# Beyond Software Watermarking: Traitor-Tracing for Pseudorandom Functions

Rishab Goyal[1*], Sam Kim[2†], Brent Waters[3,4‡], and David J. Wu[3§]

[1] MIT, Cambridge, MA
[2] Stanford University, Stanford, CA
[3] University of Texas at Austin, Austin, TX
[4] NTT Research, Sunnyvale, CA

**Abstract.** Software watermarking schemes allow a user to embed an identifier into a piece of code such that the resulting program is nearly functionally-equivalent to the original program, and yet, it is difficult to remove the identifier without destroying the functionality of the program. Such schemes are often considered for proving software ownership or for digital rights management. Existing constructions of watermarking have focused primarily on watermarking pseudorandom functions (PRFs).

In this work, we revisit the definitional foundations of watermarking, and begin by highlighting a major flaw in existing security notions. Existing security notions for watermarking only require that the identifier be successfully extracted from programs that preserve the *exact* input/output behavior of the original program. In the context of PRFs, this means that an adversary that constructs a program which computes a quarter of the output bits of the PRF or that is able to distinguish the outputs of the PRF from random are considered to be *outside* the threat model. However, in any application (e.g., watermarking a decryption device or an authentication token) that relies on PRF security, an adversary that manages to predict a quarter of the bits or distinguishes the PRF outputs from random would be considered to have defeated the scheme. Thus, existing watermarking schemes provide very little security guarantee against realistic adversaries. None of the existing constructions of watermarkable PRFs would be able to extract the identifier from a program that only outputs a quarter of the bits of the PRF or one that perfectly distinguishes.

*Email: goyal@utexas.edu. Part of this work was done while at UT Austin and the Simons Institute for the Theory of Computing. Research supported in part by an IBM PhD fellowship and the Simons-Berkeley research fellowship.

†Email: skim13@cs.stanford.edu. Part of this work was done at the Simons Institute for the Theory of Computing. Research supported by NSF, DARPA, a grant from ONR, and the Simons Foundation.

‡Email: bwaters@cs.utexas.edu. Research supported by NSF CNS-1908611, a Simons Investigator award, and a Packard Foundation Fellowship.

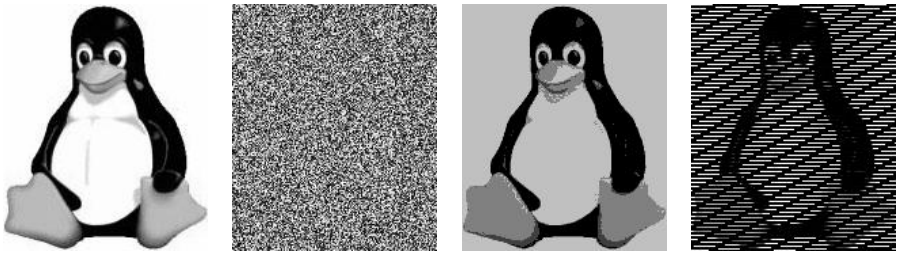
§Email: dwu4@cs.utexas.edu. Part of this work was done at the University of Virginia and while visiting the Simons Institute for the Theory of Computing. Research supported by NSF CNS-1917414, CNS-2045180, and a Microsoft Research Faculty Fellowship.

To address the shortcomings in existing watermarkable PRF definitions, we introduce a new primitive called a *traceable PRF*. Our definitions are inspired by similar definitions from public-key traitor tracing, and aim to capture a very robust set of adversaries: namely, any adversary that produces a useful distinguisher (i.e., a program that can break PRF security), can be traced to a specific identifier. We provide a general framework for constructing traceable PRFs via an intermediate primitive called *private linear constrained PRFs*. Finally, we show how to construct traceable PRFs from a similar set of assumptions previously used to realize software watermarking. Namely, we obtain a single-key traceable PRF from standard lattice assumptions and a fully collusion-resistant traceable PRF from indistinguishability obfuscation (together with injective one-way functions).

