Watermarking PRFs from Lattices: Stronger Security via Extractable PRFs*

Sam Kim[1] and David J. Wu[2†]

[1] Stanford University, Stanford, CA
[2] University of Virginia, Charlottesville, VA

Abstract. A software watermarking scheme enables one to embed a "mark" (i.e., a message) within a program while preserving the program's functionality. Moreover, there is an extraction algorithm that recovers an embedded message from a program. The main security goal is that it should be difficult to remove the watermark without destroying the functionality of the program. Existing constructions of watermarking focus on watermarking cryptographic functions like pseudorandom functions (PRFs); even in this setting, realizing watermarking from standard assumptions remains difficult. The first lattice-based construction of secret-key watermarking due to Kim and Wu (CRYPTO 2017) only ensures mark-unremovability against an adversary who does not have access to the mark-extraction oracle. The construction of Quach et al. (TCC 2018) achieves the stronger notion of mark-unremovability even if the adversary can make extraction queries, but has the drawback that the watermarking authority (who holds the watermarking secret key) can break pseudorandomness of all PRF keys in the family (including *unmarked* keys).

In this work, we construct new lattice-based secret-key watermarking schemes for PRFs that both provide unremovability against adversaries that have access to the mark-extraction oracle and offer a strong and meaningful notion of pseudorandomness even against the watermarking authority (i.e., the outputs of unmarked keys are pseudorandom almost everywhere). Moreover, security of several of our schemes can be based on the hardness of computing *nearly polynomial* approximations to worst-case lattice problems. This is a qualitatively weaker assumption than that needed for existing lattice-based constructions of watermarking (that support message-embedding), all of which require quasi-polynomial approximation factors. Our constructions rely on a new cryptographic primitive called an *extractable PRF*, which may be of independent interest.

1 Introduction

A software watermarking scheme enables a user or an authority to embed a "mark" within a program in a way that the marked program behaves almost identically

*The full version of this paper is available at `https://eprint.iacr.org/2018/986.pdf`.
†Part of this work was done while a student at Stanford University.

to the original program. It should be difficult to remove the watermark from a marked program without significantly altering the program's behavior, and moreover, it should be difficult to create new (or malformed) programs that are considered to be watermarked. The first property of *unremovability* is useful for proving ownership of software (e.g., in applications to digital rights management) while the second property of *unforgeability* is useful for authenticating software (e.g., for proving that the software comes from a trusted distributor).

1.1 Background and Motivation

Barak et al. [9, 10] and Hopper et al. [35] introduced the first rigorous mathematical framework for software watermarking. Realizing the strong security requirements put forth in these works has been difficult. Early works [42, 51, 43] made partial progress by considering weaker security models and imposing restrictions on the adversary's capabilities. This changed with the work of Cohen et al. [27], who gave the first positive construction of software watermarking (for classes of cryptographic functionalities) that achieved unremovability against *arbitrary* adversarial strategies from indistinguishability obfuscation.

More formally, a software watermarking scheme consists of two main algorithms. First, the marking algorithm takes a circuit C and outputs a "marked" circuit C' with the property that C' and C agree almost everywhere. Second, a verification algorithm takes a circuit C and outputs MARKED or UNMARKED. In the message-embedding setting, the marking algorithm also takes a message m in addition to the circuit and embeds the message m within the circuit as the watermark. In this case, we replace the verification algorithm with a mark-extraction algorithm that takes a circuit as input and which outputs either the embedded message or UNMARKED. A watermarking scheme is robust against *arbitrary* removal strategies if the adversary is given complete flexibility in crafting a circuit $\tilde{C}'$ that mimics the behavior of a marked circuit C', but does not contain the watermark. This most directly captures our intuitive notions of unremovability and is the setting that we focus on in this work.

Since the work of Cohen et al., there has been many works on building stronger variants of software watermarking [49, 50] and constructing watermarking (and variants) from simpler assumptions [16, 38, 5, 45]. While this latter line of work has made tremendous progress and has yielded constructions of watermarking from standard lattice assumptions [38], CCA-secure encryption [45], and even public-key encryption (in the stateful setting) [5], these gains have come at the price of relaxing the watermarking security requirements. As such, there is still a significant gap between the security and capabilities of the Cohen et al. construction [27] from indistinguishability obfuscation and the best schemes we have from standard assumptions. In this work, we narrow this gap and introduce a new lattice-based software watermarking scheme for pseudorandom functions (PRFs) that satisfies stronger security and provides more functionality than the previous constructions from standard assumptions.

Watermarking PRFs. While the notion of software watermarking is well-defined for general functionalities, Cohen et al. [27] showed that watermarking

is impossible for any class of learnable functions. Consequently, research on watermarking has focused on cryptographic functions like PRFs. In their work, Cohen et al. gave the first constructions of watermarking for PRFs (as well as several public-key primitives) from indistinguishability obfuscation. The Cohen et al. watermarking construction has the appealing property in that the scheme supports *public mark-extraction* (i.e., anyone is able to extract the embedded message from a watermarked program). The main drawback though is their reliance on strong (and non-standard) assumptions. Subsequently, Boneh et al. [16] introduced the concept of a private puncturable PRF and showed how to construct *secretly-extractable* watermarking schemes from a variant of private puncturable PRFs (called private programmable PRFs). Building on the Boneh et al. framework, Kim and Wu [38] showed that a relaxation of private programmable PRFs also sufficed for watermarking, and they gave the first construction of watermarking from standard lattice assumptions. Neither of these constructions support public extraction, and constructing watermarking schemes that support public extraction from standard assumptions remains a major open problem.

Towards publicly-extractable watermarking. Not only did the schemes in [16, 38] not support public extraction, they had the additional drawback that an adversary who only has access to the extraction oracle for the watermarking scheme can easily remove the watermark from a marked program (using the algorithm from [27, §2.4]). Thus, it is unclear whether these schemes bring us any closer to a watermarking scheme with public extraction. A stepping stone towards a publicly-extractable watermarking scheme is to construct a secretly-extractable watermarking scheme, except we give the adversary access to the extraction oracle. The difficulty in handling extraction queries is due to the "verifier rejection" problem that also arises in similar settings such as constructing designated-verifier proof systems or CCA-secure encryption. Namely, the adversary can submit carefully-crafted circuits to the extraction oracle and based on the oracle's responses, learn information about the secret watermarking key.

Recently, Quach et al. [45] gave an elegant and conceptually-simple construction of secretly-extractable watermarking that provided unremovability in this stronger model where the adversary has access to the extraction oracle. Moreover, their scheme also supports *public marking*: namely, anyone is able to take a PRF and embed a watermark within it. The basic version of their scheme is mark-embedding (i.e., programs are either marked or unmarked) and can be instantiated from any CCA-secure public-key encryption scheme. To support full message-embedding, their construction additionally requires private puncturable PRFs (and thus, the only standard-model instantiation today relies on lattices). In both cases, however, their scheme has the drawback in that the holder of the watermarking secret key completely compromises pseudorandomness of all PRF keys in the family (including *unmarked* keys). In particular, given even two evaluations of a PRF (on distinct points), the watermarking authority in the scheme of [45] can already distinguish the evaluations from random. While it might be reasonable to trust the watermarking authority, we note here that

users must *fully* trust the authority (even if they generate a PRF key only for themselves and never interact with the watermarking authority). Even if the authority *passively* observes PRF evaluations (generated by honest users), it is able to tell those evaluations apart from truly random values. As we discuss below, this is a significant drawback of their construction and limits its applicability. Previous constructions [27, 16, 38] did not have this drawback.

Security against the watermarking authority. Intuitively, it might seem like in any secret-key watermarking scheme, users implicitly have to trust the watermarking authority (either to mark their keys, or to verify their keys, or both), and so, there is no reason to require security against the watermarking authority. However, we note that this is not the case. For example, the marking and extraction algorithms can always be implemented by a two-party computation between the watermarking authority and the user, in which case the watermarking authority *never* sees any of the users' keys in the clear, and yet, the users still enjoy all of the protections of a watermarking scheme. In existing schemes that do not provide security against the watermarking authority [45], the PRF essentially has a "backdoor" and the watermarking authority is able to distinguish *every* evaluation or *every* PRF in the family from random. This is a significant increase in the amount of trust the user now has to place in the watermarking authority. The constructions we provide in this work provide a meaningful notion of security even against the watermarking authority. Namely, as long as the users never evaluate the PRF on a restricted set of points (which is a sparse subset of the domain and statistically hidden from the users), then the input/output behavior of both unmarked and marked keys remain pseudorandom even against the watermarking authority.

More generally, as noted above, a watermarking scheme that supports extraction queries is an intermediate primitive between secretly-extractable watermarking and publicly-extractable watermarking. If the intermediate scheme is insecure in the presence of a party who can extract, then the techniques used in that scheme are unlikely to extend to the public-key setting (where *everyone* can extract). Handling extraction queries (with security against the authority) is closer to publicly-extractable watermarking compared to notions from past works. We believe our techniques bring us closer towards publicly-extractable watermarking from standard assumptions.

1.2 Our Contributions

In this work and similar to [45], we study secretly-verifiable watermarking schemes for PRFs that provide unremovability (and unforgeability) against adversaries that have access to both the marking and the extraction oracles. Our goal is to achieve these security requirements while maintaining security even against the watermarking authority. We provide several new constructions of secretly-verifiable watermarking schemes for PRFs from standard lattice assumptions where the adversary has access to the extraction oracle. Moreover, we show that all of our constructions achieve a relaxed (but still meaningful) notion of

Scheme	Public Marking	Public Extraction	Extraction Oracle	PRF Security (Authority)	Hardness Assumption
Cohen et al. [27]	✗	✓	✓	✓	iO
Boneh et al. [16]	✗	✗	✗	✓	iO
Kim-Wu [38]	✗	✗	✗	✓	LWE*
Quach et al. [45]	✓	✗	✓	✗	LWE*
Yang et al. [50]	✗	✓	✓	✓	iO
This Work	✗	✗	✓	✓†	LWE‡
	✓	✗	✓	✓†	LWE‡ + RO

*LWE with a quasi-polynomial modulus-to-noise ratio (i.e., $2^{\log^c n}$ for constant $c > 1$).
†Our construction provides a weaker notion of *restricted* pseudorandomness against the watermarking authority.
‡LWE with a nearly polynomial modulus-to-noise ratio (i.e., $n^{\omega(1)}$).

Table 1: Comparison of our watermarkable family of PRFs to previous constructions. We focus exclusively on *message-embedding* constructions. For each scheme, we indicate whether it supports public marking and public extraction, whether mark-unremovability holds in the presence of an extraction oracle, whether unmarked keys remain pseudorandom against the watermarking authority, and the hardness assumption each scheme is based on. In the above, "iO" denotes indistinguishability obfuscation and "RO" denotes a random oracle.

pseudorandomness for unmarked keys even in the presence of the watermarking authority. Our constructions also simultaneously achieve unremovability and unforgeability (with parameters that match the lower bounds in Cohen et al. [27]). In fact, we show that meaningful notions of unforgeability (that capture the spirit of unforgeability and software authentication as discussed in [35, 27, 50]) are even possible for schemes that support public marking. Our constructions are the first to provide all of these features. Moreover, we are able to realize these new features while relying on *qualitatively weaker* lattice-based assumptions compared to all previous watermarking constructions from standard assumptions (specifically, on the hardness of computing *nearly polynomial* (i.e., $n^{\omega(1)}$) approximations to worst-case lattice problems as opposed to computing *quasi-polynomial* (i.e., $2^{\log^c(n)}$ for constant $c > 1$) approximations; see Remark 4.13). We provide a comparison of our new watermarking construction to previous schemes in Table 1, and also summarize these results below.

Extractable PRFs. The key cryptographic building block we introduce in this work is the notion of an *extractable PRF*. An extractable PRF is a standard PRF family $\mathsf{F}\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ outfitted with an extraction trapdoor td. The extractability property says that given any circuit that computes $\mathsf{F}(k, \cdot)$, the holder of the trapdoor td can recover the PRF key k (with overwhelming probability). In fact, the extraction process is *robust* in the following sense: given any circuit

$C\colon \mathcal{X} \to \mathcal{Y}$ whose behavior is "close" to $\mathsf{F}(k, \cdot)$, the extraction algorithm still extracts the PRF key k. The notion of closeness that we use is ε-closeness: we say that two circuits C_0 and C_1 are ε-close if C_0 and C_1 only differ on at most an ε-fraction of the domain. Of course, for extraction to be well-defined, it must be the case that for any pair of distinct keys k_1, k_2, the functions $\mathsf{F}(k_1, \cdot)$ and $\mathsf{F}(k_2, \cdot)$ are far apart. We capture this by imposing a statistical requirement on the PRF family called key-injectivity,[3] which requires that $\mathsf{F}(k_1, \cdot)$ and $\mathsf{F}(k_2, \cdot)$ differ on at least an ε'-fraction of points where $\varepsilon' \gg \varepsilon$. This ensures that if C is ε-close to some PRF $\mathsf{F}(k, \cdot)$, then k is unique (and extraction recovers k). In Section 2, we provide a detailed technical overview on how to construct extractable PRFs from standard lattice assumptions. We give the formal definition, construction, and security analysis in Section 4.

