Constraining Pseudorandom Functions Privately[*]

Dan Boneh, Kevin Lewi, and David J. Wu

Stanford University

Abstract. In a constrained pseudorandom function (PRF), the master secret key can be used to derive constrained keys, where each constrained key k is constrained with respect to some Boolean circuit C. A constrained key k can be used to evaluate the PRF on all inputs x for which $C(x) = 1$. In almost all existing constrained PRF constructions, the constrained key k reveals its constraint C.

In this paper we introduce the concept of *private* constrained PRFs, which are constrained PRFs with the additional property that a constrained key does not reveal its constraint. Our main notion of privacy captures the intuition that an adversary, given a constrained key k for one of two circuits C_0 and C_1, is unable to tell which circuit is associated with the key k. We show that constrained PRFs have natural applications to searchable symmetric encryption, cryptographic watermarking, and much more.

To construct private constrained PRFs we first demonstrate that our strongest notions of privacy and functionality can be achieved using indistinguishability obfuscation. Then, for our main constructions, we build private constrained PRFs for bit-fixing constraints and for puncturing constraints from concrete algebraic assumptions.

1 Introduction

A pseudorandom function (PRF) [41] 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 [12], Boyle, Goldwasser, and Ivan [16], and Kiayias, Papadopoulos, Triandopoulos and Zacharias [47], 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,

[*]The full version of this paper is available at `http://eprint.iacr.org/2015/1167.pdf`.
[1]They have also been called *functional* PRFs [16] and *delegatable* PRFs [47]

for example, in broadcast encryption [12] and in the "punctured programming" techniques of Sahai and Waters [54].

The Goldreich-Goldwasser-Micali (GGM) PRF [41] is a *puncturable* PRF, that is, a constrained PRF for the special class of puncturing constraints. In a puncturable PRF, each constrained key k is associated with an input $x_0 \in \mathcal{X}$, and the constrained key enables the evaluation at all points $x \neq x_0$ while revealing no information about $F(\mathsf{msk}, x_0)$. It is not difficult to see that the constrained key k completely reveals the point x_0.

Boneh and Waters [12] show how to use multilinear maps [33, 28, 36, 29] to construct constrained PRFs 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 [43, 44], developing schemes with additional properties such as verifiability [19], and constructing (single-key) circuit-constrained PRFs from standard lattice-based assumptions [17].

Constraining privately. In this work, we initiate the study of *private* constrained PRFs, which are 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. We formally define our privacy notion in Section 2.

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 [41], 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,

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

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.

1.1 Applications of Private Constrained PRFs

To illustrate the power of private constrained PRFs we first describe a few natural applications, including private constrained MACs, watermarkable PRFs, and searchable encryption. In Section 6.2, we also describe an application to symmetric deniable encryption.

Private constrained MACs. Constrained MACs are the secret-key variant of constrained signatures, which were first introduced by Boyle et al. [16]. 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 verifier, who is issued a constrained key[3] from the offline key distributor, is able to verify Alice's requests without knowing or learning anything about her spending limits.

Watermarking PRFs. A watermarking scheme for programs [45, 5, 25, 51, 24] 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 [45, 5, 24], 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 [24].

Hopper, Molnar, and Wagner [45] 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. [5]. In a secretly-verifiable scheme, only the holder of a secret key can test if a program is watermarked. More recently, Cohen et al. [24] 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 6.1.

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 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 and a concrete construction based on indistinguishability obfuscation in the full version of this paper [10].

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 F, 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

[3]The verifier's constrained key is chosen so that the constraint is always satisfied. Note that this is not the same as giving out the master verification key, which may allow the verifier to learn Alice's spending limits.

x, it can trivially remove the watermark by producing a circuit that simply changes the value at x. We show in Section 6.1 that this simple construction not only satisfies our notion of secretly-verifiable watermarking, but can also be easily extended to support embedding arbitrary messages as the watermark.

