implements the challenge oracle as follows:
  - **Challenge oracle.** On input $x_0, x_1 \in \{0,1\}^n$, the challenger computes $\mathsf{msk}' \leftarrow \mathsf{F.Setup}(1^\lambda)$ and sets $y = \mathsf{F.Eval}(\mathsf{msk}', x_0)$. It then outputs $\mathsf{cPRF.Program}(\mathsf{msk}, x_0, y)$.
- **Hybrid H_2.** Same as H_1, except on a challenge query $x_0, x_1 \in \{0,1\}^n$, the challenger outputs $\mathsf{cPRF.Program}(\mathsf{msk}, x_1, y)$.
- **Hybrid H_3.** This is the real experiment $\mathsf{Expt}_1^{\mathsf{ppriv}}$.

Lemma E.3. *If Π_{F} is a selectively-secure puncturable PRF, then for all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_0(\mathcal{A}) = 1] - \Pr[\mathsf{H}_1(\mathcal{A}) = 1]|$ is negligible.*

Proof. Hybrids H_0 and H_1 only differ in how y is chosen. In H_0, y is truly random, and in H_1, y is pseudorandom. Thus, indistinguishability follows directly from security of Π_{F}. Concretely, if $\mathcal{A}$ is a distinguisher for hybrids H_0 and H_1, then we can construct an adversary $\mathcal{B}$ that breaks the PRF security of Π_{F}. In particular $\mathcal{B}$ performs the setup procedure and answers the evaluation queries exactly as in hybrids H_0 and H_1. When $\mathcal{A}$ issues a challenge query, $\mathcal{B}$ queries the PRF challenger at x_0 to determine the value y. If y is truly random, $\mathcal{B}$ perfectly simulates H_0 and if y is pseudorandom, $\mathcal{B}$ perfectly simulates H_1. □

Lemma E.4. *If iO is a secure indistinguishability obfuscator and Π_{F} is selectively-secure, then for all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_1(\mathcal{A}) = 1] - \Pr[\mathsf{H}_2(\mathcal{A}) = 1]|$ is negligible.*

Proof. In hybrid H_1, on input x_0, x_1, the challenger outputs an obfuscation of the program $\bar{P}_1[\mathsf{msk}, x_0, \mathsf{F.Eval}(\mathsf{msk}', x_0)]$. Let $C_0 : \{0,1\}^n \to \{0,1\}$ be the circuit that is 1 everywhere except at x_0. By construction, the program $\bar{P}_1[\mathsf{msk}, x_0, \mathsf{F.Eval}(\mathsf{msk}', x_0)]$ is functionally equivalent to the program $P_1[C_0, \mathsf{msk}', \mathsf{msk}]$ from Figure 1. In hybrid H_2, on input x_0, x_1, the challenger outputs an obfuscation of the program $\bar{P}_1[\mathsf{msk}, x_1, \mathsf{F.Eval}(\mathsf{msk}', x_1)]$, which is functionally equivalent to the program $P_1[C_1, \mathsf{msk}', \mathsf{msk}]$, where $C_1 : \{0,1\}^n \to \{0,1\}$ is the circuit that is 1 everywhere except at x_1. By appealing to the security of iO and the argument in the proof of Theorem 3.3, we have that

$$\begin{aligned}
\mathsf{iO}\left(\bar{P}_1[\mathsf{msk}, x_0, \mathsf{F.Eval}(\mathsf{msk}', x_0)]\right) &\overset{c}{\approx} \mathsf{iO}\left(P_1[C_0, \mathsf{msk}', \mathsf{msk}]\right) \\
&\overset{c}{\approx} \mathsf{iO}\left(P_1[C_1, \mathsf{msk}', \mathsf{msk}]\right) \\
&\overset{c}{\approx} \mathsf{iO}\left(\bar{P}_1[\mathsf{msk}, x_1, \mathsf{F.Eval}(\mathsf{msk}', x_1)]\right).
\end{aligned}$$

Note that because circuits C_0 and C_1 differ only on a constant number of points (more precisely, two points), we do not require an exponential number of hybrids when applying the argument from

the proof of Theorem 3.3 (see Remark D.11). Thus, the claim follows from security of iO and Π_{F} against polynomial-time adversaries. □

Lemma E.5. *If Π_{F} is a secure puncturable PRF, then for all efficient adversaries $\mathcal{A}$, the quantity $|\Pr[\mathsf{H}_2(\mathcal{A}) = 1] - \Pr[\mathsf{H}_3(\mathcal{A}) = 1]|$ is negligible.*

Proof. Same argument as in the proof of Lemma E.3. □

Combining Lemmas E.3 through E.5, we conclude that Π_{iopPRF} is a private programmable PRF. □

F Proofs from Section 4

F.1 Proof of Theorem 4.3

Let $\mathsf{msk} \leftarrow \mathsf{cPRF.Setup}(1^\lambda)$ and let $C \in \mathcal{C}_{\mathsf{bf}}$ be a circuit. Let $s = s_1 s_2 \cdots s_n$ be the bit-fixing string for C. Let $x \in \{0,1\}^n$ be an arbitrary input for which $C(x) = 1$. Write msk as defined in Equation (4.1), and write the constrained key $\mathsf{sk} \leftarrow \mathsf{cPRF.Constrain}(\mathsf{msk}, s)$ as in Equation (4.2). By construction,

$$\begin{aligned}
\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, x) &= e\left((g^\alpha)^{\beta_0}, (D_{1,x_1})^{\beta_1}, \ldots, (D_{n,x_n})^{\beta_n}\right) \\
&= e\left((g^\alpha)^{\beta_0}, (g^{d_{1,x_1}})^{\beta_1}, \ldots, (g^{d_{n,x_n}})^{\beta_n}\right) \\
&= g_{n+1}^{\alpha\left(\prod_{i\in[n]} d_{i,x_i}\right)\left(\prod_{i=0}^{n} \beta_i\right)} \\
&= g_{n+1}^{\alpha \prod_{i\in[n]} d_{i,x_i}} = \mathsf{cPRF.Eval}(\mathsf{msk}, x).
\end{aligned}$$

The second equality follows from the fact that $C(x) = 1$, and so $D_{i,x_i} = g^{d_{i,x_i}}$ for all $i \in [n]$. The third equality follows by multilinearity of e. For the fourth inequality, we use the fact that $\beta_0 \beta_1 \cdots \beta_n = 1$, which yields the claim.

F.2 Proof of Theorem 4.4

Let $\mathcal{A}$ be an efficient and admissible adversary that participates in experiment $\mathsf{Expt}_b^{\mathsf{cPRF}}$ (Definition 2.3). We construct an efficient algorithm $\mathcal{B}$ which breaks the $(n+1)$-MDH assumption with advantage $\mathsf{Adv}^{\mathsf{cPRF}}(\Pi_{\mathsf{bfPRF}}, \mathcal{A})$. The algorithm $\mathcal{B}$ is given a $(n+1)$-MDH challenge of the form

$$(\vec{\mathbb{G}}, p, g, g_{n+1}, g^{a_1}, \ldots, g^{a_{n+2}}, g_{n+1}^u),$$

and must decide whether $u = a_1 a_2 \cdots a_{n+2}$ or if u was sampled uniformly from $\mathbb{Z}_p$.

At the beginning of the experiment, the adversary $\mathcal{A}$ commits to its challenge query $x^* = x_1^* x_2^* \cdots x_n^* \in \{0,1\}^n$ and sends x^* to $\mathcal{B}$. Then, algorithm $\mathcal{B}$ independently samples n random elements $d_1, \ldots, d_n \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. For each $i \in [n]$, define

$$(d_{i,0}^*, d_{i,1}^*) = \begin{cases} (a_i, d_i), & \text{if } x_i^* = 0 \\ (d_i, a_i), & \text{if } x_i^* = 1 \end{cases}.$$

Algorithm $\mathcal{B}$ then simulates the response to each oracle query type made by $\mathcal{A}$ in the following manner.

Constrain oracle. On input a bit-fixing string $s = s_1 s_2 \cdots s_n \in \{0,1,?\}^n$, algorithm $\mathcal{B}$ does the following. First, note that s cannot match x^* because $\mathcal{A}$ must be admissible (Definition 2.4). Let j be the first index for which $x^*_j \neq s_j$ and $s_j \neq ?$. Algorithm $\mathcal{B}$ independently samples uniformly random elements $r_1, \ldots, r_n \in \mathbb{Z}_p$ and $\beta_1, \ldots, \beta_n \in \mathbb{Z}_p$. Define $\beta_0 = (\beta_1 \beta_2 \cdots \beta_n)^{-1}$. For each $i \in [n] \setminus \{j\}$, define

$$(D_{i,0}, D_{i,1}) = \begin{cases} (g^{d^*_{i,0}}, g^{r_i}), & \text{if } s_i = 0 \\ (g^{r_i}, g^{d^*_{i,1}}), & \text{if } s_i = 1 \\ (g^{d^*_{i,0}}, g^{d^*_{i,1}}), & \text{if } s_i = ? \end{cases},$$

and define

$$(D_{j,0}, D_{j,1}) = \begin{cases} \left((g^{a_{n+2}})^{d_i}, g^{r_i}\right), & \text{if } s_i = 0 \\ \left(g^{r_i}, (g^{a_{n+2}})^{d_i}\right), & \text{if } s_i = 1 \end{cases}.$$

Finally, $\mathcal{B}$ responds with $\mathsf{sk} = \left((g^{a_{n+1}})^{\beta_0}, \left\{(D_{i,0})^{\beta_i}, (D_{i,1})^{\beta_i}\right\}_{i \in [n]}\right)$.