From extractable PRFs to watermarking. The combination of extractability and key-injectivity gives a natural path for constructing a secret-key watermarking scheme for PRFs. We begin with a high-level description of our basic mark-embedding construction which illustrates the main principles. First, we will need to extend our extractable PRF family to additionally support puncturing. In a puncturable PRF [18, 37, 19], the holder of a PRF key k can puncture k at a point x^* to derive a "punctured key" k_{x^*} with the property that k_{x^*} can be used to evaluate the PRF on all points $x \neq x^*$. Moreover, given the punctured key k_{x^*}, the value of the PRF $\mathsf{F}(k, x^*)$ at x^* is still indistinguishable from a uniformly random value.

Suppose now that we have an extractable PRF where the PRF keys can be punctured. To construct a mark-embedding watermarkable family of PRFs $\mathsf{F}\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, we take the watermarking secret key to be the trapdoor for the extractable PRF family. To mark a PRF key $k \in \mathcal{K}$, the watermarking authority derives a special point $x^{(k)} \in \mathcal{X}$ from k (using a PRF key that is also part of the watermarking secret key), and punctures k at $x^{(k)}$ to obtain the punctured key $k_{x^{(k)}}$. The watermarked program just implements PRF evaluation using the punctured key $k_{x^{(k)}}$. To check whether a circuit $C\colon \mathcal{X} \to \mathcal{Y}$ is marked, the watermarking authority applies the extraction algorithm to C to obtain a key $k \in \mathcal{K}$ (or $\perp$ if extraction does not output a key). If the extraction algorithm outputs a key $k \in \mathcal{K}$, the verification algorithm computes the special point $x^{(k)}$ from k and outputs MARKED if $C(x^{(k)}) \neq \mathsf{F}(k, x^{(k)})$ and UNMARKED otherwise. If the extraction algorithm outputs $\perp$, the algorithm outputs UNMARKED.

Unremovability of this construction essentially reduces to puncturing security. By robust extractability (and key-injectivity), if the adversary only corrupts a small number of points in a marked key (within the unremovability threshold), then the extraction algorithm successfully recovers k (with overwhelming probability). To remove the watermark, the adversary's task is to "fix" the value of the PRF at the punctured point $x^{(k)}$. Any adversary that succeeds to do so breaks puncturing security (in particular, the adversary must be able to recover the real value of the PRF at the punctured point given only the punctured key).

[3] Key-injectivity also played a role in previous watermarking constructions, though in a different context [27, 38].

Note that here, we do require that the range $\mathcal{Y}$ of the PRF be super-polynomial (if the range was polynomial, then the adversary can guess the correct value of the PRF at $x^{(k)}$ with noticeable probability and remove the watermark). Note that this basic scheme neither provides unforgeability (i.e., it is easy to construct circuits that are considered MARKED even without the watermarking key) nor supports message-embedding. As we discuss in greater detail below, both of these properties can be achieved with additional work.

Handling extraction queries. A primary objective of this work is to construct a watermarking scheme for PRFs where unremovability holds even against an adversary that has access to the extraction oracle. At first glance, our marking algorithm may appear very similar to that in [16, 38], since all of these constructions rely on some form of puncturable PRFs. These previous constructions do not satisfy unremovability in the presence of an extraction oracle because they critically rely on the adversary not being able to identify the special point $x^{(k)}$. Namely, in these constructions, to check whether a circuit C is marked or not, the authority derives the special point $x^{(k)}$ from the input/output behavior of C and then checks whether $C(x^{(k)})$ has a *specific* structure. If the adversary is able to learn the point $x^{(k)}$, then it can tweak the value of the marked circuit at $x^{(k)}$ and remove the watermark. In fact, even if the puncturable PRF completely hides the special point $x^{(k)}$, the binary search attack from Cohen et al. [27] allows the adversary to use the extraction oracle to recover $x^{(k)}$, and thus, defeat the watermarking scheme.

In our construction, to decide whether a circuit C is marked or not, the authority first extracts a key k and checks whether $C(x^{(k)}) = \mathsf{F}(k, x^{(k)})$. Therefore, in order to remove the watermark, it is not enough for the adversary to just recover the special point $x^{(k)}$ via the extraction oracle (in fact, the special point $x^{(k)}$ is public). To succeed, the adversary has to recover the original *value* of the PRF at $x^{(k)}$, which is hard when the PRF has a super-polynomial range and the PRF satisfies puncturing security. The fact that we do *not* rely on the unpredictability of the special point for security is a subtle but important distinction in our construction. In Section 5, we show that assuming the underlying PRF provides robust extractability (and key-injectivity), the adversary can simulate for itself the behavior of the extraction oracle. Thus, the presence of the extraction oracle cannot help the adversary break unremovability.

Unforgeability and message-embedding via multi-puncturing. While the basic construction above provides unremovability, it is easy to forge watermarked programs. Namely, an adversary can simply take a circuit that implements a PRF $\mathsf{F}(k, \cdot)$ and randomly corrupt a $(1/\mathsf{poly}(\lambda))$-fraction of the output (where λ is a security parameter). Then, with noticeable probability, the adversary will corrupt the PRF at the special point $x^{(k)}$ associated with k, thereby causing the verification algorithm to conclude that the circuit is marked. This is easily prevented by puncturing k at λ points $x_1^{(k)}, \ldots, x_\lambda^{(k)}$. We now say that a circuit C is marked only if $C(x_i^{(k)}) \neq \mathsf{F}(k, x_i^{(k)})$ for all $i \in [\lambda]$. Of course, this modification does not affect unremovability. Now, to forge a watermarked program, the

adversary has to construct a circuit C whose behavior closely resembles $\mathsf{F}(k,\cdot)$, and yet, C and $\mathsf{F}(k,\cdot)$ disagree on *all* of the special points $x_1^{(k)},\ldots,x_\lambda^{(k)}$, which are derived pseudorandomly from the key k. This means that unless the adversary previously made a request to mark k (in which case its circuit C would no longer be considered a forgery), the points $x_1^{(k)},\ldots,x_\lambda^{(k)}$ associated with k look uniformly random to $\mathcal{A}$. But now, if C and $\mathsf{F}(k,\cdot)$ are close, they will not differ on λ random points, except with negligible probability. We formalize this argument in Section 5.2.

The same technique of puncturing at multiple points also enables us to extend our basic mark-embedding watermarking scheme into a scheme that supports message-embedding. We take a basic bit-by-bit approach similar in spirit to the ideas taken in [43, 38, 45]. Specifically, to support embedding messages of length t in a PRF key k, we first derive from k a collection of λ pseudorandom points for each index and each possible bit: $S_{i,b}^{(k)} = \{x_{i,b,1}^{(k)},\ldots,x_{i,b,\lambda}^{(k)}\}$ for all $i \in [t]$ and $b \in \{0,1\}$. To embed a message $m \in \{0,1\}^t$ in the key k, the marking algorithm punctures k at all of the points in the sets $S_{i,m_i}^{(t)}$ for $i \in [t]$. To recover the watermark, the extraction algorithm proceeds very similarly as before. Specifically, on input a circuit C, the extraction algorithm uses the trapdoor for the underlying extractable PRF to obtain a candidate key k (or outputs UNMARKED if no key is extracted). Given a candidate key k, the extraction algorithm derives the sets $S_{i,b}^{(k)}$ for each index $i \in [t]$ and bit $b \in \{0,1\}$. For each index, the algorithm counts the number of points in $S_{i,0}^{(k)}$ and $S_{i,1}^{(k)}$ on which C and $\mathsf{F}(k,\cdot)$ disagree. For correctly-watermarked keys, C and $\mathsf{F}(k,\cdot)$ will disagree on all of the points in one of the sets and none of the points in the other set. This difference in behavior allows the extraction algorithm to recover the bit at index i. We provide the full description and analysis in the full version of this paper [39].

Public marking in the random oracle model. In the mark-embedding and message-embedding watermarking constructions we have described so far, both marking and extraction require knowledge of the watermarking secret key. If we look more closely at the marking algorithm, however, we see that the only time the watermarking key is used during marking is to derive the set of points to be punctured (specifically, the set of points to be punctured is derived by evaluating a PRF on the key k). Critically, we do *not* require that the set of punctured points be hidden from the adversary (and indeed, the watermarked key completely reveals the set of punctured points), but only that they are unpredictable (without knowledge of k). Thus, instead of using a PRF to derive the points to be punctured, we can use a random oracle. This gives a construction of a message-embedding watermarking scheme that supports *public marking*. We provide the full description and analysis of this scheme in the full version of this paper [39]. We note that Quach et al. [45] were the first to give a watermarking scheme that supported public marking *without* random oracles (for mark-embedding, they only needed CCA-secure public-key encryption while for full message-embedding, they relied on lattices). However, as noted before, their

scheme does not provide any security against a malicious watermarking authority (or provide unforgeability, which we discuss below).

Unforgeability and public marking. Recall that unforgeability for a watermarking scheme says that no efficient adversary should be able to construct a marked circuit that is significantly different from marked circuits it already received. This property seems at odds with the semantics of a watermarking scheme that supports public marking, since in the latter, anyone can mark programs of their choosing. However, we can still capture the following spirit of unforgeability by requiring that the only marked circuits that an adversary can construct are those that are close to circuits that are contained in the function class. In the case of watermarking PRFs, this means that the only circuits that would be considered to be watermarked are those that are functionally close to a legitimate PRF. This property is useful in scenarios where the presence of a watermark is used to argue *authenticity* of software (e.g., to prove to someone that the software implements a specific type of computation). In this work, we introduce a weaker notion of unforgeability that precisely captures this authenticity property. We then show that our watermarking construction supports public-marking while still achieving this form of weak unforgeability. The only previous candidate of software watermarking that supports public marking [45] does not satisfy this property, and indeed, in their scheme, it is easy to construct functions that are constant everywhere (which are decidedly *not* pseudorandom), but nonetheless would be considered to be marked.

Optimal bounds for unremovability and unforgeability. We say that a watermarking scheme is ε-unremovable if an adversary who only changes an ε-fraction of the values of a marked circuit cannot remove the watermark,[4] and that it is δ-unforgeable if an adversary cannot create a new marked program that differs on at least a δ-fraction of points from any marked circuits it was given. Conceptually, larger values of ε means that the watermark remains intact even if the adversary can corrupt the behavior of the marked program on a larger fraction of inputs, while smaller values of δ means that the adversary's forgery is allowed to agree on a larger fraction of the inputs of a marked program. Previously, Cohen et al. [27] showed that any message-embedding watermarking scheme can at best achieve $\varepsilon = 1/2 - 1/\mathsf{poly}(\lambda)$ and $\delta = \varepsilon + 1/\mathsf{poly}(\lambda)$. Our constructions in this work achieve both of these bounds (for any choice of $\mathsf{poly}(\lambda)$ factors). Previous constructions like [27, 45] did not provide unforgeability while [16, 38] could only tolerate $\varepsilon = \mathsf{negl}(\lambda)$ (and any $\delta = 1/\mathsf{poly}(\lambda)$).

Security against the watermarking authority. The key property of extractable PRFs that underlies our watermarking constructions is that there is an extraction trapdoor td that can be used to extract the original PRF key k from any circuit whose behavior is sufficiently similar to that of $\mathsf{F}(k, \cdot)$. In the case of watermarking, the watermarking authority must hold the trapdoor to

[4] This definition is the complement of the definition from previous works on watermarking [27, 16, 38, 49, 45, 50], but we adopt this to maintain consistency with our definition for robust extractability.

use it to extract watermarks from marked programs. This raises a new security concern as the watermarking authority can now break security of *all* PRFs in the family, including *unmarked* ones. As discussed in Section 1.1, this was the main drawback of the Quach et al. [45] watermarking construction.

Due to our reliance on extractable PRFs, our watermarkable family of PRFs also cannot satisfy full pseudorandomness against the watermarking authority. However, we can show a weaker property against the watermarking authority we call T-restricted pseudorandomness. Namely, we can associate a set $S \subseteq \mathcal{X}$ of size at most T with our watermarkable family of PRFs such that any adversary (even if they have the extraction trapdoor) is unable to break pseudorandomness of any (unmarked) PRF, provided that they do not query the function on points in S. The distinguisher is also provided the set S. In other words, our family of PRFs still provides pseudorandomness everywhere except S. In our concrete constructions (Construction 4.5), the restricted set S consists of λ *randomly-chosen* points in $\mathcal{X}$. This means that if the domain of the PRF is super-polynomial, our notion of T-restricted pseudorandomness strictly interpolates between weak pseudorandomness (or even non-adaptive pseudorandomness)[5] and strong pseudorandomness. It is also worth noting that from the perspective of a user who does not hold the watermarking secret key, the points in S are *statistically hidden*. This means that in any standard usage of the PRF between honest users, with overwhelming probability, the PRF would never be evaluated on one of the restricted points. Equivalently, if the watermarking authority only sees passive evaluations of the PRF, then it will not be able to break pseudorandomness of the underlying PRF. This notion of "passive" security against the watermarking authority strictly improves upon the lattice-based message-embedding watermarking construction in [45]. In their setting, the watermarking authority is able to break pseudorandomness given *any* two (distinct) evaluations of the PRF; that is, their scheme does not even satisfy weak pseudorandomness against the watermarking authority. It is an interesting and important question to obtain watermarking with security in the presence of an extraction oracle and which retrains full pseudorandomness even against the watermarking authority. The only constructions that satisfy this notion rely on obfuscation.