## 1 Introduction

Software watermarking is a mechanism for protecting against unauthorized redistribution of software. In a watermarking scheme, a user can embed some special information called a "mark" into a program such that the resulting program is nearly functionally-equivalent to the original one, and moreover, it is difficult for an adversary to remove the watermarking without destroying its input/output behavior. The majority of works studying cryptographic notions of watermarking have focused primarily on watermarking pseudorandom functions (PRFs) [CHN+16, BLW17, KW17, QWZ18, YAL+18, KW19, YAL+19, YAYX20]. Namely, the goal in each of these constructions is to embed an identifier (e.g., a user's name or a device id) into a PRF key such that (1) the marked key still preserves the input/output behavior of the original PRF; and (2) no efficient adversary is able to construct a key that both preserves the input/output behavior of the PRF on an $\varepsilon$-fraction of the domain and does not contain the identifier. The first requirement corresponds to "correctness" while the second corresponds to "unremovability."

*The limitations of existing definitions.* While these correctness and unremovability requirements seem to capture an intuitive notion of what we might desire from a watermarking scheme, they fall short of capturing meaningful notions of security in many realistic settings. For instance, take a watermarkable PRF that is secure under the above notions, and consider an adversary that takes a marked circuit $C\colon \{0,1\}^n \to \{0,1\}^n$ and outputs a circuit $C'$ that on input $x$, outputs the first $n/4$ bits of $C(x)$. Under existing definitions, mark-extraction is allowed to fail in this setting (since $C'$ does *not* preserve the input/output behavior of the marked program). At the same time, $C'$ still reveals substantial information about the original function and is often sufficient to compromise security of any cryptographic scheme that relies on the watermarked PRF. For instance, if the PRF is used to construct a symmetric encryption scheme, a circuit that outputs a quarter of the bits of the PRF completely breaks semantic security of the encryption scheme (see Fig. 1 for a visual example of this). However, even though



(a) Original (b) Encrypted (c) Recovered (pixel-level) (d) Recovered (block-level)

Fig. 1: Illustration of plaintext recovery using a PRF-based encryption scheme given a circuit that computes the leading $n/4$ bits of the PRF. Fig. 1a shows the original image and Fig. 1b shows the image encrypted using a PRF in counter mode. Figs. 1c and 1d shows the recovered image if the image is encrypted pixel-by-pixel and block-by-block, respectively, and the adversary has a circuit that computes the $n/4$ most significant bits of the PRF output.

$C'$ suffices to completely break semantic security of the encryption scheme, the watermarking scheme *cannot* recover the mark from the compromised key.

The above example highlights a limitation in existing security notions for cryptographic watermarking: namely, the existing definition only allows for a restrictive (and unrealistic) set of adversarial strategies. Indeed, *none* of the existing constructions [CHN+16, BLW17, KW17, QWZ18, YAL+18, KW19, YAL+19, YAYX20] of software watermarking remain secure if we expand the set of admissible adversarial strategies to include the simple example described above. Existing watermarking constructions all take the approach of carefully embedding the identifier in the output of the function. Such an embedding critically exploits of the assumption that the adversary must preserve much of the exact input/output behavior of the original function, in which case, most of the original outputs (that embed the identifier) are also preserved. Consequently, if the adversary constructs a circuit that does not exactly preserve the input/output behavior, then the tracing algorithm cannot recover the embedded identifier.[5]

In cryptography, it is not only prudent, but oftentimes, essential for applications, to design expressive threat models that enable the broadest range of adversaries. Indeed, the very first formal security notions [GM84, GGM84] in

[5] In some cases (e.g., [CHN+16]), the tracing algorithm can still *partially* recover the identity (e.g., a quarter of the bits) from a circuit that outputs a quarter the bits of each output. But this tracing algorithm can be defeated by an adversary which outputs a circuit that only distinguishes the output of the PRF (i.e., on input $(x, y)$, output 1 if $\mathsf{Eval}(\mathsf{msk}, x) = y$ and 0 otherwise) or a circuit that computes the parity of the bits of the PRF output.

cryptography carefully distinguished between the functionality requirements of a primitive and what the adversary would need to do to break it. In the case of semantic security [GM84], it sufficed for an adversary to *distinguish* between encryptions of two messages, and not that the adversary be able to *recover* the original message. In the case of PRFs [GGM84], it sufficed that the adversary could *distinguish* PRF evaluations from random as opposed to needing to *predict* the outputs of the PRF (and indeed, imposing such a restriction on the adversary would limit the usefulness of the primitive). In each of these examples, the adversary's objective is easier to achieve than emulating the exact functionality or semantic requirements of the primitive. This is the philosophy we take when designing our security definitions.