Evaluation oracle. On input $x \in \{0,1\}^n$, algorithm $\mathcal{B}$ treats x as a bit-fixing pattern, and applies the logic used to respond to constrain queries on x. Let sk be the resulting key. Algorithm $\mathcal{B}$ replies with $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, x)$.

Challenge oracle. The adversary $\mathcal{A}$ can only query the challenge oracle on input x^*, to which $\mathcal{B}$ returns $y = g^u_{n+1}$ (obtained directly from the $(n+1)$-MDH challenge) to the adversary.

Eventually, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also output by $\mathcal{B}$. We show that if the $(n+1)$-MDH challenger sampled u uniformly from $\mathbb{Z}_p$, then $\mathcal{B}$ has simulated $\mathsf{Expt}^{\mathsf{cPRF}}_1$ for Π_{bfPRF}, and if the $(n+1)$-MDH challenger set $u = \prod_{i \in [n+2]} a_i$, then $\mathcal{B}$ has simulated $\mathsf{Expt}^{\mathsf{cPRF}}_0$ for Π_{bfPRF}.

Set $\alpha = a_{n+1} a_{n+2}$. Algorithm $\mathcal{B}$ simulates the distribution of msk from the setup phase as the tuple

$$\mathsf{msk} \equiv \left(g_{n+1}, \alpha, \{d^*_{i,0}, d^*_{i,1}\}_{i \in [n]}\right),$$

which is identically distributed to msk obtained from $\mathsf{cPRF.Setup}(1^\lambda)$ since the scalars $a_1, \ldots, a_{n+2}$ and $d_1, \ldots, d_n$ are all distributed uniformly and independently in $\mathbb{Z}_p$.

For each constrain oracle query, on input a bit-fixing string s, let $j \in [n]$ be the first index for which $x^*_j \neq s_j$ and $s_j \neq ?$. Define

$$(\hat{D}_{j,0}, \hat{D}_{j,1}) = \begin{cases} \left(g^{d^*_{j,0}}, g^{r_j}\right), & \text{if } s_j = 0 \\ \left(g^{r_j}, g^{d^*_{j,1}}\right), & \text{if } s_j = 1 \end{cases}.$$

For scalars $\gamma_1, \ldots, \gamma_n \xleftarrow{R} \mathbb{Z}_p$ drawn independently and uniformly and $\gamma_0 = (\gamma_1 \gamma_2 \cdots \gamma_n)^{-1}$, the secret key sk that $\mathcal{B}$ returns is distributed identically as

$$\begin{aligned} \mathsf{sk} &\equiv \left((g^{a_{n+1}\, a_{n+2}})^{(a_{n+2})^{-1}\gamma_0},\ \{(D_{i,0})^{\gamma_i}, (D_{i,1})^{\gamma_i}\}_{i \neq j} \cup \left\{\hat{D}^{a_{n+2}\gamma_j}_{j,0}, \hat{D}^{a_{n+2}\gamma_j}_{j,1}\right\}\right) \\ &\equiv \left((g^{\alpha})^{\gamma_0},\ \{(D_{i,0})^{\gamma_i}, (D_{i,1})^{\gamma_i}\}_{i \neq j} \cup \left\{\hat{D}^{\gamma_j}_{j,0}, \hat{D}^{\gamma_j}_{j,1}\right\}\right). \end{aligned}$$

The first equivalence comes from the construction of sk and the distribution of the random variables β_i. The second equivalence is derived from modeling $\alpha = a_{n+1}a_{n+2}$ and from the fact that the following two distributions are identical:

$$\{\gamma_0, \ldots, \gamma_n\} \qquad \text{and} \qquad \{(a_{n+2})^{-1}\gamma_0, \gamma_1, \cdots, \gamma_{j-1}, (a_{n+2})\,\gamma_j, \gamma_{j+1}, \cdots, \gamma_n\}.$$

We conclude that $\mathcal{B}$'s response to a constrain query matches the distribution of $\mathsf{cPRF.Constrain}(\mathsf{msk}, \cdot)$. Next, since $\mathcal{B}$ correctly simulates the constrain queries, it also correctly simulates the evaluation queries. This follows from the fact that a pattern of the form $x \in \{0,1\}^n$ always matches the string x and the correctness of the scheme.

For the challenge query, if u is distributed uniformly in $\mathbb{Z}_p$, then the challenge oracle responds with a uniformly and independently distributed element of $\mathbb{G}_{n+1}$, which means that $\mathcal{B}$ has perfectly simulated $\mathsf{Expt}_1^{\mathsf{cPRF}}$. On the other hand, if $u = \prod_{i\in[n+2]} a_i$, then $y = g_{n+1}^u = g_{n+1}^{\alpha \prod_{i\in[n]} a_i}$. Since $\alpha = a_{n+1}\,a_{n+2}$, algorithm $\mathcal{B}$ has perfectly simulated $\mathsf{Expt}_0^{\mathsf{cPRF}}$ for Π_{bfPRF}. We conclude that under the $(n+1)$-MDH assumption that $\mathsf{Adv}^{\mathsf{cPRF}}[\Pi_{\mathsf{bfPRF}}, \mathcal{A}]$ is negligible. □

F.3 Proof of Theorem 4.5

Let $\mathcal{A}$ be an efficient adversary which participates in the experiment $\mathsf{Expt}_b^{\mathsf{cpriv}}$ (without access to an evaluation oracle). For any two bit-fixing strings $s_1, s_2 \in \{0,1,?\}^n$, consider the secret key $\mathsf{sk}_b = \mathsf{cPRF.Constrain}(\mathsf{msk}, s_b)$. The adversary's view in $\mathsf{Expt}_b^{\mathsf{cpriv}}$ consists solely of the key sk_b. Since sk_b contains $2n+1$ group elements which are distributed identically as $2n+1$ group elements drawn uniformly and independently from $\mathbb{G}_1$. In particular, sk_b is independent of s_1 and s_2, and so we conclude that $\mathsf{Adv}^{\mathsf{cpriv}}[\Pi_{\mathsf{bfPRF}}, \mathcal{A}] = 1/2$, which yields the claim. □

G Proofs from Section 5

G.1 Proof of Theorem 5.2

Let $\mathsf{msk} \leftarrow \mathsf{cPRF.Setup}(1^\lambda)$ and take any point $s \in \{0,1\}^n$. Let $x \in \{0,1\}^n$ be such that $x \neq s$. Using msk as defined in Equation 5.1, write the punctured key $\mathsf{sk} \leftarrow \mathsf{cPRF.Puncture}(\mathsf{msk}, s)$ as a collection of group elements $\{D_{i,0}, D_{i,1}\}_{i\in[n]}$. By construction, for all $i \in [n]$ and $b \in \{0,1\}$, each $D_{i,b}$ can be described as the product $g_{p,1}^{d_{i,b}} g_{q,1}^{z_{i,b}}$, where either $z_{i,b} = d_{i,b}$ or $z_{i,b} = 0$. To conclude the proof, we note that

$$\begin{aligned}
\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, x) &= e(D_{1,x_1}, \ldots, D_{n,x_n}) \\
&= e(g_{p,1}^{d_{1,x_1}} g_{q,1}^{z_{1,x_1}}, \ldots, g_{p,1}^{d_{n,x_n}} g_{q,1}^{z_{n,x_n}}) \\
&= e(g_{p,1}^{d_{1,x_1}}, \ldots, g_{p,1}^{d_{n,x_n}}) \cdot e(g_{q,1}^{z_{1,x_1}}, \ldots, g_{q,1}^{z_{n,x_n}}) \\
&= e(g_{p,1}^{d_{1,x_1}}, \ldots, g_{p,1}^{d_{n,x_n}}) \cdot 1 \\
&= g_{p,n}^{\prod_{i\in[n]} d_{i,x_i}} = \mathsf{cPRF.Eval}(\mathsf{msk}, x).
\end{aligned}$$

The first two equalities follow by definition of $\mathsf{cPRF.ConstrainEval}$ and $\mathsf{cPRF.Puncture}$. The third and fifth equalities follow by the multilinearity of e. For the fourth equality, we use the fact that $x \neq s$, which means that there exists an index $i \in [n]$ where $x_i \neq s_i$. By construction of $\mathsf{cPRF.Puncture}$, $z_{i,x_i} = 0$ and so the second evaluation of the multilinear map is raised to the 0^{th} power. □

G.2 Proof of Theorem 5.3

Let $\mathcal{A}$ be an efficient adversary which participates in the experiment $\mathsf{Expt}_b^{\mathsf{cPRF}}$. We construct an efficient adversary $\mathcal{B}$ that breaks the n-MDHSD assumption with advantage $\mathsf{Adv}^{\mathsf{pPRF}}[\Pi_{\mathsf{puncPRF}}, \mathcal{A}]$. Algorithm $\mathcal{B}$ is given a n-MDHSD challenge of the form

$$\left(\vec{\mathbb{G}}, N, g_1, h = g_{p,1}^{\gamma}, A_1 = g_1^{a_1}, \ldots, A_n = g_1^{a_n}, T\right),$$

and must decide if $T = g_{p,n}^{z}$ or $T = g_n^{r}$, where $z = \prod_{i\in[n]} a_i$, r is uniform in $\mathbb{Z}_N$, γ is uniform in $\mathbb{Z}_p$, $g_1 = g_{p,1}g_{q,1}$ is the canonical generator of $\mathbb{G}_1$, and $g_n = g_{p,n}g_{q,n}$ is the canonical generator of $\mathbb{G}_n$. Algorithm $\mathcal{B}$ starts by running $\mathcal{A}$ to receive a challenge point $x^* \in \{0,1\}^n$. Then, algorithm $\mathcal{B}$ samples n random elements $d_1, \ldots, d_n \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. We describe how $\mathcal{B}$ simulates the constrain, evaluation, and challenge oracle queries.