Watermarking *without* private puncturing. All existing constructions of message-embedding watermarking from standard assumptions have relied on *private* puncturable PRFs[6] in some form [38, 45]. Our message-embedding watermarking construction is the first that does *not* rely on private puncturing; standard puncturing in conjunction with key-extractability suffices. While this

[5] In the weak pseudorandomness game, the adversary is given outputs of the PRF on random inputs, while in the non-adaptive pseudorandomness game, the adversary must declare *all* of its evaluation queries before seeing any evaluations of the PRF or the public parameters.

[6] A private puncturable PRF [16] is a puncturable PRF where the punctured key also *hides* the punctured point. There are several lattice-based constructions of private puncturable PRFs (and more generally, private constrained PRFs) [25, 14, 21, 44, 26].

might seem like a minor distinction, we note that constrained PRFs can be constructed from weaker assumptions. For instance, puncturable PRFs can be built from one-way functions [30, 18, 37, 19] while the simplest constructions of private puncturable PRFs rely on lattice-based assumptions [14, 25, 21, 44, 26]. If we just consider lattice-based constrained PRFs, the Brakerski-Vaikuntanathan puncturable PRF [23] can be based on the (polynomial) hardness of solving worst-case lattice problems with a *nearly polynomial* approximation factor (i.e., $n^{\omega(1)}$),[7] while constructions of private puncturable PRFs from lattices [14, 25, 21, 44, 26] can only be based on the hardness of solving worst-case lattice problems with a *quasi-polynomial* approximation factor (i.e., $2^{\log^c n}$ for some constant $c > 1$). Since all of the existing constructions of message-embedding watermarking from standard assumptions rely on private puncturing in some form, they can only be reduced to worst-case lattice problems with quasi-polynomial approximation factors at best. In this work, we show that a variant of our construction (satisfying a relaxed notion of unforgeability as in [38]) can be based solely on worst-case lattice problems with a nearly polynomial approximation factor (Remark 4.13). Concretely, we give the first (message-embedding) watermarking scheme whose security can be based on computing nearly polynomial approximations to worst-case lattice problems (Corollary 5.4).

1.3 Additional Related Work

We now survey some additional works that use similar techniques as those in our construction.

Lattice-based PRFs. The study of lattice-based PRFs started with the seminal work of Banerjee et al. [8]. Subsequently, [15, 7] constructed the first lattice-based key-homomorphic PRFs. The first circuit-constrained PRFs were constructed in [23, 6] and were later extended to private constrained PRFs in [14, 25, 21, 44, 26].

Matrix embeddings. The matrix embedding techniques used in this work build on a series of works in the areas of attribute-based encryption [47] and predicate encryption [17, 36] from LWE. These include the attributed-based encryption constructions of [1, 31, 12, 33, 24, 20] and the (one-sided) predicate encryption constructions of [2, 28, 32, 21, 34, 48].

2 Technical Overview

In this section, we provide a technical overview of our construction of extractable PRFs from standard lattice assumptions. As described in Section 1.2, this is

[7] While the general construction described in [23] relies on worst-case lattice problems with sub-exponential approximation factors, when restricted to just puncturing constraints (which can be computable by *log-depth* circuits), it can be based on worst-case lattice problems with a nearly polynomial approximation factor by leveraging the techniques for branching program evaluation [22].

the key cryptographic primitive we rely on in our watermarking constructions (described formally in Section 5). We believe that the algebraic techniques we develop for constructing our extractable PRF are general and will find applications beyond the study of PRFs and watermarking. We highlight the core principles and techniques here, but defer the formal definitions, constructions, and analysis to Section 4.

The LWE assumption. The learning with errors (LWE) assumption [46], parameterized by n, m, q, χ, states that for a uniformly random vector $\mathbf{s} \in \mathbb{Z}_q^n$, a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and a noise vector $\mathbf{e}$ sampled from a (low-norm) error distribution χ, the distribution $(\mathbf{A}, \mathbf{s} \cdot \mathbf{A} + \mathbf{e})$[8] is computationally indistinguishable from the uniform distribution over $\mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^m$. Equivalently, rather than explicitly adding noise, the LWE assumption can instead be defined with respect to a rounding modulus $p < q$ and the component-wise rounding operation $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ [8]. This variant of the LWE assumption states that the distribution $(\mathbf{A}, \lfloor \mathbf{s} \cdot \mathbf{A} \rceil_p)$ is computationally indistinguishable from the uniform distribution over $\mathbb{Z}_q^{n \times m} \times \mathbb{Z}_p^m$; this is also known as the *learning with rounding* (LWR) assumption [8]. For the parameter setting we consider in this work, hardness of LWE implies hardness of LWR [8].

Lattice-based PRFs. A natural way to construct a pseudorandom function $\mathsf{F} \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ from the LWE assumption is to take the PRF key $k \in \mathcal{K}$ to be the LWE secret $\mathbf{s} \in \mathbb{Z}_q^n$ and define $\mathsf{F}(\mathbf{s}, x)$ to output an LWE sample $\lfloor \mathbf{s} \cdot \mathbf{A}_x \rceil_p$ for a matrix $\mathbf{A}_x$ that is uniquely determined by the input $x \in \mathcal{X}$. Note that when the domain $\mathcal{X}$ is super-polynomial, the matrix $\mathbf{A}_x$ cannot be a uniformly random matrix as required by the LWE assumption since $\mathsf{F}(\mathbf{s}, \cdot)$ must be an (efficiently-computable) deterministic function. Constructing a PRF from LWE thus amounts to designing a suitable mapping $x \mapsto \mathbf{A}_x$ such that the vector $\lfloor \mathbf{s} \cdot \mathbf{A}_x \rceil_p$ is still pseudorandom under LWE.

Nearly all existing LWE-based PRF constructions follow this general blueprint; we refer to Section 1.3 for a more comprehensive discussion of related work. Specifically, these PRF families are defined with respect to a set of public matrices $\mathsf{pp} = (\mathbf{A}_1, \ldots, \mathbf{A}_\rho)$ and an input-to-matrix mapping $\mathsf{Eval}_{\mathsf{pp}} \colon \mathcal{X} \to \mathbb{Z}_q^{n \times m}$ (that implements the mapping $x \mapsto \mathbf{A}_x$) such that the outputs of

$$\mathsf{F}(\mathbf{s}, x) := \lfloor \mathbf{s} \cdot \mathbf{A}_x \rceil_p \text{ where } \mathbf{A}_x \leftarrow \mathsf{Eval}_{\mathsf{pp}}(x) \tag{2.1}$$

are computationally indistinguishable from uniform vectors over $\mathbb{Z}_p^n$ under the LWE assumption. In this overview, rather than focusing on a particular PRF construction, we show how to obtain an extractable PRF from any lattice-based PRF family that follows this blueprint.

Key extraction via lattice trapdoors. Recall from Section 1 that in an extractable PRF family, the holder of a trapdoor td (for the PRF family) can recover the PRF key $k \in \mathcal{K}$ given only oracle access to the PRF $\mathsf{F}(k, \cdot)$. Using the basic structure of lattice-based PRF candidates from Eq. (2.1), a natural starting

[8] For notational simplicity, we drop the transpose notation when it is clear from context.

point is to design the mapping $\mathsf{Eval}_{\mathsf{pp}}\colon \mathcal{X} \to \mathbb{Z}_q^{n\times m}$ such that for a special input $x^* \in \mathcal{X}$, the matrix $\mathbf{D} \leftarrow \mathsf{Eval}_{\mathsf{pp}}(x^*)$ has a known (lattice) trapdoor $\mathsf{td}_{\mathbf{D}}$, which can be included as part of the trapdoor for the extractable PRF family (together with the special input x^*).

A lattice trapdoor $\mathsf{td}_{\mathbf{D}}$ for a matrix $\mathbf{D} \in \mathbb{Z}_q^{n\times m}$ enables sampling short preimages under the matrix $\mathbf{D}$ [3, 29, 4, 41, 40]. Specifically, given an arbitrary target matrix $\mathbf{T} \in \mathbb{Z}_q^{n\times m}$, the trapdoor $\mathsf{td}_{\mathbf{D}}$ enables sampling a short matrix $\mathbf{R}_{\mathbf{T}} \in \mathbb{Z}_q^{m\times m}$ such that $\mathbf{D}\cdot\mathbf{R}_{\mathbf{T}} = \mathbf{T}$. Additionally, the trapdoor for $\mathbf{D}$ can be used to solve the search version of the LWE problem: given an LWE instance $(\mathbf{D}, \lfloor \mathbf{s}\cdot\mathbf{D}\rceil_p)$, one first computes a short matrix $\mathbf{R}_{\mathbf{G}}$ using the trapdoor $\mathbf{D}$ and then derive the vector

$$\lfloor \mathbf{s}\cdot\mathbf{D}\rceil_p \cdot \mathbf{R}_{\mathbf{G}} = \lfloor \mathbf{s}\cdot\mathbf{D}\cdot\mathbf{R}_{\mathbf{G}}\rceil_p + \mathsf{noise} = \lfloor \mathbf{s}\cdot\mathbf{G}\rceil_p + \mathsf{noise} \in \mathbb{Z}_p^m,$$

where noise is a small error vector that occurs from the modular rounding and $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ is the standard powers-of-two gadget matrix [41]. Since $\mathbf{G}^T$ is the generator matrix for a linear error-correcting code, recovering $\mathbf{s}$ form $\lfloor \mathbf{s}\cdot\mathbf{G}\rceil_p + \mathsf{noise}$ is straightforward (c.f., [41]).

Given the trapdoor $\mathsf{td}_{\mathbf{D}}$, it is straightforward to implement the $\mathsf{Extract}$ algorithm. Namely, $\mathsf{Extract}$ first queries $\mathsf{F}(\mathbf{s},\cdot)$ on the special point $x^* \in \mathcal{X}$ to obtain the output $\lfloor \mathbf{s}\cdot\mathbf{D}\rceil_p$. It then uses the trapdoor information $\mathsf{td}_{\mathbf{D}}$ to recover the secret key $\mathbf{s}$.

Programming the trapdoor. The problem of constructing an extractable PRF family now boils down to generating a set of public parameters pp and a suitable mapping $\mathsf{Eval}_{\mathsf{pp}}\colon \mathcal{X} \to \mathbb{Z}_q^{n\times m}$ such that the matrix $\mathbf{A}_{x^*} \leftarrow \mathsf{Eval}_{\mathsf{pp}}(x^*)$ can be programmed to be a trapdoor matrix $\mathbf{D}$. At the same time, $\mathsf{Eval}_{\mathsf{pp}}$ must be designed so that the basic blueprint from Eq. (2.1) still satisfies pseudorandomness. The concept of programming the output of a PRF was previously explored in the context of constrained PRFs [16, 38, 25, 44]. These works study the notion of a *private programmable PRF* where *constrained keys* can be programmed to a specific value at a particular point (or set of points). However, the techniques used in these works do not directly apply to our setting as our goal is fundamentally different. To construct an extractable PRF, we need a PRF family such that the evaluation of *every* PRF key from the family is programmed to a trapdoor matrix. In fact, our notion is completely independent of constraining, and an extractable PRF family need not even support constraining. In other words, we want programmability with respect to the public parameters of the PRF family rather than just an individual PRF key.

The way we construct the function $\mathsf{Eval}_{\mathsf{pp}}$ is quite simple and general. We take any existing PRF construction $\mathsf{F}'\colon \mathbb{Z}_q^m \times \mathcal{X} \to \mathbb{Z}_p^m$ following the blueprint from Eq. (2.1) that is defined respect to a set of matrices $\mathsf{pp}' = (\mathbf{A}_1, \ldots, \mathbf{A}_\rho)$ and mapping $\mathsf{Eval}'_{\mathsf{pp}'}\colon \mathcal{X} \to \mathbb{Z}_q^{n\times m}$, and define a new *shifted* mapping

$$\mathsf{Eval}_{\mathsf{pp}}(x) := \mathsf{Eval}'_{\mathsf{pp}'}(x) + \mathbf{W} = \mathbf{A}_x + \mathbf{W},$$

for some shift matrix $\mathbf{W} \in \mathbb{Z}_q^{n \times m}$ and a new set of public matrices $\mathsf{pp} = (\mathbf{A}_1, \ldots, \mathbf{A}_\rho, \mathbf{W})$. First, observe that given a point $x^* \in \mathcal{X}$ and a trapdoor matrix $\mathbf{D}$, it is easy to generate a programmed set of public parameters:

1. Generate the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\rho \in \mathbb{Z}_q^{n \times m}$ as in the original PRF family.
2. Set $\mathbf{W} = \mathbf{D} - \mathbf{A}_{x^*}$ where $\mathbf{A}_{x^*} \leftarrow \mathsf{Eval}'_{\mathsf{pp}'}(x^*)$.