*This work.* Our primary goals in this work are to highlight the deficiencies of existing security notions for cryptographic watermarking and to introduce a new security framework that better models our intuitive notions of security for a watermarking scheme. Our definitions are inspired by similar notions developed in the literature on traitor tracing [CFN94, BSW06, NWZ16, GKRW18, GKW18] and recent work on watermarking public-key cryptographic primitives [GKM+19]. We begin by introducing a new notion of a *traceable PRF* which both suffices to instantiate the existing applications of watermarkable PRFs and addresses the limitations of existing watermarkable PRF definitions and offers meaningful security guarantees in realistic scenarios (e.g., they can be used to construct traceable symmetric encryption schemes). We then show how to construct non-collusion-resistant traceable PRFs from private constrained PRFs [BLW17] and fully collusion-resistant traceable PRFs from indistinguishability obfuscation [BGI+01]. We note that the assumptions needed to instantiate both of our schemes match the assumptions needed to instantiate watermarkable PRFs. This means that our new primitives can be instantiated from the same assumptions as watermarkable PRFs, and yet, provide much stronger security guarantees.

### 1.1 Our Results

Our first contribution is a new security definition that better captures the security goals in watermarkable PRFs. Here, we start from the beginning by re-examining the original motivation for building watermarkable PRFs. The original intent of watermarking PRFs is to be able to give a user a marked implementation of a PRF (e.g., for use in a symmetric encryption or authentication scheme) such that if the user later on tries to replicate the PRF functionality, there is a way to *trace* the replicated program back to the user's original key. The question is what constitutes a "valid" attempt at replicating the functionality. In this work, we consider any program that violates the security of the PRF (i.e., is able to distinguish PRF outputs from random) to be a "valid" attack. This definition is in part inspired by security definitions proposed in the setting of traitor tracing by Nishmaki et al. [NWZ16] (and adopted by later papers [GKRW18, GKW18]). In the earlier definitions starting with [CFN94], the tracing algorithm was only required to work against adversarial decoders that

could successfully decrypt and recover the original message in its entirety from a ciphertext. However, [NWZ16] observed that this definition can be too restrictive as it ruled out valid attacks that could extract *partial* information about the encrypted message (e.g., the first quarter of an encrypted video stream) or simply distinguished between different messages. Fortunately, most traitor tracing constructions developed under earlier definitions also remained secure under the strengthened definition. However, this does not appear to be the case for watermarkable PRFs.

*Our notion: traceable PRFs.* Since the functionality in this case is a PRF, a natural security notion is that if the adversary outputs any functionality that helps one break pseudorandomness (i.e., distinguish the outputs of the PRF from random), then it should be possible to trace the identity associated with the functionality. In this work, we require that the tracing algorithm succeeds against any distinguisher that can break *weak pseudorandomness* of the PRF. Specifically, any program that can distinguish the PRF outputs on *random* domain elements can be traced to one (or more) compromised keys. Observe that this not only captures adversarial strategies that preserve the exact input/output behavior of the PRF on an $\varepsilon$-fraction of the domain (as in the case of watermarking), but also the previous example of a program that outputs a quarter of the bits of the PRF. It also includes more general strategies such as a distinguisher circuit that outputs 1 if $(x, y)$ is an input/output pair of the PRF and 0 otherwise. Under our definition, no efficient adversary can remove a mark from the program unless it produces a program that does not break weak pseudorandomness of the PRF.

It is natural to ask whether we could trace the embedded mark from *any* PRF distinguisher, such as a distinguisher that can adaptively choose the inputs rather than only seeing evaluations at random points. While this may seem more natural, a closer inspection shows that it is unsatisfiable. This is because under this definition, we can construct an untraceable PRF distinguisher by simply hardwiring a *single* PRF input-output pair $(x, y)$ in the distinguisher. This distinguisher completely breaks pseudorandomness, but is untraceable as it contains no information about the PRF except a single input-output pair. It is crucial to observe that such a distinguisher is also useless for any adversarial applications of the PRF. This shows that the concept of traceability must be carefully defined to precisely capture the semantics of a "useful" distinguisher. We discuss this in more detail in Section 3.1. Our definition considers distinguishers for weak pseudorandomness, which means that the adversary's program necessarily contains information about the PRF on a noticeable fraction of the domain. We also note that this does *not* preclude a traceable PRF to satisfy pseudorandomness as a standalone primitive (and indeed, the constructions we propose in this work satisfy the usual notion of pseudorandomness). The restriction to distinguishers that break weak pseudorandomness is only in the definition of tracing security.