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) [55, 40, 30, 20, 6], 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 [23, 32, 53], a server holds a database $D = \{w_1, \dots, w_n\}$ 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 [39, 15], 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, Boyle, Gilboa and Ishai [39, 15] 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 $s_{i^*} \neq t_{i^*}$ with high probability, and so $z \neq 0^m$.

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 [53] 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 [38] 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

We formally define our notion of privacy in Section 2. In this section, we briefly outline our constructions of private constrained PRFs. As a warmup, we begin with a construction from indistinguishability obfuscation, and then we give an overview of our two constructions from concrete assumptions on multilinear maps for bit-fixing constraints and puncturing constraints.

A construction from indistinguishability obfuscation. Indistinguishability obfuscation (iO) [5, 34, 4, 54, 37, 56, 3] is a powerful primitive that has enabled a number of new constructions in cryptography [54, 14, 34]. 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, we treat it primarily as a proof-of-feasibility for private constrained PRFs. For our two main constructions, we build private constrained PRFs for more restrictive classes of constraints based on concrete assumptions over multilinear maps.

Multilinear maps [11, 33, 28, 36, 29] have been successfully applied to many problems in cryptography, most notably in constructing indistinguishability obfuscation [34, 4, 2, 37, 56, 3]. Unfortunately, a number of recent attacks [22, 13, 26, 27, 46, 21] have invalidated many of the basic assumptions on multilinear maps. However, indistinguishability obfuscation is an example of a setting where the adversary often does not have the necessary information to carry out these attacks, and so some of the existing constructions are not known to be broken [35, 31]. In our first construction from multilinear maps, we rely on the Multilinear Diffie-Hellman (MDH) assumption [11, 33] over prime-order multilinear maps. In our second construction, we rely on the Subgroup Decision assumption [9, 33] as well as a generalization which we call the Multilinear Diffie-Hellman Subgroup Decision (MDHSD) assumption over composite-order multilinear maps.[4] Our assumptions plausibly hold in existing multilinear map candidates, notably the Garg et al. construction in the prime-order setting [33], and the Coron et al. construction for the composite-order setting [28]. We also note that starting from iO, it is also possible to construct multilinear maps where the MDH assumption holds [1].

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 [12] based on prime-order multilinear maps [11, 33]. 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

Kiayias et al. [47] introduced a notion of policy privacy for delegatable PRFs. In a delegatable PRF, a proxy can evaluate the 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 efficient than our PRF constructions.

As discussed earlier, Boyle et al. [16] introduced 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

[4] In the full version [10], we show this assumption holds in a generic multilinear map model.

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 Section 6.1, we also survey the related work on cryptographic watermarking.

Private puncturing and distributed point functions. Recently, Boyle, Gilboa and Ishai introduced the notion of a distributed point function (DPF) [39, 15], which are closely related to private puncturable PRFs. In a DPF, there are two functions Gen and Eval. The function Gen takes as input a pair $x, y \in \{0,1\}^*$ and outputs two keys k_0 and k_1, and Eval is defined 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 individually appear to be distributed independently of x and y. A DPF is similar to a private puncturable PRF in that we can view k_0 as the master secret key for a PRF and k_1 as a constrained 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 $\mathsf{Eval}(k_0, \cdot)$ and $\mathsf{Eval}(k_1, \cdot)$ need not be pseudorandom),[5] 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. We note though that a private puncturable PRF can be used directly to construct a DPF: we simply let k_0 be the master secret key of the PRF and k_1 be a key punctured at x.

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 $x \xleftarrow{\mathrm{R}} S$ 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 $f \xleftarrow{\mathrm{R}} \mathsf{Funs}(S, T)$, 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

[5]Though this property is not explicitly required by a DPF, in existing constructions [39, 15], the functions $\mathsf{Eval}(k_0, \cdot)$ and $\mathsf{Eval}(k_1, \cdot)$ are individually pseudorandom.