It is easy to see that security of the original PRF family is preserved. Specifically, we now have

$$\mathsf{F}(\mathbf{s}, x) = \lfloor \mathbf{s} \cdot (\mathbf{A}_x + \mathbf{W}) \rceil_p \approx \lfloor \mathbf{s} \cdot \mathbf{A}_x \rceil_p + \lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p = \mathsf{F}'(\mathbf{s}, x) + \lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p , \quad (2.2)$$

Since a randomly sampled trapdoor matrix $\mathbf{D}$ is statistically close to uniform, the matrix $\mathbf{W}$ is also statistically close to uniform. This means that the additional vector offset $\mathbf{w} = \lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p$ introduced by $\mathbf{W}$ looks indistinguishable from a uniformly random vector under LWE. Moreover,

$$\mathsf{F}(\mathbf{s}, x^*) = \lfloor \mathbf{s} \cdot (\mathbf{A}_{x^*} + \mathbf{W}) \rceil_p = \lfloor \mathbf{s} \cdot \mathbf{D} \rceil_p ,$$

so given the trapdoor $\mathsf{td}_{\mathbf{D}}$, it is easy to recover the key $\mathbf{s}$.

2.1 Robust Extractability

The PRF family $\mathsf{F} \colon \mathbb{Z}_q^n \times \mathcal{X} \to \mathbb{Z}_p^n$ defined in Eq. (2.2) already satisfies a basic notion of key-extractability. Namely, any authority who holds the trapdoor information $(x^*, \mathsf{td}_{\mathbf{D}})$ is able to extract the PRF key given just oracle access to the function $\mathsf{F}(\mathbf{s}, \cdot)$; moreover, $\mathsf{F}(\mathbf{s}, \cdot)$ remains pseudorandom to anyone who does not possess the trapdoor. To support watermarking, however, we require a stronger security property called *robust extractability* (Definition 4.3).

Robustness and key-injectivity. At a high level, robust extractability says that the $\mathsf{Extract}$ algorithm should successfully recover the PRF key even if it is just given access to a function (modeled as a circuit) whose behavior is "close" to $\mathsf{F}(\mathbf{s}, \cdot)$. In fact, even if the adversary has oracle access to $\mathsf{Extract}$, it should not be able to produce a circuit C whose behavior is sufficiently "close" to $\mathsf{F}(\mathbf{s}, \cdot)$ for some key $\mathbf{s} \in \mathbb{Z}_q^n$, and for which, the extraction algorithm fails to extract $\mathbf{s}$ from C. The closeness metric that we use in this work is ε-closeness; namely, we say that two circuits C and C' are ε-close if they agree on all but an ε-fraction of elements in the domain. In all of our constructions, $\varepsilon = 1/\mathsf{poly}(\lambda)$. Of course, for the extractability property to be well-defined, it should be the case that for distinct keys $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}_q^n$, $\mathsf{F}(\mathbf{s}_1, \cdot)$ and $\mathsf{F}(\mathbf{s}_2, \cdot)$ should be "far" apart. As discussed in Section 1.2, we capture this by defining a notion of *key-injectivity* similar in flavor to previous definitions from [27, 38], and then show (Theorem 4.8) that over the randomness used to sample the public parameters, the basic construction in Eq. (2.2) satisfies our key-injectivity property. Thus, in the subsequent discussion, we assume without loss of generality that if a circuit C is ε-close to $\mathsf{F}(\mathbf{s}, \cdot)$ for any $\varepsilon = 1/\mathsf{poly}(\lambda)$, then $\mathbf{s}$ is unique.

The basic PRF construction from Eq. (2.2) does *not* satisfy robust extractability (for *any* closeness parameter $\varepsilon = 1/\mathsf{poly}(\lambda)$). Specifically, the adversary can recover the special point $x^* \in \mathcal{X}$ using binary search. To mount the attack, the adversary first chooses a key $\mathbf{s} \in \mathbb{Z}_q^n$, and constructs the circuit $\mathsf{F}(\mathbf{s}, \cdot)$. The adversary then (arbitrarily) partitions the domain into two halves $\mathcal{X}_1$ and $\mathcal{X}_2$ and queries the extraction oracle on a circuit C that agrees with $\mathsf{F}(\mathbf{s}, \cdot)$ on $\mathcal{X}_1$ and outputs $\perp$ on $\mathcal{X}_2$. Depending on whether the extraction algorithm succeeds in recovering $\mathbf{s}$ or not, the adversary learns which of $\mathcal{X}_1$ or $\mathcal{X}_2$ contains the special point x^*. After a polynomial number of queries, the adversary learns x^*. Once the adversary learns the special point x^*, it can always cause extraction to fail on a circuit by simply having the circuit output $\perp$ on x^* (and $\mathsf{F}(\mathbf{s}, x)$ on all $x \neq x^*$). Moreover, this circuit agrees with $\mathsf{F}(\mathbf{s}, \cdot)$ on all but a single point (i.e., they agree on all but a negligible fraction of the domain when $|\mathcal{X}|$ is super-polynomial), which breaks robust extractability.

Defending against binary search. Effectively, the binary search attack relies on the fact that the adversary can easily construct circuits C such that the behavior of $\mathsf{Extract}$ on C (specifically, whether $\mathsf{Extract}$ succeeds or not) is correlated with the secret extraction trapdoor (specifically, the point x^*). To defend against this, we develop a way to ensure that the behavior of $\mathsf{Extract}$ on a circuit C depends *only* on properties of the circuit C (and *not* on the extraction trapdoor). If this is the case, then the extraction oracle does not leak information about the extraction trapdoor, and in turn, robust extractability holds. We note that this type of approach is conceptually very similar to the notion of *strong soundness* in the context of constructing *multi-theorem* argument systems in the designated-verifier setting [11, 13].[9] To achieve this, we proceed in two steps. First, we modify the $\mathsf{Extract}$ algorithm to force the adversary to only submit circuits that are very close to an actual PRF circuit $\mathsf{F}(\mathbf{s}, \cdot)$. Then, we tweak the construction to ensure that extraction queries on circuits C that are too close to a real PRF circuit are not helpful to the adversary. We describe this below.

- **Testing for closeness.** After the $\mathsf{Extract}$ algorithm recovers a candidate key $\mathbf{s} \in \mathbb{Z}_q^n$ from a circuit C, it additionally checks whether the behavior of the circuit C and $\mathsf{F}(\mathbf{s}, \cdot)$ are "similar." While computing the exact distance between C and $\mathsf{F}(\mathbf{s}, \cdot)$ cannot be done in polynomial time, it is straightforward to construct a randomized algorithm that accepts (with overwhelming probability)

[9] In designated-verifier argument systems, an adversary who has oracle access to the verifier can observe the verifier's behavior on different statements and proof strings. When the verifier's responses are correlated with its secret verification state, the prover can potentially leverage the leakage and compromise soundness. This is the so-called "verifier rejection" problem. Strong soundness is a property that says that the responses of the verifier depend only on the statement or proof string, and *not* on the secret verification state (the analog in our setting is that the behavior of the extraction oracle only depends on the input circuit and not the extraction trapdoor). This property is very useful for arguing soundness in the presence of a verification oracle for designated-verifier argument systems.

whenever C and $\mathsf{F}(\mathbf{s}, \cdot)$ are ε_1-close and rejects (with overwhelming probability) whenever C and $\mathsf{F}(\mathbf{s}, \cdot)$ are ε_2-far, for any choice of $\varepsilon_2 > \varepsilon_1 + 1/\mathsf{poly}(\lambda)$. This can be done by sampling random points $x_1, \ldots, x_\xi \xleftarrow{R} \mathcal{X}$ and counting the number of inputs where $C(x_i) = \mathsf{F}(\mathbf{s}, x_i)$. If the number of points on which the two circuits differ is greater than $\xi \cdot (\varepsilon_1 + \varepsilon_2)/2$, then the $\mathsf{Extract}$ algorithm outputs $\perp$. By choosing $\xi = \mathsf{poly}(\lambda)$ accordingly, we can appeal to standard concentration bounds and show that $\mathsf{Extract}$ will only output a candidate key when C and $\mathsf{F}(\mathbf{s}, \cdot)$ are at least ε_2-close. When applied to watermarking, the parameter ε_1 corresponds to the unremovability threshold while the parameter ε_2 corresponds to the unforgeability threshold.

– **Embedding multiple trapdoors.** The closeness test prevents the adversary from querying the extraction oracle on circuits that are more than ε_2-far from valid PRF circuits $\mathsf{F}(\mathbf{s}, \cdot)$, since the output of $\mathsf{Extract}$ on these queries is $\perp$ with overwhelming probability. However, since $\varepsilon_2 = 1/\mathsf{poly}(\lambda)$, the adversary can still query the extraction oracle on circuits that are ε_2-close to the real PRF circuit $\mathsf{F}(\mathbf{s}, \cdot)$. In this case, each query still (roughly) allows the adversary to rule out at least an ε_2-fraction of the domain, and so, in time $\mathsf{poly}(1/\varepsilon_2) = \mathsf{poly}(\lambda)$, the adversary is again able to extract the special point x^* for the PRF family.

  The second ingredient in our construction is to embed *multiple* trapdoors. Specifically, instead of just embedding a single lattice trapdoor at x^*, we instead embed λ distinct trapdoors at λ special points $x_1^*, \ldots, x_\lambda^* \xleftarrow{R} \mathcal{X}$. Now, on input a circuit C, the $\mathsf{Extract}$ algorithm evaluates C at each special point x_i^*, and use the lattice trapdoor inversion algorithm to obtain candidate keys $\mathbf{s}_i$. It performs the closeness test described above on each candidate key $\mathbf{s}_i$ and outputs $\mathbf{s}_i$ if the closeness test succeeds, and $\perp$ if none succeed. By key-injectivity, there can only be one key $\mathbf{s}$ where $\mathsf{F}(\mathbf{s}, \cdot)$ is ε_2-close to C whenever $\varepsilon_2 < 1/2$. At a very high level, the benefit of having multiple trapdoors is that the adversary has to corrupt the value at all of the trapdoors in order to cause the output of the $\mathsf{Extract}$ algorithm to differ (in a manner that is correlated with the secret extraction state). Since the special points $x_1^*, \ldots, x_\lambda^*$ are independently and uniformly distributed, and the adversary is effectively constrained to choosing circuits C which are ε_2-close to some $\mathsf{F}(\mathbf{s}, \cdot)$, the probability that the adversary succeeds in constructing such a circuit is $\varepsilon_2^\lambda = \mathsf{negl}(\lambda)$. We refer to Section 4.2 and Theorem 4.12 for the formal analysis.

2.2 Puncturing and Pseudorandomness Given the Trapdoor

Recall from Section 1.2 that to obtain a watermarking scheme from an extractable PRF, we additionally require that the extractable PRFs support puncturing constraints. Since our techniques for building extractable PRFs are broadly applicable to many lattice-based PRFs, we can take an existing candidate with the structure from Eq. (2.1) and derive from it an extractable PRF. In particular, we can apply our general construction to the Brakerski-Vaikuntanathan constrained

PRF [23], and obtain a puncturable extractable PRF. To achieve the stronger security notion of (T-restricted) pseudorandomness against an authority that holds the extraction trapdoor, we have to develop new techniques. We discuss the challenges below.

Security against the authority. As discussed in Section 1.2, a key contribution of our work is showing that the keys in our watermarkable PRF family still provide a relaxed form of pseudorandomness even against the holder of the watermarking secret key. This property amounts to showing that the underlying extractable PRF satisfies T-restricted pseudorandomness against an adversary who is given the extraction trapdoor. Specifically, we show that as long as the adversary (who has the trapdoor) is not allowed to query the PRF on the special points $x_1^*, \ldots, x_\lambda^*$, then pseudorandomness holds. This set of special points constitute the *restricted set* in the T-restricted pseudorandomness experiment. First, recall from Eq. (2.2) that

$$\mathsf{F}(\mathbf{s}, x) = \lfloor \mathbf{s} \cdot (\mathbf{A}_x + \mathbf{W}) \rceil_p \approx \mathsf{F}'(\mathbf{s}, x) + \lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p ,$$

where $\mathsf{F}'(\mathbf{s}, x)$ is the existing PRF (specifically, the Brakerski-Vaikuntanathan PRF [23]). At first glance, one might be tempted to believe that T-restricted pseudorandomness against the authority follows immediately from the security of F' since the value $\mathsf{F}(\mathbf{s}, x)$ is just $\mathsf{F}'(\mathbf{s}, x)$ shifted by $\lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p$ where $\mathbf{W} = \mathbf{D} - \mathbf{A}_{x^*}$. Without the extraction trapdoor, $\mathbf{D}$ is statistically close to uniform, so we can appeal to LWE to argue that the shift $\lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p$ is uniformly random (and looks independent of $\mathsf{F}'(\mathbf{s}, x)$). But given the trapdoor matrix $\mathbf{D}$, this is no longer the case; the shift $\lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p$ is *correlated* with the PRF key $\mathbf{s}$, and not easily simulated without knowing $\mathbf{s}$ itself. Thus, it is unclear how to directly reduce security of F to security of the underlying PRF F'.