Constrain oracle. Let $s \in \{0,1\}^n$ be the point from the adversary. For each $i \in [n]$ and $b \in \{0,1\}$, algorithm $\mathcal{B}$ constructs the elements $D_{i,b}$ as follows:

$$(D_{i,0}, D_{i,1}) = \begin{cases} (A_i, h^{d_i}), & \text{if } s_i = 0 \\ (h^{d_i}, A_i), & \text{if } s_i = 1 \end{cases}. \tag{G.1}$$

Algorithm $\mathcal{B}$ then outputs the punctured key $\mathsf{sk} = \{D_{i,0}, D_{i,1}\}_{i\in[n]}$.

Evaluation oracle. On input $x \in \{0,1\}^n$, algorithm $\mathcal{B}$ invokes the constrain oracle on x^* to obtain the punctured key sk. It then returns $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, x)$. Note that since $\mathcal{A}$ is admissible, $x \neq x^*$.

Challenge oracle. Algorithm $\mathcal{A}$ can only query the challenge oracle on input x^*, to which algorithm $\mathcal{B}$ responds with the value T.

Eventually, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also output by $\mathcal{B}$. We show that if $T = g_{p,\ell}^{z}$, then $\mathcal{B}$ perfectly simulates experiment $\mathsf{Expt}_0^{\mathsf{cPRF}}$ for $\mathcal{A}$ and if $T = g_\ell^{s}$, then $\mathcal{B}$ perfectly simulates experiment $\mathsf{Expt}_1^{\mathsf{cPRF}}$ for $\mathcal{A}$. Then, the advantage of $\mathcal{B}$ in breaking the n-MDHSD assumption is precisely $\mathsf{Adv}^{\mathsf{pPRF}}[\Pi_{\mathsf{puncPRF}}, \mathcal{A}]$, which proves the claim.

For each $i \in [n]$, let $d_i' \in \mathbb{Z}_N$ be the unique element that satisfies $d_i' = \gamma\, d_i \pmod p$ and $d_i' = d_i \pmod q$. Now, for $i \in [n]$, define

$$(d_{i,0}^*, d_{i,1}^*) = \begin{cases} (a_i, d_i'), & \text{if } x_i^* = 0 \\ (d_i', a_i), & \text{if } x_i^* = 1 \end{cases}, \tag{G.2}$$

where the values a_i are implicit from the n-MDHSD challenge. We now show that $\mathcal{B}$'s responses to the constrain, evaluation, and challenge queries are consistent with the responses in the real game where the msk sampled by the challenger is given by the following:

$$\mathsf{msk}^* = \left(p, q, g_1, g_{p,1}, g_{p,n}, \left\{d_{i,0}^*, d_{i,1}^*\right\}_{i\in[n]}\right). \tag{G.3}$$

Since the groups $\vec{\mathbb{G}}$, generators $\{g_{p,i}, g_{q,i}\}_{i\in[n]}$, and exponents $\{a_i, d_i\}_{i\in[n]}$ are drawn from the same distribution as in the $\mathsf{cPRF.Setup}$ algorithm, we have that msk^* is distributed identically as the master secret key generated in the real scheme.

Now, we show that algorithm $\mathcal{B}$ correctly simulates the constrain oracle—that is, the real distribution of sk is identical to the output of $\mathsf{cPRF.Puncture}(\mathsf{msk}^*, s)$. To do this, let $\mathsf{sk} = \{D_{i,0}, D_{i,1}\}_{i\in[n]}$ be the punctured key output by honestly invoking $\mathsf{cPRF.Puncture}(\mathsf{msk}^*, s)$, and let $\widetilde{\mathsf{sk}} = \left\{\widetilde{D}_{i,0}, \widetilde{D}_{i,1}\right\}_{i\in[n]}$ be the punctured key simulated by $\mathcal{B}$. For each $i \in [n]$ and $b \in \{0,1\}$, we write

$$D_{i,b} = g_{p,1}^{u_{i,b}} g_{q,1}^{v_{i,b}} \qquad \text{and} \qquad \widetilde{D}_{i,b} = g_{p,1}^{\widetilde{u}_{i,b}} g_{q,1}^{\widetilde{v}_{i,b}}.$$

Note that in both the real and simulated distributions, the components in $\mathbb{G}_{p,1}$ and $\mathbb{G}_{q,1}$ are independent (by the Chinese Remainder Theorem). Hence, proving correctness of the simulation for the constrain oracle is equivalent to showing that both

$$\{u_{i,0}, u_{i,1}\}_{i\in[n]} \equiv \{\widetilde{u}_{i,0}, \widetilde{u}_{i,1}\}_{i\in[n]} \qquad \text{and} \qquad \{v_{i,0}, v_{i,1}\}_{i\in[n]} \equiv \{\widetilde{v}_{i,0}, \widetilde{v}_{i,1}\}_{i\in[n]}.$$

Consider the components in $\mathbb{G}_{p,1}$. We note that since the adversary is admissible (Definition 2.4), $s = x^*$. Then, by the definition of $d^*_{i,b}$ from Equation G.2, and the fact that $d'_i = \gamma\, d_i \pmod p$, we have that

$$(u_{i,0}, u_{i,1}) = \begin{cases} (a_i,\, \gamma\, d_i) & \text{if } s_i = 0 \\ (\gamma\, d_i,\, a_i), & \text{if } s_i = 1 \end{cases},$$

But since $h = g_{p,1}^{\gamma}$, this is precisely the distribution of $\widetilde{u}_{i,b}$ constructed by $\mathcal{B}$ from Equation (G.1). Thus, $\{u_{i,0}, u_{i,1}\}_{i\in[n]}$ and $\{\widetilde{u}_{i,0}, \widetilde{u}_{i,1}\}_{i\in[n]}$ are identically distributed.

Now, we consider the components in the $\mathbb{G}_q$ subgroup independently and show that $\{v_{i,0}, v_{i,1}\}_{i\in[n]} \equiv \{\widetilde{v}_{i,0}, \widetilde{v}_{i,1}\}_{i\in[n]}$. In the real scheme,

$$(v_{i,0}, v_{i,1}) = \begin{cases} (a_i,\, 0), & \text{if } s_i = 0 \\ (0,\, a_i), & \text{if } s_i = 1 \end{cases},$$

Once again, this precisely coincides with the distribution chosen by $\mathcal{B}$. We conclude that $\{v_{i,0}, v_{i,1}\}_{i\in[n]}$ and $\{\widetilde{v}_{i,0}, \widetilde{v}_{i,1}\}_{i\in[n]}$ are distributed identically. This completes the proof that algorithm $\mathcal{B}$ has perfectly simulated the responses to the constrain oracle queries. Given that $\mathcal{B}$ correctly simulates the responses to the constrain oracle queries, the fact that $\mathcal{B}$ also correctly simulates the responses to evaluation oracle queries follows from trivially by correctness of Π_{puncPRF} (Theorem 5.2).

For the last step, we claim that the simulator correctly simulates the challenge oracle. To see this, note that if $T = g_{p,n}^z$, then the output of $\mathsf{cPRF.Eval}(\mathsf{msk}^*, x)$ is given by

$$y = g_{p,n}^{\prod_{i\in[n]} d^*_{i,x_i}} = g_{p,n}^{\prod_{i\in[n]} a_i} = g_{p,n}^z,$$

and if $T = g_n^s$, since s is uniform over $\mathbb{Z}_N$, the challenge T is also uniform in $\mathbb{G}_n$. Hence, we have shown that algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Expt}_0^{\mathsf{cPRF}}$ for $\mathcal{A}$ when $T = g_{p,n}^z$ and that algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Expt}_1^{\mathsf{cPRF}}$ for $\mathcal{A}$ when T is distributed uniformly. This allows us to conclude that we conclude that the advantage of $\mathcal{B}$ in breaking the n-MDHSD assumption is precisely $\mathsf{Adv}^{\mathsf{cPRF}}[\Pi_{\mathsf{puncPRF}}, \mathcal{A}]$. □

G.3 Proof of Theorem 5.4

For $b \in \{0,1\}$, we present a series of hybrid experiments $\mathsf{H}_0^{(b)}, \ldots, \mathsf{H}_n^{(b)}$ between a challenge and an adversary $\mathcal{A}$ which makes a single challenge query and no evaluation queries. We will define $\mathsf{H}_0^{(b)}$ to behave identically to $\mathsf{Expt}_b^{\mathsf{cpriv}}$ for Π_{puncPRF} and $\mathsf{H}_n^{(b)}$ to behave independently of b.