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) [41]. 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 [41]). *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 $\mathcal{A}$,*

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

We also review the definition of a constrained PRF [12, 47, 16]. 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 [12, 47, 16]). *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. We now 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 [12]. This 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 $f \xleftarrow{\text{R}} \mathsf{Funs}(\{0,1\}^n, \{0,1\}^m)$ 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 $\Pr[\mathsf{Expt}_b^{\mathsf{cPRF}}(\mathcal{A}) = 1]$ 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:*

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

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 [12], 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 [7] (at the expense of a super-polynomial loss in the security reduction).

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 $\Pr[\mathsf{Expt}_b^{\mathsf{cpriv}}(\mathcal{A}) = 1]$ 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 $i, j \in [d]$,*

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

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:*

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

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. To see this, 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 $C_1^{(1)}(x) = 1$ 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 implies that $z_2 \neq z_1$ 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 can be a weaker notion of privacy (for instance, in the case of d-key privacy for bounded d), in all of our candidate applications, a scheme that satisfies this weaker notion suffices.

Puncturable PRFs. A *puncturable* PRF [54, 47, 12, 16] 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.

3 Private Circuit Constrained PRFs from Obfuscation

In this section, we show how multi-key private circuit-constrained PRFs follow straightforwardly from indistinguishability obfuscation and puncturable PRFs (implied by one-way functions [41, 12, 47, 16]). First, we review the notion of indistinguishability obfuscation introduced by Barak et al. [5].

Definition 3.1 (Indistinguishability Obfuscation (iO) [5, 34]). *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 the distinguishing advantage* $\mathsf{Adv}_{\mathsf{iO},\mathcal{A}}(\lambda)$ *is negligible:*

$$\mathsf{Adv}_{\mathsf{iO},\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}(\mathsf{iO}(C_0)) = 1] - \Pr[\mathcal{A}(\mathsf{iO}(C_1)) = 1]| = \operatorname{negl}(\lambda).$$

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 We provide a more detailed discussion of this in the full version [10]. Also in the full version, we define the notion of a private *programmable* PRF and show how to adapt our private circuit-constrained PRF to also obtain a private programmable PRF from (polynomially-hard) iO and one-way functions.

Construction overview. Our starting point is the circuit-constrained PRF by Boneh and Zhandry [14, 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. We now describe our construction of a multi-key private circuit-constrained PRF. 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 $\mathsf{iO}\left(P_1\left[C, \mathsf{msk}', \mathsf{msk}\right]\right)$, where $P_1\left[C, \mathsf{msk}', \mathsf{msk}\right]$ is the following program:[6]

```
Constants: a circuit $C : \{0,1\}^n \to \{0,1\}$, and master secret keys
$\mathsf{msk}_0$, $\mathsf{msk}_1$ for the puncturable PRF $\Pi_{\mathsf{F}} = (\mathsf{F.Setup}, \mathsf{F.Puncture},$
$\mathsf{F.ConstrainEval}, \mathsf{F.Eval})$.

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

  1. Let $b = C(x)$. Output $\mathsf{F.Eval}(\mathsf{msk}_b, x)$.
```

Fig. 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 the full version [10].

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).*

We note that Theorem 3.3 only requires subexponentially-secure[7] iO if the set of challenge circuits $\{C_0^{(j)}\}_{j\in[d]}$ and $\{C_1^{(j)}\}_{j\in[d]}$ the adversary submits differs 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}. We discuss this in greater detail in the full version [10].

[6]We pad the program $P_1\left[C, \mathsf{msk}', \mathsf{msk}\right]$ to the maximum size of any program that appears in the hybrid experiments in the proofs of Theorem 3.2 and 3.3.