Consider a potential reduction algorithm $\mathcal{B}$ that uses an adversary for F in the T-restricted pseudorandomness security game to break the security of F'. In this case, $\mathcal{B}$ is given the extraction trapdoor. If the reduction algorithm $\mathcal{B}$ is able to correctly simulate the evaluation $\mathsf{F}(\mathbf{s}, x)$ on all points $x \in \mathcal{X}$, then it can use its trapdoor information $\mathsf{td}_{\mathbf{D}}$ to extract $\mathbf{s}$ and break security of F' itself. Thus, for the proof to go through, we minimally need to rely on some type of "puncturing" argument (c.f., [23]). A possible starting point is to give the reduction algorithm $\mathcal{B}$ a punctured key k'_S for F' that enables evaluation of F' at all points except the restricted points $S = (x_1^*, \ldots, x_\lambda^*)$. Then, $\mathcal{B}$ can simulate the correct PRF evaluations at all non-restricted points, but it is unable to compute the evaluations at the special points for itself.

Unfortunately, this basic puncturing approach is still insufficient to prove security. Namely, even if the reduction algorithm can simulate the non-shifted PRF evaluation $\mathsf{F}'(\mathbf{s}, x)$ at all of the non-restricted points, it must still simulate the shift $\lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p$ *without* knowledge of the key $\mathbf{s}$. To address this, we additionally need to "program" the evaluations of the punctured key k_S at the non-punctured points. Specifically, we program the key k_S to introduce a shift by the key-dependent vector $\lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p$ at all of the non-punctured points. This latter step relies on an adaptation of the technique of *programmable matrix embeddings* from [38]. This

enables $\mathcal{B}$ to simulate the full PRF evaluation $\mathsf{F}(\mathbf{s}, x) = \mathsf{F}'(\mathbf{s}, x) + \lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p$ for the adversary. We refer to Section 4.2 for the full details of the construction and security analysis.

3 Preliminaries

We begin by introducing some of the notation we use in this work. For an integer $n \geq 1$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is sampled from $\mathcal{D}$; for a finite set S, we write $x \xleftarrow{\text{R}} S$ to denote that x is sampled uniformly from S.

Unless specified otherwise, we use λ to denote the security parameter. We say a function $f(\lambda)$ is negligible in λ, denoted by $\mathsf{negl}(\lambda)$, if $f(\lambda) = o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We say that an event happens with overwhelming probability if its complement happens with negligible probability. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We use $\mathsf{poly}(\lambda)$ to denote a quantity whose value is bounded by a fixed polynomial in λ. For two families of distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, we write $\mathcal{D}_1 \overset{s}{\approx} \mathcal{D}_2$ if the two distributions are statistically indistinguishable (i.e., the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is negligible). We now define the circuit-similarity metric we use in this work.

Definition 3.1 (Circuit Similarity). *Fix a circuit class $\mathcal{C}$ on ρ-bit inputs. For two circuits $C, C' \in \mathcal{C}$ and for a non-decreasing function $\varepsilon \colon \mathbb{N} \to \mathbb{N}$, we write say that C is ε-close to C', denoted $C \sim_\varepsilon C'$, if C and C' agree on all but an ε-fraction of inputs. More precisely, we write*

$$C \sim_\varepsilon C' \iff \Pr[x \xleftarrow{\text{R}} \{0,1\}^\rho : C(x) \neq C'(x)] \leq \varepsilon.$$

Similarly, we write $C \not\sim_\varepsilon C'$ to denote that C and C' differ on at least an ε-fraction of inputs.

We provide additional background on lattice-based cryptography in the full version of this paper [39].

4 Extractable PRF

In this section, we introduce the core notion of an extractable PRF that we use throughout this work. Due to space limitations, we just present our definition of *robust extractability*. In the full version of this paper [39], we provide the formal definitions of both pseudorandomness as well as our relaxed notion of T-restricted pseudorandomness from Section 1, where pseudorandomness holds on all but a small number (i.e., up to T) points.

Definition 4.1 (Extractable PRF). *An* extractable PRF *with key-space $\mathcal{K}$, domain $\mathcal{X}$, and range $\mathcal{Y}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{EPRF}} = (\mathsf{PrmsGen}, \mathsf{SampleKey}, \mathsf{Eval}, \mathsf{Extract})$ with the following syntax:*

– $\mathsf{PrmsGen}(1^\lambda) \to (\mathsf{pp}, \mathsf{td})$*: On input the security parameter* λ*, the parameter-generation algorithm outputs a set of public parameters* pp *and a trapdoor* td*.*
– $\mathsf{SampleKey}(\mathsf{pp}) \to k$*: On input the public parameters* pp*, the key-generation algorithm outputs a PRF key* $k \in \mathcal{K}$*.*
– $\mathsf{Eval}(\mathsf{pp}, k, x) \to y$*: On input the public parameters* pp*, a PRF key* $k \in \mathcal{K}$*, and input* $x \in \mathcal{X}$*, the PRF evaluation algorithm outputs a value* $y \in \mathcal{Y}$*.*
– $\mathsf{Extract}(\mathsf{pp}, \mathsf{td}, C) \to k/\perp$*: On input the public parameters* pp*, the trapdoor* td*, and a circuit* $C\colon \mathcal{X} \to \mathcal{Y}$*, the extraction algorithm outputs a key* $k \in \mathcal{K} \cup \{\perp\}$*. Without loss of generality, the* Extract *algorithm can also be defined to take a circuit whose domain is any* superset *of the PRF domain* $\mathcal{X}$*.*

The public parameters pp *of an extractable PRF induces a PRF family* $\mathsf{F}_{\mathsf{pp}}\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ *where* $\mathsf{F}_{\mathsf{pp}}(k, x) := \mathsf{Eval}(\mathsf{pp}, k, x)$ *and* $\mathsf{F}_{\mathsf{pp}}.\mathsf{KeyGen}(1^\lambda)$ *computes and returns* $k \leftarrow \mathsf{SampleKey}(\mathsf{pp})$*. Note that the description of the induced PRF family* F *does* not *include the trapdoor* td*.*

Definition 4.2 (Extract-and-Test). *An extractable PRF* $\Pi_{\mathsf{EPRF}} = (\mathsf{PrmsGen}, \mathsf{SampleKey}, \mathsf{Eval}, \mathsf{Extract})$ *with key-space* $\mathcal{K}$ *has an "extract-and-test" extraction algorithm if* Extract *can additionally be decomposed into two algorithms* $(\mathsf{ExtractCandidates}, \mathsf{TestCandidate})$ *with the following properties:*

– $\mathsf{ExtractCandidates}(\mathsf{pp}, \mathsf{td}, C) \to S$*: On input the public parameters* pp*, the trapdoor* td*, and a circuit* C*, the candidate extraction algorithm outputs a (possibly empty) set* $S \subseteq \mathcal{K}$ *of candidate keys, where* $|S| = \mathsf{poly}(\lambda)$*.*
– $\mathsf{TestCandidate}(\mathsf{pp}, \mathsf{td}, C, k) \to b$*: On input the public parameters* pp*, the trapdoor* td*, a circuit* $C\colon \mathcal{X} \to \mathcal{Y}$*, and a candidate key* $k \in \mathcal{K}$*, the test candidate algorithm outputs a bit* $b \in \{0, 1\}$*. Note that we allow* TestCandidate *to be a* randomized *algorithm.*

Moreover, the $\mathsf{Extract}(\mathsf{pp}, \mathsf{td}, C)$ *algorithm can be written as follows:*

– $\mathsf{Extract}(\mathsf{pp}, \mathsf{td}, C)$*: First invoke* $\mathsf{ExtractCandidates}(\mathsf{pp}, \mathsf{td}, C)$ *to obtain a set* $S \subseteq \mathcal{K}$ *of candidate keys. For each* $k \in S$*, compute* $b_k \leftarrow \mathsf{TestCandidate}(\mathsf{pp}, \mathsf{td}, C, k)$*. Output any* $k \in S$ *where* $b_k = 1$*. If* $b_k = 0$ *for all* $k \in S$*, output* $\perp$*.*

Definition 4.3 (Robust Extractability). *Fix a security parameter* λ *and closeness parameters* $\varepsilon_1, \varepsilon_2$*. Let* $\Pi_{\mathsf{EPRF}} = (\mathsf{PrmsGen}, \mathsf{SampleKey}, \mathsf{Eval}, \mathsf{Extract})$ *be an extractable pseudorandom function with key-space* $\mathcal{K}$*, domain* $\mathcal{X}$*, and range* $\mathcal{Y}$*. Suppose* Π_{EPRF} *has an extract-and-test extraction algorithm where for* $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PrmsGen}(1^\lambda)$ *and* $\varepsilon_1 < \varepsilon_2$*, the* TestCandidate *algorithm satisfies the following two properties:*

– *For all* $k \in \mathcal{K}$ *and* $C(\cdot) \sim_{\varepsilon_1} \mathsf{Eval}(\mathsf{pp}, k, \cdot)$*,* $\Pr[\mathsf{TestCandidate}(\mathsf{pp}, \mathsf{td}, C, k) = 1] = 1 - \mathsf{negl}(\lambda)$*.*
– *For all* $k \in \mathcal{K}$ *and* $C(\cdot) \not\sim_{\varepsilon_2} \mathsf{Eval}(\mathsf{pp}, k, \cdot)$*,* $\Pr[\mathsf{TestCandidate}(\mathsf{pp}, \mathsf{td}, C, k) = 1] = \mathsf{negl}(\lambda)$*.*

Next, for an adversary $\mathcal{A}$, we define two experiments $\mathsf{ExtReal}_{\mathcal{A}}(\lambda, \varepsilon_1, \varepsilon_2)$ and $\mathsf{ExtIdeal}_{\mathcal{A}}(\lambda, \varepsilon_1, \varepsilon_2)$:

- ***Setup phase:*** *At the start of both experiments, the challenger samples $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PrmsGen}(1^\lambda)$ and gives pp to $\mathcal{A}$.*
- ***Query phase:*** *Adversary $\mathcal{A}$ can issue any (polynomial) number of extraction queries to the challenger. On an extraction oracle query $C\colon \mathcal{X} \to \mathcal{Y}$, the challenger in the two experiments responds as follows:*
  - $\mathsf{ExtReal}$: *In the real experiment, the challenger replies with $\mathsf{Extract}(\mathsf{pp}, \mathsf{td}, C)$.*
  - $\mathsf{ExtIdeal}$: *In the ideal experiment, the challenger proceeds as follows:*
    - *If there exists a* unique *$k \in \mathcal{K}$ where $C(\cdot) \sim_{\varepsilon_2} \mathsf{Eval}(\mathsf{pp}, k, \cdot)$, the challenger computes $b_k \leftarrow \mathsf{TestCandidate}(\mathsf{pp}, \mathsf{td}, C, k)$. It replies with k if $b_k = 1$ and $\perp$ if $b_k = 0$.*
    - *Otherwise, the challenger replies with $\perp$.*
- ***Output phase:*** *Once the adversary $\mathcal{A}$ is done making queries, it outputs a bit $b \in \{0, 1\}$. This is the output of the experiment.*

We say that Π_{EPRF} satisfies $(\varepsilon_1, \varepsilon_2)$-robust extractability if for all (possibly unbounded) adversaries $\mathcal{A}$ making any polynomial number $Q = \mathsf{poly}(\lambda)$ queries, we have that

$$\left|\Pr\left[\mathsf{ExtReal}_{\mathcal{A}}(\lambda, \varepsilon_1, \varepsilon_2) = 1\right] - \Pr\left[\mathsf{ExtIdeal}_{\mathcal{A}}(\lambda, \varepsilon_1, \varepsilon_2) = 1\right]\right| = \mathsf{negl}(\lambda).$$

Remark 4.4 (Generalized Candidate Testing). In our constructions, we will require a generalized version of $\mathsf{TestCandidate}$ with the following properties:

- The $\mathsf{TestCandidate}$ algorithm is *publicly-computable*; namely, $\mathsf{TestCandidate}$ does *not* depend on the trapdoor td. To make this explicit, in the case where $\mathsf{TestCandidate}$ is publicly-computable, we write the algorithm as $\mathsf{TestCandidate}(\mathsf{pp}, C, k)$.
- If C_1, C_2 satisfy $C_1 \sim_\varepsilon C_2$ for some $\varepsilon = \mathsf{negl}(\lambda)$, then for all pp and all $k \in \mathcal{K}$,

$$\Pr[\mathsf{TestCandidate}(\mathsf{pp}, C_1, k) \neq \mathsf{TestCandidate}(\mathsf{pp}, C_2, k)] = \mathsf{negl}(\lambda).$$

- Instead of taking as input a candidate key $k \in \mathcal{K}$ as input, the $\mathsf{TestCandidate}$ can also take as input an arbitrary circuit $C'\colon \mathcal{X} \to \mathcal{Y}$, with the property that for $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PrmsGen}(1^\lambda)$ and $k \leftarrow \mathsf{SampleKey}(\mathsf{pp})$, and any circuit $C'\colon \mathcal{X} \to \mathcal{Y}$ where $C' \sim_\varepsilon \mathsf{Eval}(\mathsf{pp}, k, \cdot)$ and $\varepsilon = \mathsf{negl}(\lambda)$,

$$\{\mathsf{TestCandidate}(\mathsf{pp}, C, k)\} \overset{s}{\approx} \{\mathsf{TestCandidate}(\mathsf{pp}, C, C')\},$$

  where the randomness is taken over the random coins in $\mathsf{PrmsGen}$, $\mathsf{SampleKey}$, and $\mathsf{TestCandidate}$.