Hybrid $\mathsf{H}_0^{(b)}$. First, the challenger computes $\mathsf{msk} \leftarrow \mathsf{cPRF.Setup}(1^\lambda)$, where msk is as described in Equation (5.1). The adversary then queries the challenge oracle on two points $s^{(0)}, s^{(1)} \in \{0,1\}^n$, to which the challenger replies with $\mathsf{sk} \leftarrow \mathsf{cPRF.Puncture}(\mathsf{msk}, s^{(b)})$. In particular, the challenger first constructs, for all $i \in [n]$,

$$(D_{i,0}, D_{i,1}) = \begin{cases} (g_1^{d_{i,0}}, g_{p,1}^{d_{i,1}}), & \text{if } s_i^{(b)} = 0 \\ (g_{p,1}^{d_{i,0}}, g_1^{d_{i,1}}), & \text{if } s_i^{(b)} = 1 \end{cases},$$

where $d_{i,0}$ and $d_{i,1}$ are components of msk. The challenger gives the punctured key $\mathsf{sk} = \{(D_{i,0}), (D_{i,1})\}_{i \in [n]}$ to $\mathcal{A}$. Afterwards, the adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

Hybrid $\mathsf{H}_j^{(b)}$ for $j \in [n]$. In hybrid $\mathsf{H}_j^{(b)}$, the challenger behaves exactly as in hybrid $\mathsf{H}_{j-1}^{(b)}$, except that the challenger sets

$$(D_{j,0}, D_{j,1}) = (g_1^{d_{j,0}}, g_1^{d_{j,1}}),$$

and the values $D_{i,b}$ for all other indices $i \in [n] \setminus \{j\}$ remain the same.

We show that under the Subgroup Decision assumption (Definition A.2), hybrids $\mathsf{H}_{j-1}^{(b)}$ and $\mathsf{H}_j^{(b)}$ are computationally indistinguishable for all $j \in [n]$ and $b \in \{0,1\}$. Let $\mathcal{A}$ be an efficient adversary that is able to distinguish hybrid $\mathsf{H}_{j-1}^{(b)}$ from $\mathsf{H}_j^{(b)}$ with advantage ε. We construct an algorithm $\mathcal{B}$ that breaks the Subgroup Decision assumption with the same advantage ε. Algorithm $\mathcal{B}$ is given a Subgroup Decision challenge of the form

$$(\vec{\mathbb{G}}, N, g_1, g_{p,1}^{\gamma}, T),$$

where γ is uniform over $\mathbb{Z}_p$, and its goal is to determine if $T = g_1^r$ or if $T = g_{p,1}^r$ for a random $r \in \mathbb{Z}_N$. Algorithm $\mathcal{B}$ begins by running $\mathcal{A}$. First, the adversary $\mathcal{A}$ sends two input strings $s^{(0)}, s^{(1)} \in \{0,1\}^n$ to $\mathcal{B}$. For $i \in [n]$, algorithm $\mathcal{B}$ chooses random exponents $d_{i,0}, d_{i,1} \xleftarrow{R} \mathbb{Z}_N$. For all $i < j$, $\mathcal{B}$ sets $(D_{i,0}, D_{i,1}) = (g_1^{d_{i,0}}, g_1^{d_{i,1}})$, and for all $i > j$, $\mathcal{B}$ sets $(D_{i,0}, D_{i,1})$ as follows:

$$(D_{i,0}, D_{i,1}) = \begin{cases} (g_1^{d_{i,0}}, g_{p,1}^{d_{i,1}}), & \text{if } s_i^{(b)} = 0 \\ (g_{p,1}^{d_{i,0}}, g_1^{d_{i,1}}), & \text{if } s_i^{(b)} = 1 \end{cases}$$

Finally, it sets

$$(D_{j,0}, D_{j,1}) = \begin{cases} (g_1^{d_{j,0}}, T), & \text{if } s_j^{(b)} = 0 \\ (T, g_1^{d_{j,1}}), & \text{if } s_j^{(b)} = 1 \end{cases}.$$

First, note that for all $i \neq j$, the components $(D_{i,0}, D_{i,1})$ are distributed identically to the components in $\mathsf{H}_{j-1}^{(b)}$ and $\mathsf{H}_j^{(b)}$. Next, if $T = g^r$, then T is uniform in $\mathbb{G}_1$, and so $(D_{j,0}, D_{j,1})$ consists of two uniformly distributed elements in $\mathbb{G}_1$, which means that $\mathcal{B}$ has simulated $\mathsf{H}_j^{(b)}$. On the other hand, if

$T = g_p^r$, then T is a uniform random element in $\mathbb{G}_{p,1}$, so $\mathcal{B}$ has simulated $\mathsf{H}_{j-1}^{(b)}$. Thus, if $\mathcal{A}$ is able to distinguish $\mathsf{H}_{j-1}^{(b)}$ from $\mathsf{H}_j^{(b)}$ with advantage ε for any $j \in [n]$, then $\mathcal{B}$ is able to break the Subgroup Decision assumption with the same advantage ε.

Finally, we note that in $\mathsf{H}_n^{(b)}$, the challenger behaves independently of b, and hence the adversary has no advantage in guessing b correctly. By taking a union bound, we therefore conclude that for all efficient adversaries $\mathcal{A}$ that participates in the experiment $\mathsf{Expt}_b^{\mathsf{cpriv}}$, there exists an adversary $\mathcal{B}$ that breaks the Subgroup Decision assumption with advantage at least $\mathsf{Adv}^{\mathsf{ppriv}}[\Pi_{\mathsf{puncPRF}}, \mathcal{A}]/n$. □

H Proofs from Section 6.1

H.1 Proof of Theorem 6.6

Let $(\mathsf{sk}, \mathsf{dk}) \leftarrow \mathsf{DE.Setup}(1^\lambda)$. Take any message $m \in \mathcal{M}$ and let $\mathsf{ct} \leftarrow \mathsf{DE.Encrypt}(\mathsf{sk}, m)$. Then, $\mathsf{ct} = (r, y \oplus m)$ for $r \in \{0,1\}^n$ and $y = \mathsf{cPRF.ConstrainEval}(\mathsf{sk}, r)$. Then,

$$\mathsf{DE.Decrypt}(\mathsf{sk}, \mathsf{ct}) = \mathsf{cPRF.ConstrainEval}(\mathsf{sk}, r) \oplus (y \oplus m) = y \oplus y \oplus m = m.$$

Correctness follows. □

H.2 Proof of Theorem 6.7

Semantic security follows directly from PRF security. Let $(\mathsf{sk}, \mathsf{msk})$ be the keys the challenger generates via $\mathsf{DE.Setup}$ at the beginning of the security game. Suppose the adversary makes q encryption queries. Let $r_1, \ldots, r_q \in \{0,1\}^n$ be the random points sampled by $\mathsf{DE.Encrypt}(\mathsf{sk}, \cdot)$ when responding to the q queries. Since $n = \Omega(\lambda)$ and $q = \mathrm{poly}(\lambda)$, with probability $1 - \mathrm{negl}(\lambda)$, the points $r_1, \ldots, r_q$ sampled by $\mathsf{DE.Encrypt}(\mathsf{sk}, \cdot)$ will be unique, and moreover for all $i \in [q]$, $r_i \neq x$ where x is the punctured point in the secret key. By correctness of Π_{pprf}, we have that $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}, r_i) = \mathsf{cPRF.Eval}(\mathsf{msk}, r_i)$ for all $i \in [q]$. Semantic security now follows from a standard hybrid argument and PRF security of Π_{pprf}. □

H.3 Proof of Theorem 6.8

We proceed via a hybrid argument. First, we define the hybrid experiments:

- **Hybrid H_0.** This is the real experiment $\mathsf{Expt}_0^{\mathsf{DE}}$ (Definition 6.4).
- **Hybrid H_1.** This is identical to H_0, except when constructing ct_{i^*}, the challenger chooses a random $r \xleftarrow{\mathrm{R}} \{0,1\}^n$ and sets $\mathsf{ct}_{i^*} = (r_{i^*}, r \oplus m_{i^*})$.
- **Hybrid H_2.** This is identical to H_1, except the challenger sends $\left(\mathsf{sk}, \{\mathsf{ct}_i\}_{i \in [q]}\right)$ to the adversary.
- **Hybrid H_3.** This is the real experiment $\mathsf{Expt}_1^{\mathsf{DE}}$ (Definition 6.4).

As usual, for an adversary $\mathcal{A}$, we write $\mathsf{H}_i(\mathcal{A})$ to denote the output of $\mathcal{A}$ in H_i. We now show that each successive pair of hybrids is computationally indistinguishable.

Lemma H.1. *If Π_{pprf} is selectively-secure, then for all efficient adversaries $\mathcal{A}$, the quantity $|\Pr[\mathsf{H}_0(\mathcal{A}) = 1] - \Pr[\mathsf{H}_1(\mathcal{A}) = 1]|$ is negligible.*

Proof. Suppose $\mathcal{A}$ distinguishes hybrids H_0 from H_1 with advantage ε. We use $\mathcal{A}$ to build an adversary $\mathcal{B}$ that breaks the puncturing security of Π_{pprf}. Algorithm $\mathcal{B}$ proceeds as follows.

1. Chooses a random point $z \xleftarrow{R} \{0,1\}^n$ and query the puncturing oracle at z to obtain a punctured key sk_z. Let $\mathsf{sk}' = \mathsf{sk}_z$. Query the challenge oracle to obtain the challenge T.

2. Start running algorithm $\mathcal{A}$. Adversary $\mathcal{A}$ will output a tuple of messages $(m_1, \ldots, m_q)$ and an index $i^* \in [q]$. For each $i \neq i^*$, choose a random point $r'_i \xleftarrow{R} \{0,1\}^n$, and let $t'_i = \mathsf{cPRF.ConstrainEval}(\mathsf{sk}_z, r'_i)$. Set $r'_{i^*} = z$ and set $t'_{i^*} = T$. For each $i \in [q]$, construct the ciphertext $\mathsf{ct}'_i = (r'_i, t'_i \oplus m_i)$.