[7]Specifically, we require that for all efficient adversaries $\mathcal{A}$, the distinguishing advantage $\mathsf{Adv}_{\mathsf{iO},\mathcal{A}}(\lambda)$ defined in Definition 3.1 satisfies $2^n \cdot \mathsf{Adv}_{\mathsf{iO},\mathcal{A}}(\lambda) = \text{negl}(\lambda)$.

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 [12]. 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 [12]). *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 [11], which are a generalization of bilinear maps [48, 49, 8]. While constructing ideal multilinear maps remains an open problem, there have been several recent candidates of graded encodings schemes [33, 28, 36, 29], which are 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 [12, Appendix B]. We begin by defining multilinear maps over prime-order groups. In the full version [10], we also recall the ℓ-Multilinear Diffie-Hellman assumption [11, 33] over prime-order multilinear maps.

Definition 4.2 (Prime-Order Multilinear Map [11, 33, 28, 36, 29]). *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 [12]. 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 $i + j \le n + 1$. 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 [12], 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 $d^*_{i,b} \xleftarrow{\text{R}} \mathbb{Z}_p$. 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 $g_i \in \mathbb{G}_i$ for all $i \in [n+1]$. As usual, we set $g = g_1$. Next, for $i \in [n]$, it samples $(d_{i,0}, d_{i,1}) \xleftarrow{\text{R}} \mathbb{Z}_p^2$, along with a random $\alpha \xleftarrow{\text{R}} \mathbb{Z}_p$. 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 $\beta_1 \ldots, \beta_n \xleftarrow{\text{R}} \mathbb{Z}_p$ uniformly and independently, along with n random elements $r_1, \ldots, r_n \xleftarrow{\text{R}} \mathbb{Z}_p$. 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 the full version [10].

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

Theorem 4.4. *Under the $(n+1)$-MDH assumption, 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 the full version [10], we review the standard Subgroup Decision assumption [9, 33] over composite-order groups, and a new assumption which we call the ℓ-Multilinear Diffie-Hellman Subgroup Decision (MDHSD) assumption. Also in the full version, 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 [11, 28, 29]). *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 $\mathbb{G}_{p,i}$, $g_{q,i}$ be a canonical generator of $\mathbb{G}_{q,i}$, and $g_i = g_{p,i} g_{q,i}$. 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 [50], 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 $(d_{1,0}, d_{1,1}), \ldots, (d_{n,0}, d_{n,1}) \xleftarrow{\text{R}} \mathbb{Z}_N^2$, and outputs the following master secret key msk:
$$\mathsf{msk} = \Big(p, q, g_1, g_{p,1}, g_{p,n}, \{d_{i,0}, d_{i,1}\}_{i\in[n]}\Big) \tag{5.1}$$
- $\mathsf{cPRF.Puncture}(\mathsf{msk}, s \in \{0,1\}^n)$. 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 the full version [10].

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

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

Theorem 5.4. *Under the Subgroup Decision assumption, 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 using private constrained PRFs to build secretly-verifiable message-embedding watermarking of PRFs and symmetric deniable encryption.

6.1 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 [24], 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 [24] while the unforgeability definitions are adapted from that in [25]. We then show how to construct a watermarkable PRF from any private programmable PRF. Finally, we conclude with a survey of related work.

Definition 6.1 (Watermarkable Family of PRFs [24, adapted]). *For the security parameter λ and a message space $\{0,1\}^t$, a secretly-verifiable message-embedding watermarking scheme for 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.*

[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 the full version [10].