Key-injectivity. As discussed in Section 2, a property that is often useful in conjunction with robust extractability is key-injectivity. We give the formal definition in the full version of this paper [39].

4.1 Puncturable Extractable PRFs

In a puncturable PRF [18, 37, 19], the PRF key k can be used to derive a punctured key k_{x^*} that can be used to evaluate the PRF everywhere *except* the punctured point $x^* \in \mathcal{X}$. Moreover, the actual PRF value $\mathsf{F}(k, x^*)$ remains pseudorandom even given the punctured key. More generally, we can consider puncturing the PRF at a set $S \subseteq \mathcal{X}$. In this case, the punctured key k_S can be used to evaluate the PRF at all points in $\mathcal{X} \setminus S$, while the PRF values at points in S remain pseudorandom. This is also called a *constrained PRF* [18]. In our setting, we primarily consider puncturing at sets containing up to $\mathsf{poly}(\lambda)$ elements. We review the formal definitions in the full version of this paper [39].

4.2 Constructing Extractable PRFs

In this section, we present our extractable PRF family from standard lattice assumptions. Although our construction follows the main ideas that we outlined in Section 2, implementing these ideas algebraically is non-trivial. We begin with a brief algebraic overview of our construction.

Construction overview. As discussed in Section 2, our PRF family is defined with respect to a set of public matrices in $\mathbb{Z}_q^{n \times m}$, which we denote by $(\mathbf{A}_j)_{j\in[\rho]}$, $(\widetilde{\mathbf{A}}_{\alpha,\beta})_{\alpha\in[n],\beta\in[m]}$, $(\mathbf{B}_{i,j})_{i\in[t],j\in[\rho]}$, $(\mathbf{C}_j)_{j\in[\rho]}$, $\mathbf{V}$, and $\mathbf{W}$. Here, n, m, q are lattice parameters, t is the number of punctured points, and ρ is the bit-length of the PRF input. These matrices can be logically partitioned into three sets of matrices that handle different correctness or security goals.

- The matrices $(\mathbf{A}_j)_{j\in[\rho]}$, $(\widetilde{\mathbf{A}}_{\alpha,\beta})_{\alpha\in[n],\beta\in[m]}$ are used for the T-restricted pseudorandomness proof. As discussed in Section 2.2, handling the evaluation queries in T-restricted pseudorandomness requires generating a punctured key that is specifically programmed to enable simulation of the key-dependent shift (i.e., the $\lfloor \mathbf{s} \cdot \mathbf{W} \rceil_p$ term in Eq. (2.2)).
- The matrices $(\mathbf{B}_{i,j})_{i\in[t],j\in[\rho]}$, $(\mathbf{C}_j)_{j\in[\rho]}$, $\mathbf{V}$ implement the constrained PRF construction of [23].
- The matrix $\mathbf{W}$ is the shift matrix. As described in Section 2, matrix $\mathbf{W}$ is generated by first evaluating $\mathsf{Eval}_{\mathsf{pp}}$ on the rest of the matrices $(\mathbf{A}_j)_{j\in[\rho]}$, $(\widetilde{\mathbf{A}}_{\alpha,\beta})_{\alpha\in[n],\beta\in[m]}$, $(\mathbf{B}_{i,j})_{i\in[t],j\in[\rho]}$, $(\mathbf{C}_j)_{j\in[\rho]}$, $\mathbf{V}$, and then defining it as a corresponding shifted matrix from a trapdoor matrix $\mathbf{D}$.

As discussed in Section 2.1, to achieve robust extractability, we need to embed multiple trapdoors. We support this by simply concatenating together multiple copies of the PRF, where each copy is associated with one of the trapdoors. We now give the formal construction.

Construction 4.5 (Puncturable Extractable PRFs). Let λ be a security parameter, and $\varepsilon_1, \varepsilon_2$ be distance parameters where $0 < \varepsilon_1 < \varepsilon_2 < 1/2$, and $\varepsilon_2 \geq \varepsilon_1 + 1/\mathsf{poly}(\lambda)$. We define the following scheme parameters:

- (n, m, q, χ_B) – lattice parameters, where χ_B is a B-bounded distribution,

– p – the rounding modulus,
– t – a bound on the number of points to be punctured (indexed by i),
– ρ – the bit-length of the PRF input (indexed by j),
– η – the number of special points where we embed the extraction trapdoor (indexed by ℓ).

Throughout this section and in the analysis, we will assume that $n, m, t, \rho, \eta = \mathsf{poly}(\lambda)$. Let $(\mathsf{TrapGen}, \mathsf{Invert})$ be the lattice trapdoor algorithms (see full version of this paper [39]). For an input $x \in \{0,1\}^\rho$, we define the equality function $f_x^{\mathsf{eq}} \colon \{0,1\}^\rho \to \{0,1\}$ where

$$f_x^{\mathsf{eq}}(x^*) = \begin{cases} 1 & \text{if } x = x^* \\ 0 & \text{otherwise.} \end{cases}$$

More generally, for a set of points $S \subseteq \{0,1\}^\rho$ of size t (represented as a concatenation of the bit-strings in S), we define the containment function $f_x^{\mathsf{con}} \colon \{0,1\}^{t\rho} \to \{0,1\}$ where

$$f_x^{\mathsf{con}}(S) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{otherwise.} \end{cases}$$

Note that both the equality circuit f_x^{eq} and the containment circuit f_x^{con} for any $x \in \{0,1\}^\rho$ can be computed by a circuit of depth $d = O(\log \rho + \log t) = O(\log \lambda)$. Our (puncturable) extractable PRF $\Pi_{\mathsf{PRF}} = (\mathsf{PrmsGen}, \mathsf{SampleKey}, \mathsf{Eval}, \mathsf{Extract}, \mathsf{Puncture}, \mathsf{PunctureEval})$ with key-space $\mathcal{K} = [-B, B]^n$, domain $\mathcal{X} = \{0,1\}^\rho \backslash \{\mathbf{0}\}$, and range $\mathcal{Y} = \mathbb{Z}_p^{\eta m}$ is defined as follows:[10]

– $\mathsf{PrmsGen}(1^\lambda)$: On input the security parameter λ, the $\mathsf{PrmsGen}$ algorithm begins by sampling $(\mathbf{A}_j^{(\ell)})_{j\in[\rho]}$, $(\widetilde{\mathbf{A}}_{\alpha,\beta}^{(\ell)})_{\alpha\in[n],\beta\in[m]}$, $(\mathbf{B}_{i,j}^{(\ell)})_{i\in[t],j\in[\rho]}$, $(\mathbf{C}_j^{(\ell)})_{j\in[\rho]}$, $\mathbf{V}^{(\ell)}$ uniformly at random from $\mathbb{Z}_q^{n\times m}$ for every $\ell \in [\eta]$. It also samples a set of η special points $h^{(\ell)} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ along with trapdoor matrices $(\mathbf{D}^{(\ell)}, \mathsf{td}_{\mathbf{D}^{(\ell)}}) \leftarrow \mathsf{TrapGen}(1^\lambda)$ for all $\ell \in [\eta]$. Then, for all $\ell \in [\eta]$, it computes
  - $\mathbf{A}_{h^{(\ell)}}^{(\ell)} \leftarrow \mathsf{EvalP}_{\mathsf{pk}}\Big(f_{h^{(\ell)}}^{\mathsf{eq}}, (\mathbf{A}_j^{(\ell)})_{j\in[\rho]}, (\widetilde{\mathbf{A}}_{\alpha,\beta}^{(\ell)})_{\alpha\in[n],\beta\in[m]}\Big)$,
  - $\mathbf{B}_{h^{(\ell)}}^{(\ell)} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\Big(f_{h^{(\ell)}}^{\mathsf{con}}, (\mathbf{B}_{i,j}^{(\ell)})_{i\in[t],j\in[\rho]}\Big)$,
  - $\mathbf{C}_{h^{(\ell)}}^{(\ell)} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\Big(f_{h^{(\ell)}}^{\mathsf{eq}}, (\mathbf{C}_j^{(\ell)})_{j\in[\rho]}\Big)$,

  and defines the matrix

$$\mathbf{W}^{(\ell)} = \mathbf{A}_{h^{(\ell)}}^{(\ell)} + \mathbf{B}_{h^{(\ell)}}^{(\ell)} \mathbf{G}^{-1}\big(\mathbf{C}_{h^{(\ell)}}^{(\ell)}\big)\mathbf{G}^{-1}\big(\mathbf{V}^{(\ell)}\big) + \mathbf{D}^{(\ell)} \in \mathbb{Z}_q^{n\times m}. \tag{4.1}$$

  Finally, it outputs

$$\mathsf{pp} = \Big(\mathbf{W}^{(\ell)}, \big(\mathbf{A}_j^{(\ell)}\big)_{j\in[\rho]}, \big(\widetilde{\mathbf{A}}_{\alpha,\beta}^{(\ell)}\big)_{\alpha\in[n],\beta\in[m]}, \big(\mathbf{B}_{i,j}^{(\ell)}\big)_{i\in[t],j\in[\rho]}, \big(\mathbf{C}_j^{(\ell)}\big)_{j\in[\rho]}, \mathbf{V}^{(\ell)}\Big)_{\ell\in[\eta]}, \tag{4.2}$$

  and $\mathsf{td} = \big(h^{(\ell)}, \mathsf{td}_{\mathbf{D}^{(\ell)}}\big)_{\ell\in[\eta]}$.

[10] We refer to the full version of this paper [39] for the specification of the $\mathsf{Eval}_{\mathsf{pk}}$, $\mathsf{Eval}_{\mathsf{ct}}$, $\mathsf{EvalP}_{\mathsf{pk}}$, and $\mathsf{EvalP}_{\mathsf{ct}}$ algorithms for computing on matrix embeddings [12, 38].

– $\mathsf{SampleKey}(\mathsf{pp})$: On input the public parameters pp, the key-generation algorithm samples a key $\mathbf{s} \leftarrow \chi^n$, and outputs the PRF key $k = \mathbf{s}$.
– $\mathsf{Eval}(\mathsf{pp}, k, x)$: On input the public parameters pp (as specified in Eq. (4.2)), a PRF key $k = \mathbf{s}$, and an input $x \in \{0,1\}^\rho \setminus \{\mathbf{0}\}$, the evaluation algorithm first computes the matrices
  - $\mathbf{A}_x^{(\ell)} \leftarrow \mathsf{EvalP}_{\mathsf{pk}}\Big(f_x^{\mathsf{eq}}, (\mathbf{A}_j^{(\ell)})_{j\in[\rho]}, (\widetilde{\mathbf{A}}_{\alpha,\beta}^{(\ell)})_{\alpha\in[n],\beta\in[m]}\Big)$,
  - $\mathbf{B}_x^{(\ell)} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\Big(f_x^{\mathsf{con}}, (\mathbf{B}_{i,j}^{(\ell)})_{i\in[t],j\in[\rho]}\Big)$,
  - $\mathbf{C}_x^{(\ell)} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\Big(f_x^{\mathsf{eq}}, (\mathbf{C}_j^{(\ell)})_{j\in[\rho]}\Big)$,

  for all $\ell \in [\eta]$. Then, it sets

$$\mathbf{Z}_x^{(\ell)} = \mathbf{A}_x^{(\ell)} + \mathbf{B}_x^{(\ell)}\mathbf{G}^{-1}(\mathbf{C}_x^{(\ell)})\mathbf{G}^{-1}(\mathbf{V}^{(\ell)}), \tag{4.3}$$

  for all $\ell \in [\eta]$, and computes the vector

$$\widetilde{\mathbf{y}}_x = \mathbf{s}\big(\mathbf{W}^{(1)} - \mathbf{Z}_x^{(1)} \mid \cdots \mid \mathbf{W}^{(\eta)} - \mathbf{Z}_x^{(\eta)}\big) \in \mathbb{Z}_q^{\eta m}. \tag{4.4}$$