3. Send $\left(\mathsf{sk}', \{\mathsf{ct}'_i\}_{i\in[q]}\right)$ to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

First, since the r'_i are drawn uniformly and independently from $\{0,1\}^n$, with overwhelming probability, $r'_i \neq z$. By correctness of the puncturing scheme, $t'_i = \mathsf{cPRF.ConstrainEval}(\mathsf{sk}_z, r'_i) = \mathsf{cPRF.Eval}(\mathsf{msk}, r'_i)$, where msk is the master secret key chosen by the puncturing security challenger. We now show that if $T = \mathsf{cPRF.Eval}(\mathsf{msk}, z)$, then $\mathcal{B}$ perfectly simulates H_0 and if T is uniform over $\{0,1\}^n$, then $\mathcal{B}$ perfectly simulates H_1.

- Consider the case where $T = \mathsf{cPRF.Eval}(\mathsf{msk}, z)$. In hybrid H_0, each ciphertext ct_i is a tuple of the form $(r_i, t_i \oplus m_i)$ where $r_i \xleftarrow{R} \{0,1\}^n$, $t_i = \mathsf{cPRF.ConstrainEval}(\mathsf{sk}_x, r_i)$, $\mathsf{sk}_x \leftarrow \mathsf{cPRF.Puncture}(\mathsf{msk}, x)$, and $x \xleftarrow{R} \{0,1\}^n$. Since the r_i are sampled uniformly and independently from $\{0,1\}^n$, with all but negligible probability, $r_i \neq x$. By correctness of the puncturing scheme, $t_i = \mathsf{cPRF.ConstrainEval}(\mathsf{sk}_x, r_i) = \mathsf{cPRF.Eval}(\mathsf{msk}, r_i)$. At the end of the experiment, the challenger gives a punctured key $\mathsf{sk}_{r_{i^*}} = \mathsf{cPRF.Puncture}(\mathsf{msk}, r_{i^*})$ together with the ciphertexts $\{\mathsf{ct}_i\}_{i\in[q]}$ to the adversary. In the simulation, for all $i \in [q]$, each r'_i is sampled uniformly and independently from $\{0,1\}^n$ and moreover, if $T = \mathsf{cPRF.Eval}(\mathsf{msk}, z)$, then $t'_i = \mathsf{cPRF.Eval}(\mathsf{msk}, r'_i)$, exactly as in H_0. Moreover, $\mathcal{B}$ constructs the secret key sk' exactly as the challenger does in H_0, so we conclude that $\mathcal{B}$ perfectly simulates H_0 for $\mathcal{A}$.

- Suppose instead that T was uniform in $\{0,1\}^\ell$. As in hybrid H_0, each ciphertext in H_1 can be written as a tuple $(r_i, t_i \oplus m_i)$ where $r_i \xleftarrow{R} \{0,1\}^n$, $t_i = \mathsf{cPRF.Eval}(\mathsf{msk}, r_i)$ for all $i \neq i^*$, and $t_{i^*} \xleftarrow{R} \{0,1\}^\ell$. If T is uniform in $\{0,1\}^\ell$, then the ciphertexts constructed by $\mathcal{B}$ precisely coincides with this distribution. Moreover, in hybrid H_1, the challenger gives the adversary a secret key punctured at r_{i^*}, and in the simulation, algorithm $\mathcal{B}$ gives the adversary a key punctured at $z = r'_{i^*}$. We conclude that if T is random, then $\mathcal{B}$ perfectly simulates H_1 for $\mathcal{A}$.

Thus, if $\mathcal{A}$ is able to distinguish hybrids H_0 from H_1 with advantage ε, then $\mathsf{Adv}^{\mathsf{cPRF}}[\Pi_{\mathsf{pprf}}, \mathcal{B}] = \varepsilon$. This proves the lemma. □

Lemma H.2. *If Π_{pprf} is 1-key private (Definition 2.10), then for all efficient adversaries $\mathcal{A}$, the quantity $|\Pr[\mathsf{H}_1(\mathcal{A}) = 1] - \Pr[\mathsf{H}_2(\mathcal{A}) = 1]|$ is negligible.*

Proof. Suppose $\mathcal{A}$ can distinguish H_1 from H_2 with advantage ε. We use $\mathcal{A}$ to build an adversary $\mathcal{B}$ that breaks the 1-key privacy of Π_{pprf}. Algorithm $\mathcal{B}$ proceeds as follows.

1. Choose two points $x_0, x_1 \xleftarrow{R} \{0,1\}^n$, and submit them to the 1-key privacy challenger to obtain a punctured key sk_b. Define $\mathsf{sk}' = \mathsf{sk}_b$.

2. Start running algorithm $\mathcal{A}$. Adversary $\mathcal{A}$ will output a tuple of messages $(m_1, \ldots, m_q)$ and an index $i^* \in [q]$. For $i \neq i^*$, choose $r'_i \xleftarrow{R} \{0,1\}^n$, and set $t'_i = \mathsf{cPRF.ConstrainEval}(\mathsf{sk}_b, r'_i)$. Set $r'_{i^*} = x_0$, and choose $t'_{i^*} \xleftarrow{R} \{0,1\}^\ell$. For all $i \in [q]$, construct the ciphertexts $\mathsf{ct}'_i = (r'_i, t'_i \oplus m_i)$.

3. Send $\left(\mathsf{sk}', \{\mathsf{ct}'_i\}_{i \in [q]}\right)$ to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

Since each r'_i is chosen uniformly and independently from $\{0,1\}^n$, with all but negligible probability, $r'_1, \ldots, r'_q \notin \{x_0, x_1\}$. Thus, by correctness of the punctured PRF, $\mathsf{cPRF.ConstrainEval}(\mathsf{sk}_b, r'_i) = \mathsf{cPRF.Eval}(\mathsf{msk}, r'_i)$, where msk is the underlying master secret key chosen by the 1-key privacy challenger. We now show that if $b = 0$, then $\mathcal{B}$ perfectly simulates for $\mathcal{A}$ hybrid H_1 and if $b = 1$, then $\mathcal{B}$ perfectly simulates for $\mathcal{A}$ hybrid H_2.

- Suppose $b = 0$. Then in the simulation, $\mathsf{ct}'_i = (r'_i, t'_i \oplus m_i)$, where for all $i \neq i^*$, $t'_i = \mathsf{cPRF.Eval}(\mathsf{msk}, r'_i)$ and $t'_{i^*} \xleftarrow{R} \{0,1\}^\ell$. In hybrid H_1, the challenger similarly samples randomness $r_i \xleftarrow{R} \{0,1\}^n$ for all $i \in [q]$, and for $i \neq i^*$, sets $t_i = \mathsf{cPRF.ConstrainEval}(\mathsf{sk}_x, r_i) = \mathsf{cPRF.Eval}(\mathsf{msk}, r_i)$. Note that the last equality holds with overwhelming probability since r_i are sampled uniformly and independently in $\{0,1\}^n$, and by correctness of Π_{pprf}. The challenger sets $t_{i^*} \xleftarrow{R} \{0,1\}^\ell$ in hybrid H_1. Thus, the ciphertexts $\mathcal{B}$ constructs in the simulation are distributed identically to those the challenger constructs in H_1. Finally, in H_1, the challenger outputs a secret key punctured at r_{i^*}. In the simulation, when $b = 0$, $\mathcal{B}$ gives $\mathcal{A}$ a secret key punctured at $x_0 = r'_{i^*}$. Since x_0 is sampled from the same distribution as r_{i^*} in H_1, we conclude that $\mathcal{B}$ perfectly simulates H_1 in this case.

- Suppose $b = 1$. The distribution of the ciphertexts is identical to that described in the previous case. Similarly, the ciphertexts in H_0 and H_1 are constructed in exactly the same manner. Thus, we conclude that the distribution of ciphertexts in the simulation are distributed as in H_2. In hybrid H_2, then challenger constructs a key punctured at x, where x is uniform in $\{0,1\}^n$ and independent of the randomness $r_1, \ldots, r_q$ used to construct the ciphertexts. In the simulation, when $b = 1$, $\mathcal{B}$ gives $\mathcal{A}$ a secret key punctured at a uniform random point x_1 independent of the randomness used in constructing the ciphertexts. We conclude that $\mathcal{B}$ correctly simulates H_2 in this case.

We conclude that if $\mathcal{A}$ can distinguish H_1 from H_2 with advantage ε, then $\mathsf{Adv}^{\mathsf{cpriv}}[\Pi_{\mathsf{pprf}}, \mathcal{B}] = \varepsilon$. This proves the lemma. □

Lemma H.3. *For all adversaries $\mathcal{A}$,* $\Pr[\mathsf{H}_2(\mathcal{A}) = 1] = \Pr[\mathsf{H}_3(\mathcal{A}) = 1]$.