*Traceable PRF syntax.* A traceable PRF scheme consists of four algorithms: $\mathsf{Setup}$, $\mathsf{KeyGen}$, $\mathsf{Eval}$, and $\mathsf{Trace}$. The $\mathsf{Setup}$ algorithm samples a PRF key $\mathsf{msk}$ and a tracing key $\mathsf{tk}$ that is used for tracing. The key-generation algorithm takes

as input the PRF key $\mathsf{msk}$ and an identifier $\mathsf{id}$ and outputs a "marked" key $\mathsf{sk}_{\mathsf{id}}$. The evaluation algorithm $\mathsf{Eval}$ takes as input either the PRF key $\mathsf{msk}$ or an identity key $\mathsf{sk}_{\mathsf{id}}$ and implements PRF evaluation. We require that $\mathsf{Eval}(\mathsf{sk}_{\mathsf{id}}, \cdot)$ and $\mathsf{Eval}(\mathsf{msk}, \cdot)$ agree almost everywhere (i.e., on all but a negligible fraction of the domain). This property is the analog of the "correctness" or "functionality-preserving" property in the setting of watermarking schemes. Finally, there is a trace algorithm $\mathsf{Trace}$ that takes as input the tracing key $\mathsf{tk}$ and has *oracle* access to a distinguisher $D$, and outputs a set of compromised keys (if any). Our security requirement says that if the distinguisher $D$ is able to break *weak pseudorandomness* of the PRF (i.e., distinguish the outputs of $\mathsf{Eval}(\mathsf{msk}, \cdot)$ at random points from those of a random function), then the tracing algorithm must successfully identify a set of compromised keys used to construct $D$. Similar to the corresponding notions in traitor tracing (and watermarking), we can consider several variations of our basic schema and requirements:

- **Collusion-resistance:** We say that a traceable PRF is fully collusion-resistant if an adversary who has an arbitrary number of identity keys $S = \{\mathsf{sk}_{\mathsf{id}_1}, \ldots, \mathsf{sk}_{\mathsf{id}_k}\}$ still cannot construct a useful distinguisher $D$ where $\mathsf{Trace}^D(\mathsf{tk})$ does not output a non-empty set $T \subseteq S$.[6] We say that a scheme satisfies bounded (resp., $Q$-key) collusion resistance if security only holds against adversaries that compromise an a priori bounded number of keys (resp., at most $Q$ keys). In this work, we show how to construct a single-key traceable PRF from standard lattice assumptions[7] and a fully collusion resistant traceable PRF from indistinguishability obfuscation (and injective one-way functions).
- **Public tracing vs. secret tracing:** We say that a traceable PRF supports public tracing if security holds even if the tracing key $\mathsf{tk}$ is public. Otherwise, we say the traceable PRF is in the secret tracing setting. Our basic single-key traceable PRF from lattices is secure in the secret-tracing setting, while our obfuscation-based construction is secure in the public-tracing setting.

We provide the full definition in Section 4.1.

*Constructing traceable PRFs.* To construct traceable PRFs, we introduce an intermediate primitive of a *private linear constrained PRF*. This primitive can be viewed as a symmetric analog of a "private linear broadcast encryption" (PLBE) from [BSW06] and which has featured prominently in a number of subsequent

[6] We cannot stipulate that $T = S$ since the adversary might not use every compromised key when constructing the distinguisher $D$. The tracing algorithm can only recover the keys the adversary actually uses.