  Finally, it outputs the rounded vector $\mathbf{y}_x = \lfloor \widetilde{\mathbf{y}} \rceil_p \in \mathbb{Z}_p^{\eta m}$.
– $\mathsf{Extract}(\mathsf{pp}, \mathsf{td}, C)$: The extraction algorithm is defined with respect to two sub-algorithms $\mathsf{ExtractCandidates}$ and $\mathsf{TestCandidate}$ (as in Definition 4.2) that are defined as follows:
  - $\mathsf{ExtractCandidates}(\mathsf{pp}, \mathsf{td}, C)$: On input the public parameters pp (as specified in Eq. (4.2)), a trapdoor $\mathsf{td} = \big(h^{(\ell)}, \mathsf{td}_{\mathbf{D}^{(\ell)}}\big)_{\ell\in[\eta]}$, and a circuit $C\colon \{0,1\}^\rho \to \mathbb{Z}_p^{\eta m}$, the candidate extraction algorithm evaluates the circuit on the test points to get $\mathbf{y}^{(\ell)} \leftarrow C\big(h^{(\ell)}\big)$ for all $\ell \in [\eta]$. Then, for all $\ell \in [\eta]$, it parses the vector $\mathbf{y}^{(\ell)} = (\mathbf{y}_1^{(\ell)} \mid \cdots \mid \mathbf{y}_\eta^{(\ell)})$ where each $\mathbf{y}_1^{(\ell)}, \ldots, \mathbf{y}_\eta^{(\ell)} \in \mathbb{Z}_p^m$. Then, it extracts $\mathbf{s}^{(\ell)} \leftarrow \mathsf{Invert}(\mathsf{td}_{\mathbf{D}^{(\ell)}}, \mathbf{y}_\ell^{(\ell)})$ and outputs the set of all $\mathbf{s}^{(\ell)}$ for which $\mathbf{s}^{(\ell)} \neq \perp$ and $\mathbf{s}^{(\ell)} \in [-B, B]^n$.
  - $\mathsf{TestCandidate}(\mathsf{pp}, C, k)$: Let $\delta = (\varepsilon_2 - \varepsilon_1)/2 = 1/\mathsf{poly}(\lambda)$, $\varepsilon = \varepsilon_1 + \delta$, and $\xi = \lambda/\delta^2 = \mathsf{poly}(\lambda)$. On input the public parameters pp, a key $k = \mathbf{s}$, and a circuit $C\colon \{0,1\}^\rho \to \mathbb{Z}_p^{\eta m}$, the test candidate algorithm samples $x_1^*, \ldots, x_\xi^* \xleftarrow{\text{R}} \{0,1\}^\rho$ and computes the number $N_{\mathbf{s}}$ of indices $i \in [\xi]$ where $C(x_i^*) \neq \mathsf{Eval}(\mathsf{pp}, \mathbf{s}, x_i^*)$. If $N_{\mathbf{s}} \leq \varepsilon\xi$, then output 1. Otherwise, output 0. Note that $\mathsf{TestCandidate}$ is *publicly-computable* (it does not require a trapdoor).

  The full extraction algorithm follows the extract-and-test procedure described in Definition 4.2.
– $\mathsf{Puncture}(\mathsf{pp}, k, S)$: On input the public parameter pp (as specified in Eq. (4.2)), a PRF key $k = \mathbf{s}$, and a set of points to be punctured $S = \{x_i\}_{i\in[t]}$, the $\mathsf{Puncture}$ algorithm first samples error vectors $\mathbf{e}_{\mathbf{A},j}^{(\ell)}, \mathbf{e}_{\widetilde{\mathbf{A}},\alpha,\beta}^{(\ell)}, \mathbf{e}_{\mathbf{B},i,j}^{(\ell)}, \mathbf{e}_{\mathbf{W}}^{(\ell)} \leftarrow \chi^m$ for all $i \in [t]$, $j \in [\rho]$, $\alpha \in [n]$, $\beta \in [m]$, and $\ell \in [\eta]$. Then, for each $\ell \in [\eta]$ it defines the vectors
  - $\mathbf{a}_j^{(\ell)} = \mathbf{s}\mathbf{A}_j^{(\ell)} + \mathbf{e}_{\mathbf{A},j}^{(\ell)}$ for all $j \in [\rho]$,
  - $\tilde{\mathbf{a}}_{\alpha,\beta}^{(\ell)} = \mathbf{s}\widetilde{\mathbf{A}}_{\alpha,\beta}^{(\ell)} + \mathbf{e}_{\widetilde{\mathbf{A}},\alpha,\beta}^{(\ell)}$ for all $\alpha \in [n]$ and $\beta \in [m]$,

- $\mathbf{b}_{i,j}^{(\ell)} = \mathbf{s}\big(\mathbf{B}_{i,j}^{(\ell)} + x_{i,j} \cdot \mathbf{G}\big) + \mathbf{e}_{\mathbf{B},i,j}^{(\ell)}$ for all $i \in [t]$ and $j \in [\rho]$,
- $\mathbf{w}^{(\ell)} = \mathbf{s}\mathbf{W}^{(\ell)} + \mathbf{e}_{\mathbf{W}}^{(\ell)}$.

Finally, it outputs the punctured key

$$k_S = \Big(S, \big(\mathbf{w}^{(\ell)}, (\mathbf{a}_j^{(\ell)})_{j\in[\rho]}, (\tilde{\mathbf{a}}_{\alpha,\beta}^{(\ell)})_{\alpha\in[n],\beta\in[m]}, (\mathbf{b}_{i,j}^{(\ell)})_{i\in[t],j\in[\rho]}\big)_{\ell\in[\eta]}\Big). \tag{4.5}$$

– $\mathsf{PunctureEval}(\mathsf{pp}, k_S, x)$: On input the public parameters pp (as specified in Eq. (4.2)), the punctured key k_S (as specified in Eq. (4.5)), and an input $x \in \{0,1\}^\rho \setminus \{\mathbf{0}\}$, the punctured evaluation algorithm computes the following for each $\ell \in [\eta]$:

- $\mathbf{a}_x^{(\ell)} \leftarrow \mathsf{EvalP}_{\mathsf{ct}}\big(f_x^{\mathsf{eq}}, \mathbf{0}, (\mathbf{A}_j^{(\ell)})_{j\in[\rho]}, (\widetilde{\mathbf{A}}_{\alpha,\beta}^{(\ell)})_{\alpha\in[n],\beta\in[m]}, (\mathbf{a}_j^{(\ell)})_{j\in[\rho]}, (\tilde{\mathbf{a}}_{\alpha,\beta}^{(\ell)})_{\alpha\in[n],\beta\in[m]}\big)$,
- $\mathbf{b}_x^{(\ell)} \leftarrow \mathsf{Eval}_{\mathsf{ct}}\big(f_x^{\mathsf{con}}, S, (\mathbf{B}_{i,j}^{(\ell)})_{i\in[t],j\in[\rho]}, (\mathbf{b}_{i,j}^{(\ell)})_{i\in[t],j\in[\rho]}\big)$,
- $\mathbf{C}_x^{(\ell)} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\big(f_x^{\mathsf{eq}}, (\mathbf{C}_j^{(\ell)})_{j\in[\rho]}\big)$.

Then, for each $\ell \in [\eta]$, it sets

$$\mathbf{z}_x^{(\ell)} = \mathbf{a}_x^{(\ell)} + \mathbf{b}_x^{(\ell)}\mathbf{G}^{-1}(\mathbf{C}_x^{(\ell)})\mathbf{G}^{-1}(\mathbf{V}^{(\ell)}), \tag{4.6}$$

and computes the vector

$$\mathbf{y}_x = (\mathbf{w}^{(1)} - \mathbf{z}_x^{(1)} \mid \cdots \mid \mathbf{w}^{(\eta)} - \mathbf{z}_x^{(\eta)}) \in \mathbb{Z}_q^{\eta m}. \tag{4.7}$$

Finally, it outputs the rounded vector $\mathbf{y}_x = \lfloor \widetilde{\mathbf{y}}_x \rceil_p \in \mathbb{Z}_p^{\eta m}$.

Security analysis. We now show that under the LWE and 1D-SIS-R [23, 14] assumptions (with suitable parameters),[11] the puncturable extractable PRF construction from Construction 4.5 satisfies correctness, puncturing security, and robust extractability. We give the formal theorem statements here, but defer the formal proofs to the full version of this paper [39]. We also discuss adaptive security in the full version.

Theorem 4.6 (Perfect Correctness for Most Keys). *Fix a security parameter λ and lattice parameters n, m, q, p, B. Suppose $m = \Omega(n \log q)$, $q = \Omega(np\sqrt{\log q})$, and $2^\rho B \cdot m^{O(\log \lambda)} \cdot p/q = \mathsf{negl}(\lambda)$. Then, the extractable PRF Π_{EPRF} from Construction 4.5 satisfies perfect correctness for most keys.*

Theorem 4.7 (Almost-Functionality-Preserving for All Keys). *Fix a security parameter λ and lattice parameters n, m, q, p, B. Suppose $m = \Omega(n \log q)$, $q = \Omega(np\sqrt{\log q})$, and $\rho = \omega(\log \lambda)$. Then, under the $\mathsf{1D\text{-}SIS\text{-}R}_{m',p,q,E}$ assumption for $m' = nm\eta$ and $E = B \cdot m^{O(\log \lambda)}$, the extractable PRF Π_{EPRF} from Construction 4.5 is almost-functionality-preserving for all keys.*

Theorem 4.8 (Key-Injectivity). *Fix a security parameter λ and lattice parameters n, m, q, p, B. Suppose $m = \Omega(n \log q)$, $q = \Omega(np\sqrt{\log q})$, and $2^\rho(4B+1)^n/p^{\eta m} = \mathsf{negl}(\lambda)$. Then, the extractable PRF Π_{EPRF} from Construction 4.5 satisfies key-injectivity.*

[11] We refer to full version of this paper [39] for the formal statements of these assumptions.

Theorem 4.9 (Puncturing Security). *Fix a security parameter λ and lattice parameters n, m, q, p, B. Suppose $m = \Omega(n \log q)$, $q = \Omega(np\sqrt{\log q})$, and $2^\rho B \cdot m^{O(\log \lambda)} \cdot p/q = \mathsf{negl}(\lambda)$. Then, under the $\mathsf{LWE}_{n,m',q,\chi}$ assumption for $m' = \eta m(nm + (t+2)\rho + 1) + \eta m$, the extractable PRF Π_{EPRF} from Construction 4.5 satisfies selective puncturing security.*

Corollary 4.10 (Pseudorandomness). *Fix a security parameter λ and lattice parameters n, m, q, p, B. Suppose the conditions in Theorem 4.9 hold. Then, the extractable PRF Π_{EPRF} from Construction 4.5 satisfies selective pseudorandomness.*

Theorem 4.11 (T-Restricted Psueodrandomness). *Fix a security parameter λ and lattice parameters n, m, q, p, B. Suppose $m = \Omega(n \log q)$ and $q = \Omega(np\sqrt{\log q})$ and $2^\rho B \cdot m^{O(\log \lambda)} \cdot p/q = \mathsf{negl}(\lambda)$. Then, under the $\mathsf{LWE}_{n,m',q,\chi}$ assumption for $m' = \eta m(nm + \rho(t+2)) + \eta m$, the extractable PRF Π_{EPRF} from Construction 4.5 satisfies selective T-restricted pseudorandomness for $T = \eta$.*

Theorem 4.12 (Robust Extractability). *Fix a security parameter λ and lattice parameters n, m, q, p, B. Take any $0 < \varepsilon_1 < \varepsilon_2 < 1/2$ where $\varepsilon_2 - \varepsilon_1 \geq 1/\mathsf{poly}(\lambda)$. Let Π_{EPRF} be the extractable PRF from Construction 4.5. Suppose $m = \Omega(n \log q)$, $q = \Omega(np\sqrt{\log q})$, $m \geq 2n \log q$, $\lceil q/p \rceil \leq q/4$, and $\eta = \omega(\log \lambda)$, and that Π_{EPRF} satisfies key-injectivity. Then, Π_{EPRF} satisfies $(\varepsilon_1, \varepsilon_2)$-robust extractability (Definition 4.3). Moreover, the $\mathsf{TestCandidate}$ algorithm in Construction 4.5 satisfies the generalized candidate testing properties from Remark 4.4.*

4.3 Concrete Parameter Instantiations

In the full version of this paper [39], we describe one possible instantiation for the parameters of the extractable PRF scheme in Construction 4.5. We choose our parameters so that the underlying LWE and 1D-SIS assumptions that we rely on reduce to approximating worst-case lattice problems to within a sub-exponential factor $2^{\tilde{O}(n^{1/c})}$ for some constant c where n is the lattice dimension.

Remark 4.13 (Extractable PRFs from Weaker Lattice Assumptions). If we relax the requirements on the extractable PRF and only require the standard notion of correctness (Theorem 4.6), then it is possible to instantiate the parameters such that all of the remaining properties only rely on the hardness of solving worst-case lattice problems with a *nearly polynomial* approximation factor. We provide more details in full version of this paper [39].

5 Watermarking from Puncturable Extractable PRFs

In this section, we show how to use our extractable PRF to construct a mark-embedding watermarking scheme in the secret-key setting. In the full version of this paper [39], we show how to extend this construction to obtain *message-embedding watermarking* from the same assumptions. We also show to obtain a scheme that supports *public marking* in the random oracle model.

5.1 Watermarking PRFs

We begin by formally introducing the notion of a watermarkable PRF family.