Proof. In H_2, $\mathsf{ct}_{i^*} = (r_{i^*}, r \oplus m_{i^*})$, where r is chosen independently and uniformly from $\{0,1\}^\ell$. In H_3, $\mathsf{ct}_{i^*} = (r_{i^*}, m^* \oplus \mathsf{cPRF.ConstrainEval}(\mathsf{sk}_x, r_{i^*}))$, where m^* is chosen independently and uniformly from $\{0,1\}^\ell$. In both cases, both components of ct_{i^*} are distributed uniformly and independently of all other ct_i for $i \neq i^*$. For $i \neq i^*$, the ciphertexts ct_i are generated identically in H_2 and H_3. The claim follows. □

Combining Lemmas H.1 through H.3, we conclude that Π_{DE} is a deniable encryption scheme. □

I Proofs from Section 6.2

I.1 Proof of Theorem 6.17

We show that the Π_{wm} is correct (Definition 6.11). Take a message $m \in \{0,1\}^\ell$. Let $\mathsf{msk} = (\mathsf{k}, z_1, \ldots, z_d) \leftarrow \mathsf{WM.Setup}(1^\lambda)$ and $(k, C) \leftarrow \mathsf{WM.Mark}(\mathsf{msk}, m)$. We now check each of the requirements:

- Since k is generated by invoking the setup algorithm for Π_{pprf}, it is properly distributed and the first requirement is satisfied.
- Let $\mathbf{w} \in (\{0,1\}^m)^d$ be a vector where $\mathbf{w}_i = \mathsf{pPRF.Eval}(k, z_i)$ for each $i \in [d]$. Let $(x^*, y^*, \tau^*) = F(\mathsf{k}, \mathbf{w})$. By construction, $C(\cdot) = \mathsf{pPRF.ProgramEval}(\mathsf{sk}_k, \cdot)$ where sk_k is the output of $\mathsf{pPRF.Program}(k, x^*, (y^*, m \oplus \tau^*))$. By correctness of Π_{pprf}, for all inputs $x \neq x^*$, we have that

$$C(x) = \mathsf{pPRF.ProgramEval}(\mathsf{sk}_k, x) = \mathsf{pPRF.Eval}(k, x). \tag{I.1}$$

  Certainly then, $C(\cdot) \sim_f \mathsf{pPRF.Eval}(k, \cdot)$ where $1/f = \mathrm{negl}(\lambda)$, so the second requirement is satisfied.
- Since $(x^*, y^*, \tau^*) = F(\mathsf{k}, \mathbf{w})$, by PRF security of F, we have for all $i \in [d]$, $\Pr[x^* = z_i] = \mathrm{negl}(\lambda)$. Since $d = \mathrm{poly}(\lambda)$, by a union bound, with overwhelming probability, $x^* \neq z_i$ for all $i \in [d]$, and so by Equation (I.1), we conclude $C(z_i) = \mathsf{pPRF.Eval}(k, z_i)$ for all $i \in [d]$. Thus, with overwhelming probability, the tuple (x, y, τ) computed by the verification algorithm $\mathsf{WM.Verify}(\mathsf{msk}, C)$ satisfies $(x, y, \tau) = (x^*, y^*, \tau^*)$. The verification algorithm then computes $C(x) = C(x^*) = (y^*, m \oplus \tau^*)$, and outputs $(m \oplus \tau^*) \oplus \tau^* = m$, so the third requirement holds.

Since all three requirements hold, we conclude that Π_{wm} is correct. □

I.2 Proof of Theorems 6.18 and 6.19

To prove security of Π_{wm} we construct a series of hybrid experiments between a challenger and an adversary $\mathcal{A}$, where $\mathcal{A}$ is given access to a marking oracle and a challenge oracle. We will assume without loss of generality that $\mathcal{A}$ only makes *distinct* marking queries, since repeated marking queries can be responded to by simply recalling the response from the previous such query. We write $\mathsf{H}_i(\mathcal{A})$ to denote the output of the challenger in hybrid H_i. The first hybrid experiment, H_0, is identical to $\mathsf{Expt}_{\mathsf{wm}}$; we recall it here for clarity.

Definition I.1 (Hybrid H_0)**.** Let $\mathsf{H}_0 = \mathsf{Expt}_{\mathsf{wm}}$, so H_0 represents the real watermarking game. For clarity, we recall the challenger's behavior in $\mathsf{Expt}_{\mathsf{wm}}$ for the construction Π_{wm} here. First, the challenger chooses $\mathsf{k} \xleftarrow{\mathrm{R}} \mathcal{K}$ and $(z_1, \ldots, z_d) \xleftarrow{\mathrm{R}} (\{0,1\}^n)^d$ uniformly at random. Then, the challenger responds to each marking or challenge oracle query made by $\mathcal{A}$ in the following manner.

- Marking oracle: On input a message $m \in \{0,1\}^\ell$, the challenger generates $k \leftarrow \mathsf{pPRF.Setup}(1^\lambda)$, and computes $\mathbf{w} = (\mathsf{pPRF.Eval}(k, z_1), \ldots, \mathsf{pPRF.Eval}(k, z_d))$. It then sets $(x, y, \tau) = F(\mathsf{k}, \mathbf{w})$ and $v = m \oplus \tau$. Then, it computes $\mathsf{sk}_k \leftarrow \mathsf{pPRF.Program}(k, x, (y, v))$, sets the circuit $C(\cdot) = \mathsf{pPRF.ProgramEval}(\mathsf{sk}_k, \cdot)$ and returns (k, C) to $\mathcal{A}$.

- Challenge oracle: On input a message $m \in \{0,1\}^\ell$, the challenger generates a key by computing $k^* \xleftarrow{R} \mathsf{pPRF.Setup}(1^\lambda)$, and then computes $\mathbf{w} = (\mathsf{pPRF.Eval}(k^*, z_1), \ldots, \mathsf{pPRF.Eval}(k^*, z_d))$ and sets $(x^*, y^*, \tau^*) = F(\mathsf{k}, \mathbf{w})$ and $v^* = m \oplus \tau^*$. Then, it computes the key $\mathsf{sk}_{k^*} \leftarrow \mathsf{pPRF.Program}(k^*, x^*, (y^*, v^*))$, sets $C^*(\cdot) = \mathsf{pPRF.ProgramEval}(\mathsf{sk}_{k^*}, \cdot)$. It gives C^* to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs a circuit C'. The challenger computes the tuple $\mathbf{w} = (C'(z_1), \ldots, C'(z_d))$, sets $(x, y, \tau) = F(\mathsf{k}, \mathbf{w})$, and $(y', v) = C'(x)$. Finally, it outputs $v \oplus \tau$ if $y = y'$, and $\perp$ otherwise.

Proof of Theorem 6.18. We proceed via a hybrid argument. First, we define our sequence of hybrid experiments:

- **Hybrid H_1.** This is identical to H_0, except the challenger initializes a set $\mathcal{Z} = \varnothing$ in the setup phase. When responding to marking and challenge queries, the challenger, after computing $\mathbf{w}$, checks to see if $\mathbf{w} \in \mathcal{Z}$. If so, the challenger sets the Bad_1 flag and then continues with the experiment. Otherwise, the challenger adds $\mathbf{w}$ to $\mathcal{Z}$. During the verification step, if the Bad_1 flag has been set, the challenger aborts the experiment and outputs Bad_1. Otherwise, it proceeds as in H_0.

- **Hybrid H_2.** This is identical to H_1, except in the setup phase, the challenger chooses a random function $g : (\{0,1\}^n)^d \to \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\ell$ instead of a random key $\mathsf{k} \in \mathcal{K}$. When responding to marking queries, challenge queries, and in the verification step, the challenger computes the tuple (x, y, τ) by evaluating $g(\mathbf{w})$ rather than $F(\mathsf{k}, \mathbf{w})$. In addition, when responding to the challenge query, if $x^* = z_i$ for some $i \in [d]$, the challenger sets the Bad_2 flag and continues with the experiment. At the beginning of the verification step, if the Bad_2 flag has been set, the challenger aborts the experiments and outputs Bad_2. Otherwise, it proceeds as in H_1.

- **Hybrid H_3.** This is identical to H_2, except when the challenger responds to marking and challenge queries, it samples $(x, y, \tau) \xleftarrow{R} \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\ell$ uniformly rather than computing $g(\mathbf{w})$. Also, in the verification step, after $\mathcal{A}$ outputs a circuit C', the challenger aborts and outputs Bad_3 if for some $i \in [d]$, $C'(z_i) \neq \mathsf{pPRF.ProgramEval}(\mathsf{sk}_{k^*}, z_i)$. Otherwise, the challenger sets $(y', v) = C'(x^*)$ and outputs $v \oplus \tau^*$ if $y' = y^*$, and $\perp$ otherwise.

- **Hybrid H_4.** This is identical to H_3, except that in the verification step, after the challenger performs the check for Bad_3, it computes the tuples $(y', v) = C'(x^*)$ and $(\bar{y}^*, \bar{v}^*) = \mathsf{pPRF.ProgramEval}(\mathsf{sk}_{k^*}, x^*)$. The challenger outputs $v \oplus (\bar{v}^* \oplus m)$ if $y' = \bar{y}^*$ and $\perp$ otherwise. Here, sk_{k^*} is the key the challenger generates when responding to the challenge query.

- **Hybrid H_5.** This is identical to H_4, except when responding to the challenge oracle, the challenger sets $\mathsf{sk}_{k^*} \leftarrow \mathsf{pPRF.Program}(k^*, 0^n, (y^*, v^*))$ instead of $\mathsf{pPRF.Program}(k^*, x^*, (y^*, v^*))$.

We now show that each consecutive pair of hybrid experiments is computationally indistinguishable. In addition, we show in Lemma I.7, that $\Pr[\mathsf{H}_5(\mathcal{A}) \neq m]$ is negligible. This completes the proof of Theorem 6.18. Note that in all of our proofs below, we will implicitly assume that $d = \mathrm{poly}(\lambda)$ in our construction.