- $\mathsf{WM.Setup}(1^\lambda) \to \mathsf{msk}$. *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 $m \in \{0,1\}^t$, 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 $m \in \{0,1\}^t \cup \{\bot\}$.*

Definition 6.2 (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*

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

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 6.3 (Correctness ([24, 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 $m \in \{0,1\}^t$, $\mathsf{msk} \leftarrow \mathsf{WM.Setup}(1^\lambda)$, $(k, C) \leftarrow \mathsf{WM.Mark}(\mathsf{msk}, m)$, 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 $1/f(n) = \text{negl}(\lambda)$.*
- *$\Pr[\mathsf{WM.Verify}(\mathsf{msk}, C) = m]$ 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 6.4 (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 $m \in \{0,1\}^t$, the challenger returns the pair $(k, C) \leftarrow \mathsf{WM.Mark}(\mathsf{msk}, m)$ to $\mathcal{A}$.*
- ***Challenge oracle.*** *On input a message $m \in \{0,1\}^t$, 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 6.5 (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 $1/f(n) = \text{negl}(\lambda)$.*

Definition 6.6 (Unremovability). *A watermarking scheme $\mathit{\Pi}$ is* **unremovable** *if for all efficient and unremoving admissible adversaries $\mathcal{A}$, if $m \in \{0,1\}^t$ is the message submitted by $\mathcal{A}$ to the challenge oracle in $\mathsf{Expt}_{\mathsf{wm}}$, the probability $\Pr[\mathsf{Expt}_{\mathsf{wm}}(\mathcal{A}) \neq m]$ is negligible.*

Definition 6.7 ($\boldsymbol{\delta}$-Unforging Admissibility). *We say an adversary $\mathcal{A}$ is* **$\boldsymbol{\delta}$-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 6.8 ($\boldsymbol{\delta}$-Unforgeability). *We say a watermarking scheme $\mathit{\Pi}$ is* **$\boldsymbol{\delta}$-unforgeable** *if for all efficient and δ-unforging admissible adversaries $\mathcal{A}$, the probability $\Pr[\mathsf{Expt}_{\mathsf{wm}}(\mathcal{A}) \neq \bot]$ is negligible.*

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

– $\mathsf{WM.Setup}(1^\lambda)$. The setup algorithm chooses $\mathsf{k} \xleftarrow{\text{R}} \mathcal{K}$ and $(z_1, \ldots, z_d) \xleftarrow{\text{R}} (\{0,1\}^n)^d$ 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 the point $(x, y, \tau) = F(\mathsf{k}, (\mathsf{pPRF.Eval}(k', z_1), \ldots, \mathsf{pPRF.Eval}(k', z_d)))$ and $v = m \oplus \tau$. Then, it computes $\mathsf{sk}_k \leftarrow \mathsf{pPRF.Program}(k', x, (y, v))$ and outputs (k', C), where $C(\cdot) = \mathsf{pPRF.ProgramEval}(\mathsf{sk}_k, \cdot)$.
– $\mathsf{WM.Verify}(\mathsf{msk}, C)$. 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 state our correctness and security theorems here, but defer their proofs to the full version [10].

Theorem 6.9. *If F is a secure PRF and $\mathit{\Pi}_{\mathsf{pprf}}$ is a programmable PRF, then the watermarking scheme $\mathit{\Pi}_{\mathsf{wm}}$ is correct.*

Theorem 6.10. *If F is a secure PRF and $\mathit{\Pi}_{\mathsf{pprf}}$ is a private programmable PRF, then the watermarking scheme $\mathit{\Pi}_{\mathsf{wm}}$ is unremovable.*

Theorem 6.11. *If F is a secure PRF and $\mathit{\Pi}_{\mathsf{pprf}}$ is a programmable PRF, then for $\delta = 1/\text{poly}(\lambda)$, the watermarking scheme $\mathit{\Pi}_{\mathsf{wm}}$ is δ-unforgeable.*

Related work. Recently, Cohen et al. [24] showed how to construct publicly-verifiable watermarking for puncturable PRFs from indistinguishability obfuscation. They pursue the notion of approximate functionality-preserving for watermarking, where the watermarked program agrees with the original program on *most* inputs. Previously, Barak et al. [5] showed that assuming iO, perfectly functionality-preserving watermarking is impossible.

Cohen et al. [25] 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 [25], Nishimaki and Wichs [51] 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 [24].