Definition 5.1 (Watermarkable Family of PRFs). *Fix a security parameter λ and a message space $\mathcal{M}$. A secretly-extractable, message-embedding watermarkable family of PRFs with key-space $\mathcal{K}$, a domain $\mathcal{X}$, and a range $\mathcal{Y}$ is a tuple of algorithms $\Pi_{\mathsf{WM}} = (\mathsf{Setup}, \mathsf{Mark}, \mathsf{Extract})$ with the following properties:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{wsk})$*: On input the security parameter λ, the setup algorithm outputs public parameters* pp *and the watermarking secret key* wsk.
- $\mathsf{Mark}(\mathsf{wsk}, k, m) \to C$*: On input the watermarking secret key* wsk*, a PRF key $k \in \mathcal{K}$, and a message $m \in \mathcal{M}$, the mark algorithm outputs a circuit $C \colon \mathcal{X} \to \mathcal{Y}$.*
- $\mathsf{Extract}(\mathsf{wsk}, C) \to m$*: On input the watermarking secret key* wsk *and a circuit $C \colon \mathcal{X} \to \mathcal{Y}$, the extraction algorithm outputs a string $m \in \mathcal{M} \cup \{\bot\}$.*

Moreover, Π_{WM} includes the description of a PRF family $\mathsf{F} \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. The description of the PRF family may include the public parameters pp *for the watermarkable PRF family, as sampled by the* Setup *algorithm. We often refer to Π_{WM} as a watermarking scheme for the PRF family $\mathsf{F} \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$.*

Remark 5.2 (Mark-Embedding Watermarking). To simplify the description of our construction (and just focus on the main ideas), we also consider the weaker notion of *mark-embedding* watermarking where programs are either considered to be MARKED or UNMARKED. Equivalently, this corresponds to Definition 5.1 where $\mathcal{M} = \{\text{MARKED}\}$. When describing a mark-embedding watermarking scheme, we simplify the Mark algorithm to only take in two parameters: the watermarking secret key wsk and the PRF key k. In this case, we will also often write UNMARKED in place of $\bot$.

Correctness. The two correctness requirements on a software watermarking scheme are that the watermarked keys (approximately) implement the same functionality as the original key. Moreover, the extraction algorithm should successfully extract the embedded message from a marked key. We give the formal definitions and some discussion in the full version of this paper [39].

Pseudorandomness. The second property we require on a watermarkable family of PRFs is the usual notion of pseudorandomness for the PRF family. As discussed in Section 1, we also consider a stronger notion where pseudorandomness should hold even against the watermarking authority (i.e., the holder of the watermarking secret key). While many existing watermarking schemes based on obfuscation or lattices [27, 16, 38] naturally satisfy this property, both our scheme and that of Quach et al. [45] do not provide full pseudorandomness. However, in our case, we can achieve the weaker notion of T-restricted pseudorandomness against the watermarking authority. Intuitively, this means that pseudorandomness is ensured even against the watermarking authority provided that the authority does not

see the PRF evaluations on any of T "special" points. We define this formally in the full version of this paper [39].

Unforgeability and unremovability. The main security notions for a cryptographic watermarking scheme we consider are unremovability and unforgeability. Conceptually, unremovability says that an efficient adversary cannot should not be able to remove a watermark from a marked program while unforgeability says that an adversary should not be able to construct a new marked program. We provide the formal definitions in the full version of this paper [39].

5.2 Mark-Embedding Watermarking

In this section, we present our basic construction of a mark-embedding watermarkable family of PRFs (in the secret-key setting) from extractable PRFs. We refer to Section 1.2 for a high-level overview of this construction. In the full version of this paper [39], we build upon this construction to obtain message-embedding watermarking (and, in the random oracle model, message-embedding watermarking with public marking).

Construction 5.3 (Mark-Embedding Watermarkable PRFs). Let λ be a security parameter. Our *mark-embedding* watermarkable PRF relies on the following primitives:

- Let $\Pi_{\mathsf{EPRF}} = (\mathsf{EX.PrmsGen}, \mathsf{EX.SampleKey}, \mathsf{EX.Eval}, \mathsf{EX.Extract}, \mathsf{EX.Puncture}, \mathsf{EX.PunctureEval})$ be a puncturable extractable PRF with key-space $\mathcal{K}_{\mathsf{EPRF}}$, domain $\mathcal{X}$, and range $\mathcal{Y}$.
- Let $\mathsf{PRF}\colon \mathcal{K}_{\mathsf{PRF}} \times \mathcal{K}_{\mathsf{EPRF}} \to \mathcal{X}^\lambda$ be a pseudorandom function.

We construct a watermarkable PRF $\Pi_{\mathsf{WM}} = (\mathsf{Setup}, \mathsf{Mark}, \mathsf{Extract})$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples a PRF key $k_{\mathsf{PRF}} \xleftarrow{R} \mathcal{K}_{\mathsf{PRF}}$, and parameters for the extractable PRF $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{EX.PrmsGen}(1^\lambda)$. It outputs the public parameters pp and the watermarking secret key $\mathsf{wsk} = (k_{\mathsf{PRF}}, \mathsf{pp}, \mathsf{td})$.
- $\mathsf{Mark}(\mathsf{wsk}, k)$: On input the watermarking secret key $\mathsf{wsk} = (k_{\mathsf{PRF}}, \mathsf{pp}, \mathsf{td})$, and a key $k \in \mathcal{K}_{\mathsf{EPRF}}$, the marking algorithm derives points $(x_1^*, \ldots, x_\lambda^*) \leftarrow \mathsf{PRF}(k_{\mathsf{PRF}}, k)$ and a punctured key $k' \leftarrow \mathsf{EX.Puncture}(\mathsf{pp}, k, (x_1^*, \ldots, x_\lambda^*))$. It outputs the circuit $C\colon \mathcal{X} \to \mathcal{Y}$ that implements the punctured evaluation algorithm $\mathsf{EX.PunctureEval}(\mathsf{pp}, k', \cdot)$.
- $\mathsf{Extract}(\mathsf{wsk}, C)$: On input the watermarking secret key $\mathsf{wsk} = (k_{\mathsf{PRF}}, \mathsf{pp}, \mathsf{td})$, and a circuit $C\colon \mathcal{X} \to \mathcal{Y}$, the extraction algorithm first extracts a key $k \leftarrow \mathsf{EX.Extract}(\mathsf{pp}, \mathsf{td}, C)$. If $k = \bot$, output UNMARKED. Otherwise, it computes $(x_1^*, \ldots, x_\lambda^*) \leftarrow \mathsf{PRF}(k_{\mathsf{PRF}}, k)$. If $C(x_i^*) \neq \mathsf{EX.Eval}(\mathsf{pp}, k, x_i^*)$ for all $i \in [\lambda]$, then output MARKED. Otherwise, output UNMARKED.

The underlying PRF family $\mathsf{F}\colon \mathcal{K}_{\mathsf{EPRF}} \times \mathcal{X} \to \mathcal{Y}$ (induced by the public parameters pp for the watermarking scheme) is defined as $\mathsf{F}(k, x) := \mathsf{Eval}(\mathsf{pp}, k, x)$ and $\mathsf{F.KeyGen}$ simply returns $\mathsf{EX.SampleKey}(\mathsf{pp})$. Note that the description of the PRF family F includes the public parameters pp, but not the other components in the watermarking secret key wsk.

In the full version of this paper [39], we show that assuming Π_{EPRF} is a puncturable extractable PRF and PRF is a secure pseudorandom function, then Π_{WM} from Construction 5.3 is a mark-embedding watermarking scheme that provides unremovability, unforgeability, and T-restricted pseudorandomness against the watermarking authority.

5.3 Watermarking Instantiations from Lattices

In this section, we summarize our main results on constructing new lattice-based watermarking schemes from our puncturable extractable PRF. We refer to the full version of this paper [39] for the full details of our constructions (as well as extensions to the main construction).

Corollary 5.4 (Message-Embedding Watermarking from Lattices). *Fix a security parameter λ. Take any $0 < \varepsilon < \delta < 1/2$ where $\delta > \varepsilon + 1/\mathsf{poly}(\lambda)$. Then, assuming it is difficult to approximate to worst-case lattice problems (e.g.,* GapSVP *or* SIVP*) with a* nearly polynomial *approximation factor, there exists a secret-key message-embedding watermarking scheme that satisfies ε-unremovability, a relaxed version of δ-unforgeability (see the full version of this paper [39]), and T-restricted pseudorandomness against the watermarking authority for $T = \lambda$. Assuming hardness of approximating worst-case lattice problems with a* sub-exponential *approximation factor, the resulting watermarking scheme satisfies the standard notion of δ-unforgeability. Moreover, under the same assumptions in the random oracle model, we obtain watermarking schemes that satisfy weak δ-unforgeability (and all of the other properties) that additionally supports public marking.*

Acknowledgments

We thank Willy Quach, Sina Shiehian, Daniel Wichs, and Giorgos Zirdelis for many insightful conversations. We thank the anonymous CRYPTO reviewers for helpful feedback on the presentation. This work was funded by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

1. S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, 2010.
2. S. Agrawal, D. M. Freeman, and V. Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, 2011.
3. M. Ajtai. Generating hard instances of the short basis problem. In *ICALP*, 1999.
4. J. Alwen and C. Peikert. Generating shorter bases for hard random lattices. In *STACS*, 2009.

5. F. Baldimtsi, A. Kiayias, and K. Samari. Watermarking public-key cryptographic functionalities and implementations. In *ISC*, 2017.
6. A. Banerjee, G. Fuchsbauer, C. Peikert, K. Pietrzak, and S. Stevens. Key-homomorphic constrained pseudorandom functions. In *TCC*, 2015.
7. A. Banerjee and C. Peikert. New and improved key-homomorphic pseudorandom functions. In *CRYPTO*, 2014.
8. A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, 2012.
9. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.
10. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2), 2012.
11. N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, 2013.
12. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, 2014.
13. D. Boneh, Y. Ishai, A. Sahai, and D. J. Wu. Lattice-based snargs and their application to more efficient obfuscation. In *EUROCRYPT*, 2017.
14. D. Boneh, S. Kim, and H. W. Montgomery. Private puncturable prfs from standard lattice assumptions. In *EUROCRYPT*, 2017.
15. D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan. Key homomorphic prfs and their applications. In *CRYPTO*, 2013.
16. D. Boneh, K. Lewi, and D. J. Wu. Constraining pseudorandom functions privately. In *PKC*, 2017.
17. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, 2007.
18. D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.
19. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.
20. Z. Brakerski, D. Cash, R. Tsabary, and H. Wee. Targeted homomorphic attribute-based encryption. In *TCC*, 2016.
21. Z. Brakerski, R. Tsabary, V. Vaikuntanathan, and H. Wee. Private constrained prfs (and more) from LWE. In *TCC*, 2017.
22. Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, pages 1–12, 2014.
23. Z. Brakerski and V. Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC*, 2015.
24. Z. Brakerski and V. Vaikuntanathan. Circuit-ABE from LWE: unbounded attributes and semi-adaptive security. In *CRYPTO*, 2016.
25. R. Canetti and Y. Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In *EUROCRYPT*, 2017.
26. Y. Chen, V. Vaikuntanathan, and H. Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In *CRYPTO*, 2018.
27. A. Cohen, J. Holmgren, R. Nishimaki, V. Vaikuntanathan, and D. Wichs. Watermarking cryptographic capabilities. In *STOC*, 2016.
28. R. Gay, P. Méaux, and H. Wee. Predicate encryption for multi-dimensional range queries from lattices. In *PKC*, 2015.

29. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.
30. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions (extended abstract). In *FOCS*, 1984.
31. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. In *STOC*, 2013.
32. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, 2015.
33. S. Gorbunov and D. Vinayagamurthy. Riding on asymmetry: Efficient ABE for branching programs. In *ASIACRYPT*, 2015.
34. R. Goyal, V. Koppula, and B. Waters. Lockable obfuscation. In *FOCS*, 2017.
35. N. Hopper, D. Molnar, and D. A. Wagner. From weak to strong watermarking. In *TCC*, 2007.
36. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, 2008.
37. A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS*, 2013.
38. S. Kim and D. J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In *CRYPTO*, 2017.
39. S. Kim and D. J. Wu. Watermarking prfs from lattices: Stronger security via extractable prfs. *IACR Cryptology ePrint Archive*, 2018:986, 2018.
40. V. Lyubashevsky and D. Wichs. Simple lattice trapdoor sampling from a broad class of distributions. In *PKC*, 2015.
41. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.
42. D. Naccache, A. Shamir, and J. P. Stern. How to copyright a function? In *PKC*, 1999.
43. R. Nishimaki. How to watermark cryptographic functions. In *EUROCRYPT*, 2013.
44. C. Peikert and S. Shiehian. Privately constraining and programming PRFs, the LWE way. In *PKC*, 2018.
45. W. Quach, D. Wichs, and G. Zirdelis. Watermarking PRFs under standard assumptions: Public marking and security with extraction queries. In *TCC*, 2018.
46. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.
47. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.
48. D. Wichs and G. Zirdelis. Obfuscating compute-and-compare programs under LWE. In *FOCS*, 2017.
49. R. Yang, M. H. Au, J. Lai, Q. Xu, and Z. Yu. Collusion resistant watermarking schemes for cryptographic functionalities. *IACR Cryptology ePrint Archive*, 2017.
50. R. Yang, M. H. Au, J. Lai, Q. Xu, and Z. Yu. Unforgeable watermarking schemes with public extraction. In *SCN*, 2018.
51. M. Yoshida and T. Fujiwara. Toward digital watermarking for cryptographic data. *IEICE Transactions*, 94-A(1), 2011.