Lemma I.2. *If Π_{pprf} is a secure PRF, then for all efficient adversaries $\mathcal{A}$, the absolute difference $|\Pr[\mathsf{H}_0(\mathcal{A}) \neq m] - \Pr[\mathsf{H}_1(\mathcal{A}) \neq m]|$ is negligible.*

Proof. To prove this lemma, it suffices to show that for any efficient adversary $\mathcal{A}$ which makes Q marking oracle and challenge oracle queries and causes H_1 to output Bad_1 with probability ε, we can construct an algorithm $\mathcal{B}$ which wins the PRF security experiment with advantage at least a negligible amount less than ε/Q. Recall in the PRF security game (Definition 2.1), $\mathcal{B}$ is either given oracle access to the PRF or given oracle access to a truly random function and its goal is to distinguish the two experiments.

We construct algorithm $\mathcal{B}$ as follows. First, a random index $j \xleftarrow{R} [Q]$ is selected. Algorithm $\mathcal{B}$ then conducts a faithful simulation of H_1 for the setup phase and the first $j-1$ marking oracle or challenge oracle queries. Finally, on the j^{th} query, for each $i \in [d]$, algorithm $\mathcal{B}$ queries its oracle on each z_i for $i \in [d]$ and receives outputs y_i. Algorithm $\mathcal{B}$ then halts the experiment with $\mathcal{A}$, and if $(y_1, \ldots, y_d) \in \mathcal{Z}$, algorithm $\mathcal{B}$ outputs 0, and otherwise it outputs 1.

By construction, the challenger in H_1 aborts the experiment and outputs Bad_1 if and only if the challenger sets the Bad_1 flag on some round j. Since $\mathcal{A}$ causes H_1 to abort with probability ε, then it must cause the challenger in H_1 to set the Bad_1 flag on the j^{th} query (for a uniformly selected index j) with probability at least ε/Q. Now, if $b = 0$ (if $\mathcal{B}$ is given oracle access to the PRF) then $\mathcal{B}$ has perfectly simulated H_1 up until the j^{th} query, and hence $\mathcal{B}$ will output 0 with probability at least ε/Q. If $b = 1$ (if $\mathcal{B}$ is given oracle access to the truly random function) then the probability that $\mathcal{B}$ outputs 0 is equal to the probability that $(y_1, \ldots, y_d) \in \mathcal{Z}$, which is bounded by $(j-1)/2^m \leq Q/2^m = \text{negl}(\lambda)$, since $Q = \text{poly}(\lambda)$ and $m = \Omega(\lambda)$. □

Lemma I.3. *If F is a secure PRF, then for all efficient adversaries $\mathcal{A}$, the absolute difference $|\Pr[\mathsf{H}_1(\mathcal{A}) \neq m] - \Pr[\mathsf{H}_2(\mathcal{A}) \neq m]|$ is negligible.*

Proof. For any efficient adversary $\mathcal{A}$ which makes Q marking oracle and challenge oracle queries, we construct an algorithm $\mathcal{B}$ for the PRF security experiment. In the PRF security game, adversary $\mathcal{B}$ is given oracle access to either the PRF $F(\mathsf{k}, \cdot)$ for $\mathsf{k} \xleftarrow{R} \mathcal{K}$ or oracle access to a truly random function $f \xleftarrow{R} \mathsf{Funs}((\{0,1\}^n)^d, \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\ell)$. We show that $\mathcal{B}$ can win the PRF security game with advantage at least $|\Pr[\mathsf{H}_1(\mathcal{A}) \neq m] - \Pr[\mathsf{H}_2(\mathcal{A}) \neq m]|$.

Algorithm $\mathcal{B}$ simulates H_2, except that instead of computing $F(\mathsf{k}, \mathbf{w})$, $\mathcal{B}$ instead queries its oracle on $\mathbf{w}$ to obtain a value (x, y, τ), which $\mathcal{B}$ uses to respond to marking and challenge query, as well as in the verification step. If the oracle in the PRF security game implements the PRF, then $\mathcal{B}$ has simulated H_1. If instead the oracle implements a truly random function, then $\mathcal{B}$ has simulated H_2.

To conclude the proof, it suffices to check that the challenger does not output Bad_2 in H_2. By construction, the challenger in H_2 sets Bad_2 only if $g(\mathbf{w})$ outputs (x^*, y^*, τ^*) such that $x^* = z_i$ for some $i \in [d]$. Since g is a truly random function, for any $i \in [d]$, $x^* = z_i$ with probability $1/2^n$. By a union bound, $x^* \neq z_i$ for all $i \in [d]$ with probability $1 - d/2^n = 1 - \text{negl}(\lambda)$. Thus, with overwhelming probability, the challenger does not set Bad_2 in H_2, and the lemma follows. □

Lemma I.4. *For all unremoving admissible adversaries $\mathcal{A}$ (Definition 6.13), the absolute difference $|\Pr[\mathsf{H}_2(\mathcal{A}) \neq m] - \Pr[\mathsf{H}_3(\mathcal{A}) \neq m]|$ is negligible.*

Proof. Since the conditions for outputting Bad_1 and Bad_2 are identical between H_2 and H_3, if the challenger outputs Bad_1 or Bad_2 in one experiment, it produces the same output in the other, and the claim follows. Thus, suppose that the challenger in H_3 does not output Bad_1 or Bad_2. Then, the vectors $\mathbf{w}$ that appear in the marking and challenge queries are all distinct. Hence, the outputs of $g(\mathbf{w})$ are all distributed uniformly and independently in $\{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\ell$, which precisely coincides with the distribution in H_3. It remains to show that the adversary cannot cause H_3

to output Bad_3 with non-negligible probability, and that the remainder of the simulation in the verification step is correct.

Observe that rather than sampling $(z_1, \ldots, z_d) \xleftarrow{R} (\{0,1\}^n)^d$ during the setup phase, the challenger in H_3 could at this point sample $(z_1, \ldots, z_d) \xleftarrow{R} (\{0,1\}^n)^d$ *after* $\mathcal{A}$ outputs the circuit C' during the verification step. This is because the challenger's response to the marking queries and challenge queries are now independent of $(z_1, \ldots, z_d)$.[13] Then, since $\mathcal{A}$ is unremoving admissible, for $z \xleftarrow{R} \{0,1\}^n$, the probability $\Pr[C'(z) \neq \mathsf{pPRF.ProgramEval}(\mathsf{sk}_{k^*}, z)]$ is negligible. By a union bound, the probability that there exists some $i \in [d]$ where $C'(z_i) \neq \mathsf{pPRF.ProgramEval}(\mathsf{sk}_{k^*}, z_i)$ is $d \cdot \mathrm{negl}(\lambda)$. Since $d = \mathrm{poly}(\lambda)$ in our construction, we conclude that the probability that the challenger in H_3 outputs Bad_3 is negligible.

Assuming the challenger in H_3 does not output Bad_3, then $x^* \neq z_i$ for all $i \in [d]$. Then, by correctness of Π_{pprf}, $\mathsf{pPRF.ProgramEval}(\mathsf{sk}_{k^*}, z_i) = \mathsf{pPRF.Eval}(k^*, z_i)$ for all $i \in [d]$. Thus, it is correct to use the randomly sampled values (x^*, y^*, τ^*) from the challenge oracle query as the value for $g(\mathbf{w})$. The remainder of the verification step is implemented identically in H_2 and H_3. The lemma follows. □

Lemma I.5. *For all adversaries $\mathcal{A}$, $\Pr[\mathsf{H}_3(\mathcal{A}) \neq m] = \Pr[\mathsf{H}_4(\mathcal{A}) \neq m]$.*

Proof. By correctness of Π_{pprf}, $\mathsf{pPRF.ProgramEval}(\mathsf{sk}_{k^*}, x^*) = (y^*, v^*)$, so $(\bar{y}^*, \bar{v}^*) = (y^*, v^*)$. In H_3, $v^* = m \oplus \tau^*$ so $\bar{v}^* \oplus m = v^* \oplus m = \tau^*$. Thus, hybrids H_3 and H_4 are identical experiments. □

Lemma I.6. *If Π_{pprf} is 1-key private, then for all efficient adversaries $\mathcal{A}$, the absolute difference $|\Pr[\mathsf{H}_4(\mathcal{A}) \neq m] - \Pr[\mathsf{H}_5(\mathcal{A}) \neq m]|$ is negligible.*

Proof. Suppose $|\Pr[\mathsf{H}_4(\mathcal{A}) \neq m] - \Pr[\mathsf{H}_5(\mathcal{A}) \neq m]| = \varepsilon$. We use $\mathcal{A}$ to build an adversary $\mathcal{B}$ that can distinguish between experiments $\mathsf{Expt}_0^{\mathsf{ppriv}}$ from $\mathsf{Expt}_1^{\mathsf{ppriv}}$ (Definition C.6) with advantage at least ε.

At the beginning of the 1-time privacy game for the programmable PRF, the challenger computes $\mathsf{msk} \leftarrow \mathsf{pPRF.Setup}(1^\lambda)$ and samples a point $(y^*, \tau^*) \xleftarrow{R} \{0,1\}^m \times \{0,1\}^\ell$ uniformly at random. Algorithm $\mathcal{B}$ simulates the setup phase as described in H_4 and H_5.