Comparison to previous works. In previous constructions of watermarkable PRFs [51, 25, 24], 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 Cohen et al. [24]. In our model, we are able to achieve full security for unremovability as well as strong unforgeability.

6.2 Symmetric Deniable Encryption

The notion of deniable encryption was first introduced by Canetti, Dwork, Naor, and Ostrovsky [18]. 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.

[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).

Recently, Sahai and Waters [54] used indistinguishability obfuscation [5, 34, 4, 54, 37, 56] to give the first construction of public-key deniable encryption that achieves the security notions put forth by Canetti et al.[10] 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 [18].

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 encryption key, and yet, will only correctly decrypt all but one of the intercepted ciphertexts.[11] 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 [18, 52] 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).

In this section, we give a formal definition of symmetric deniable encryption adapted from those of Canetti et al. [18]. 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 6.12 (Symmetric Deniable Encryption [18, 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* $\mathsf{sk}' \in \mathcal{K}$.

[10] 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.

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

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 [42].

Definition 6.13 (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*

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

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

Definition 6.14 (Semantic Security [42, 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. [18, 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 6.15 (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 $m^* \xleftarrow{\mathrm{R}} \mathcal{M}$, 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 6.16. *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 $x \xleftarrow{\text{R}} \{0,1\}^n$ 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 $r \xleftarrow{\text{R}} \{0,1\}^n$ 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 state our correctness and security theorems here, but defer their proofs to the full version [10].

Theorem 6.17. *The deniable encryption scheme Π_{DE} is correct.*

Theorem 6.18. *If Π_{cprf} is a secure PRF, then Π_{DE} is semantically secure.*

Theorem 6.19. *If Π_{cprf} is a 1-key private, selectively-secure PRF, then Π_{DE} is deniable (Definition 6.16).*

Related work. 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 [52] 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.

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 to generate fake 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.

7 Conclusions

In this work, we introduce the notion of privacy for constrained PRFs, and give a number of interesting applications including 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.

References

[1] Martin R. Albrecht, Pooya Farshim, Dennis Hofheinz, Enrique Larraia, and Kenneth G. Paterson. Multilinear maps from obfuscation. In *TCC*, pages 446–473, 2016.
[2] Prabhanjan Vijendra Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. In *ACM CCS*, pages 646–658, 2014.
[3] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *TCC*, pages 528–556, 2015.
[4] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, 2014.
[5] 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.
[6] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, pages 224–241, 2009.
[7] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.
[8] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, pages 213–229, 2001.
[9] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, pages 325–341, 2005.
[10] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. *IACR Cryptology ePrint Archive*, 2015:1167, 2015.
[11] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003.
[12] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, pages 280–300. Springer, 2013.
[13] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. *IACR Cryptology ePrint Archive*, 2014:930, 2014.
[14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, pages 480–499, 2014.
[15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT*, pages 337–367, 2015.

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

[17] 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.

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

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

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

[21] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new CLT multilinear map over the integers. In *EUROCRYPT*, pages 509–536, 2016.

[22] 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.

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

[24] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *STOC*, pages 1115–1127, 2016.

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

[26] 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.

[27] Jean-Sébastien Coron, Moon Sung Lee, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. *IACR Cryptology ePrint Archive*, 2015:1037, 2015.

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

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

[30] 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.

[31] Rex Fernando, Peter M. R. Rasmussen, and Amit Sahai. Preventing CLT zeroizing attacks on obfuscation. *IACR Cryptology ePrint Archive*, 2016:1070, 2016.

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

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

[34] 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.

[35] Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In *TCC*, pages 241–268, 2016.

[36] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*, pages 498–527, 2015.

[37] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In *FOCS*, pages 151–170, 2015.

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

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

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

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

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

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

[44] Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In *ASIACRYPT*, pages 79–102, 2015.

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

[46] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. In *EUROCRYPT*, pages 537–565, 2016.

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

[48] 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.

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

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

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

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

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

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

[55] 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.

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