[7] A traceable PRF bears many similarities with a *constrained* PRF [BW13, KPTZ13, BGI14], and all known constructions of collusion-resistant constrained PRFs for sufficiently complex constraints from standard lattice assumptions are secure only in the single-key setting [BV15]. Fully collusion-resistance constrained PRFs for general constraints are only known from indistinguishability obfuscation [BZ14] and one-way functions. Recent work has shown how to construct indistinguishability obfuscation from the combination of *multiple* standard assumptions [JLS21].

traitor tracing constructions [GKSW10, NWZ16, GKW18]. First, recall that in a constrained PRF [BW13, KPTZ13, BGI14], the holder of the PRF master secret key $\mathsf{msk}$ can issue a constrained key $\mathsf{sk}_f$ for a constraint $f$ such that the constrained key can be used to evaluate on only the inputs $x$ that satisfy the constraint (i.e., the inputs $x$ where $f(x) = 1$). Moreover, the value of the PRF at points $x$ where $f(x) = 0$ remain pseudorandom even given $\mathsf{sk}_f$.

A private linear constrained PRF is similar in spirit to a constrained PRF for a class of linear constraints.[8] In this case, the constrained keys are each associated with a $\kappa$-bit index $\mathsf{id} \in [0, 2^\kappa - 1]$. Every input in the domain is associated with a *private* index $t \in [0, 2^\kappa]$ and a constrained key for index $\mathsf{id}$ can be used to evaluate the PRF on all inputs whose index $t \leq \mathsf{id}$. In addition to the usual $\mathsf{Setup}$ (for sampling the PRF key), $\mathsf{KeyGen}$ (for issuing constrained keys), and $\mathsf{Eval}$ (for evaluating the PRF), there is a fourth algorithm $\mathsf{Samp}$ that is used to sample domain elements with a given index $t$ together with the PRF evaluation at the sampled point. The sampling algorithm $\mathsf{Samp}$ can either be public-key algorithm (in which case we obtain a publicly-traceable PRF) or a secret-key algorithm (in which case we obtain a secretly-traceable PRF). Similar to a PLBE scheme, there are three main security requirements we require on a private linear constrained PRF:

- **Normal hiding:** A random domain element is computationally indistinguishable from a randomly-sampled domain element with index 0 (output by $\mathsf{Samp}$), even given any collection of identity keys.
- **Identity hiding:** A randomly-sampled domain element with index $i$ is computationally indistinguishable from a randomly-sampled domain element with index $j$, provided that the adversary does not have any identity keys for an index $\mathsf{id} \in [i, j-1]$.
- **Pseudorandomness:** The PRF evaluations on randomly-sampled domain elements with index $2^\kappa$ are computationally indistinguishable from uniform given any collection of identity keys.

Given a private linear constrained PRF satisfying the above properties, we can construct a traceable PRF using a similar type of transformation used to construct traitor tracing from PLBE. Namely, we can reduce the tracing problem to a "jump-finding" problem as follows. Let $D$ be the decoder constructed by the adversary. By assumption, we assume that $D$ is useful: namely, it breaks weak pseudorandomness of the encryption scheme. This means that $D$ is able to distinguish the PRF evaluation at a randomly-sampled domain element from a uniformly random value with non-negligible advantage $\varepsilon$. By the normal hiding property, $D$ must also have advantage $\varepsilon$ when distinguishing evaluations at randomly-sampled points with index 0. Next, by the pseudorandomness property, the distinguishing advantage of $D$ for randomly-sampled points with index $2^\kappa$ must be

[8] As we describe more formally below, the "privacy" requirement refers to a property on the inputs to the PRF, and *not* the notion of constraint-privacy in the standard definition of a "private constrained PRF" from [BLW17].

negligible. Thus, there must be a "jump" in the decoder's distinguishing advantage on domain elements on some index $0 < t < 2^\kappa$. By the identity-hiding property, such "jumps" can only occur on indices for which the adversary possesses an identity key. We can then identify these jumps (and correspondingly, the set of compromised keys) by either performing a linear scan over the identity space (when the identity space is polynomial) [BSW06, GKSW10, GKW18] or by using a jump-finding algorithm (when the identity space is exponential) [NWZ16].