Whenever $\mathcal{A}$ makes a marking oracle query, $\mathcal{B}$ can simulate the response perfectly without interacting with the challenger for $\mathsf{Expt}_b^{\mathsf{ppriv}}$. At some point, $\mathcal{A}$ will make a challenge oracle query on input $m \in \{0,1\}^\ell$. First, $\mathcal{B}$ samples $x^* \xleftarrow{R} \{0,1\}^n$ uniformly at random, and then submits the two inputs $x_0 = x^*$ and $x_1 = 0^n$ to the challenger for $\mathsf{Expt}_b^{\mathsf{ppriv}}$. The challenger responds with the key $\mathsf{sk}_{k^*} = \mathsf{pPRF.Program}(\mathsf{msk}, x_b, (y^*, \tau^*))$, which is returned to $\mathcal{A}$. The verification procedure is identical in H_3 and H_4. We argue that when $b = 0$, algorithm $\mathcal{B}$ perfectly simulates hybrid H_3 and when $b = 1$, algorithm $\mathcal{B}$ perfectly simulates hybrid H_4. First, in both H_3 and H_4, the value at the programmed point is (y^*, v^*), where y^* is uniform over $\{0,1\}^m$, $v^* = m \oplus \tau^*$, and τ^* is uniform over $\{0,1\}^\ell$. Since τ^* is independent of m, $v^* = m \oplus \tau^*$ is also uniform over $\{0,1\}^\ell$. When $b = 0$, the adversary is given an evaluation circuit associated with the programmed key $\mathsf{pPRF.Program}(\mathsf{msk}, x^*, (y^*, \tau^*))$, where x^*, y^*, and τ^* are all uniform over their respective sets, and msk is the output of $\mathsf{pPRF.Setup}$. Thus, $\mathcal{B}$ correctly simulates experiment H_3. When $b = 1$, the adversary is given an evaluation circuit associated with the programmed key $\mathsf{pPRF.Program}(\mathsf{msk}, 0^n, (y^*, \tau^*))$, where y^* and τ^* are uniform over their respective sets. This is the distribution in H_4. □

[13] While the conditions for setting the flags Bad_1 and Bad_2 do depend on $(z_1, \ldots, z_d)$, the setting of the flag has no effect on the experiment until the verification phase. Thus, it is equivalent to check whether the flags Bad_1 and Bad_2 are set after sampling $(z_1, \ldots, z_d)$ in the verification phase.

Lemma I.7. *For all unremoving admissible adversaries $\mathcal{A}$, if $m \in \{0,1\}^\ell$ is the message submitted to the challenge oracle query, then $\mathsf{Pr}[\mathsf{H}_5(\mathcal{A}) \neq m]$ is negligible.*

Proof. Rather than sampling $x^* \xleftarrow{R} \{0,1\}^n$ when responding to the challenge query, the challenger in H_5 could instead sample $x^* \xleftarrow{R} \{0,1\}^n$ *after* $\mathcal{A}$ outputs the circuit C' during the verification step. This is because the response to the challenge query is now independent of x^*. Let $x^* \xleftarrow{R} \{0,1\}^n$ and $(y', v) = C'(x^*)$. Since $\mathcal{A}$ is unremoving admissible (Definition 6.13), the probability that $(y', v) \neq \mathsf{pPRF.ProgramEval}(\mathsf{sk}_{k^*}, x^*)$ is negligible. This means that with overwhelming probability, $(y', v) = \mathsf{pPRF.ProgramEval}(\mathsf{sk}_{k^*}, x^*) = (\bar{y}^*, \bar{v}^*)$. In the verification step, the challenger affirms that $y' = \bar{y}^*$, and outputs $v \oplus (\bar{v}^* \oplus m) = m$. Hence, the probability that H_5 does not output m is negligible. □

By combining Lemmas I.2-I.7, we have shown that Π_{wm} satisfies unremovable watermarking security. □

Proof of Theorem 6.19. We proceed via a hybrid argument. First, we describe our sequence of hybrids.

- **Hybrid H_1.** This is the same as hybrid H_1 from the proof of Theorem 6.18.
- **Hybrid H_2.** This is identical to H_1, except in the setup phase, the challenger chooses a random function $g : (\{0,1\}^n)^d \to \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\ell$ instead of a random key $\mathsf{k} \in \mathcal{K}$. When responding to marking queries, challenge queries, and in the verification step, the challenger computes the tuple (x, y, τ) by evaluating $g(\mathbf{w})$ rather than $F(\mathsf{k}, \mathbf{w})$.
- **Hybrid H_3.** This is identical to H_2, except when the challenger responds to marking queries, it samples $(x, y, \tau) \xleftarrow{R} \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\ell$ uniformly rather than computing $g(\mathbf{w})$. In the verification step, after computing $\mathbf{w} = (C'(z_1), \ldots, C'(z_d))$, where C' is the circuit output by the adversary, the challenger aborts the experiment and outputs Bad_2 if $\mathbf{w} \in \mathcal{Z}$ (where $\mathcal{Z}$ is specified in H_1). Otherwise, it proceeds as in H_2.
- **Hybrid H_4.** This is identical to H_3, except in each of the marking queries and in the verification step, the challenger samples $(x, y, \tau) \xleftarrow{R} \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\ell$ independently and uniformly at random instead of setting $(x, y, \tau) = g(\mathbf{w})$.

We now show that each consecutive pairs of hybrid experiments is computationally indistinguishable. In addition, in Lemma I.12, we show that $\mathsf{Pr}[\mathsf{H}_4(\mathcal{A}) \neq \perp]$ is negligible, which completes the proof of Theorem 6.19.

Lemma I.8. *If Π_{pprf} is secure, then for all efficient adversaries $\mathcal{A}$, $|\Pr[\mathsf{H}_0(\mathcal{A}) \neq \perp] - \Pr[\mathsf{H}_1(\mathcal{A}) \neq \perp]|$ is negligible.*

Proof. The proof of this statement is identical to that of Lemma I.2. □

Lemma I.9. *If F is a secure PRF, then $|\Pr[\mathsf{H}_1(\mathcal{A}) \neq \perp] - \Pr[\mathsf{H}_2(\mathcal{A}) \neq \perp]|$ is negligible.*

Proof. Hybrid H_2 is identical to the corresponding hybrid in the proof of Theorem 6.18, except the challenger does not perform the check for Bad_2. The proof thus follows same argument as that given in the proof of Lemma I.3. □

Lemma I.10. *For all δ-unforging admissible adversaries $\mathcal{A}$ where $\delta = 1/\mathrm{poly}(\lambda)$ (Definition 6.15), the absolute difference $|\Pr[\mathsf{H}_2(\mathcal{A}) \neq \perp] - \Pr[\mathsf{H}_3(\mathcal{A}) \neq \perp]|$ is negligible.*

Proof. This proof is very similar to the proof of Lemma I.4. Since the condition for outputting Bad_1 is identical between H_2 and H_3, if the challenger outputs Bad_1 in one experiment, then it produces the same output in the other, and the claim follows. Thus, suppose that the challenger does not output Bad_1. Then, the vectors $\mathbf{w}$ that appear in the marking queries are all distinct, and so the outputs of $g(\mathbf{w})$ in H_2 are all distributed uniformly and independently in $\{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\ell$, which precisely coincides with the distribution in H_3. It suffices to show that the adversary cannot cause the challenger in H_3 to output Bad_2 with non-negligible probability.

Let $Q = \mathrm{poly}(\lambda)$ be the number of marking queries the adversary makes, and let $C_1, \ldots, C_Q$ be the circuits the challenger outputs in response to the mark queries. Let C' be the circuit the adversary outputs, and for $i \in [Q]$, let S_i be the set of points on which $C'(\cdot)$ and $C_i(\cdot)$ differ. Since $\mathcal{A}$ is δ-unforging admissible, for all $i \in [Q]$, $|S_i|/2^n \geq \delta$. Now, as argued in the proof of Lemma I.4, the challenger in H_3 can sample $(z_1, \ldots, z_d) \xleftarrow{\mathrm{R}} (\{0,1\}^n)^d$ *after* $\mathcal{A}$ outputs its circuit C' during the verification step. In this case then, for each $j \in d$, $\Pr[z_j \in S_i] = |S_i|/2^n \geq \delta$. It follows that

$$\Pr\left[\forall j \in [d] : z_j \notin S_i\right] = \left(1 - \frac{|S_i|}{2^n}\right)^d \leq (1-\delta)^{\lambda/\delta} = \mathrm{negl}(\lambda),$$

where we have used the fact that $d = \lambda/\delta$ and $\delta = 1/\mathrm{poly}(\lambda)$. Since this holds for all $i \in [Q]$, we conclude that with overwhelming probability, it is the case that for each $i \in [Q]$, there exists some j for which $z_j \in S_i$, or equivalently, where $C'(z_j) \neq C_i(z_j)$. Since $\mathbf{w} = (C'(z_1), \ldots, C'(z_d))$, we conclude that with overwhelming probability $\mathbf{w} \notin \mathcal{Z}$. The claim follows. □

Lemma I.11. *For all adversaries $\mathcal{A}$, $\Pr[\mathsf{H}_3(\mathcal{A}) \neq \perp] = \Pr[\mathsf{H}_4(\mathcal{A}) \neq \perp]$.*

Proof. This follows from the fact that, conditioned on the challenger not outputting Bad_1 and Bad_2, the function g is never evaluated on the same input twice. Hence, the inputs to g are always distinct, in which case, the outputs (x, y, τ) are distributed uniformly and independently. □

Lemma I.12. *For all adversaries $\mathcal{A}$, $\Pr[\mathsf{H}_5(\mathcal{A}) \neq \perp]$ is negligible.*

Proof. In the verification step, we can think of the tuple (x, y, τ) as being sampled uniformly and independently *after* the adversary submits the circuit C'. Let $C'(x) = (y', v')$. Then, the probability that $y' = y$ is exactly $1/2^m = \mathrm{negl}(\lambda)$, where the probability is taken over the distribution of y. □

Combining Lemmas I.8 through I.12, we conclude that Π_{wm} is δ-unforgeable. □