Due to some technical differences between PLBE and private linear constrained PRFs, we actually have to run the tracing algorithm *twice* in our construction. Very briefly, the reason behind this requirement is that, unlike encryption systems where the distinguisher just receives a single ciphertext and has to output its guess, the distinguisher in the case of traceable PRFs receives a tuple consisting of *both* a random domain element together with its evaluation. Here, the distinguisher may stop working if it notices the tracer is changing the distribution used to sample the inputs (i.e., domain elements). This means that if the tracer only performs a single scan, such decoders may evade detection. Thus, we need to apply the underlying tracing algorithm twice to circumvent this issue. In the first scan, the tracer runs the scan with a consistent output distribution, and then it performs a second scan where the output distribution is random and essentially independent of the input distribution. We provide the full technical details in Section 4.2.

*Constructing private linear constrained PRFs.* In this work, we describe two constructions of private linear constrained PRFs. The first construction gives a single-key private linear constrained PRF in the secret-tracing setting and can be instantiated from LWE while the second construction is a collusion-resistant private linear constrained PRF in the public-tracing setting. Interestingly, both of our constructions rely on a similar set of building blocks as those used for watermarkable PRFs. We give a high-level sketch of our main constructions here:

- **Single-key private linear constrained PRF.** Our first construction combines a private constrained PRF together with an authenticated encryption scheme with pseudorandom ciphertexts (such authentication encryption schemes can be based on one-way functions). Recall first that a private constrained PRF is a constrained PRF where the constrained key $\mathsf{sk}_f$ hides the associated constraint function $f$.

  Let $\ell$ be the bit-length of the ciphertexts in the authenticated encryption scheme, and let $\mathsf{sk}$ be the secret key of the authenticated encryption scheme. The domain of our PRF will be $\{0,1\}^\ell$, and a point with index $t \in [0, 2^\kappa]$ will be an authenticated encryption of $t$. A constrained key for an identity $\mathsf{id}$ consists of a private constrained key for the function $f_{\mathsf{sk},\mathsf{id}}$ where $f_{\mathsf{sk},\mathsf{id}}(x) = 0$ whenever $x$ is a valid encryption under $\mathsf{sk}$ of some index $t' > \mathsf{id}$, and is 1 otherwise. The (secret-key) sampling algorithm will first encrypt the target index $t$ under $\mathsf{sk}$ and output the resulting ciphertext $\mathsf{ct}_t$ together with the PRF evaluation at $\mathsf{ct}_t$.

At a high-level, the security proof relies on the fact that a private constrained PRF hides the constraint function, which in this particular case, means that it hides the secret key $\mathsf{sk}$. Then, normal hiding and identity hiding follows from the fact that ciphertexts are pseudorandom, and pseudorandomness follows from constrained security of the underlying constrained PRF. We give the full description and analysis in Section 5.

– **Collusion-resistant private linear constrained PRF with public-tracing.** Our second construction gives a fully collusion resistant private linear constrained PRF that supports public tracing from indistinguishability obfuscation and injective one-way functions. By the recent breakthrough work of Jain et al. [JLS21], both assumptions hold assuming the existence of a PRG in $\mathsf{NC}^0$ together with the $\mathsf{LWE}$, $\mathsf{LPN}$, and $\mathsf{SXDH}$ assumptions. The high-level idea is very similar to our secret-key scheme above. Namely, the domain elements are ciphertexts in a (puncturable) public-key encryption scheme [CHN+16], and the identity keys consist of an obfuscated program with the decryption key hard-wired within it. To publicly sample inputs/outputs of the PRF (needed for public tracing), we provide an obfuscated program with a (puncturable) PRF key hard-wired within. We provide the details and analysis in Section 6.

*An application: secret-key traitor tracing.* We note that our notion of traceable PRFs lends itself naturally to a secret-key traitor tracing scheme. For instance, we can take our encryption scheme to be standard nonce-based encryption with a PRF (i.e., to encrypt a message $m$, sample a random $r \leftarrow \{0,1\}^n$ and compute the ciphertext $\mathsf{ct} = (r, m \oplus \mathsf{PRF}(k, r))$. If we instantiate the underlying PRF with a traceable PRF, then the resulting scheme immediately gives a traitor tracing scheme. Namely, any decoder that is able to distinguish between the encryption of two messages $m_0$ and $m_1$ also necessarily is able to distinguish $\mathsf{PRF}(k, r)$ from uniformly random for a random choice of $r \leftarrow \{0,1\}^n$. The claim then follows by tracing security. We stress here that a similar notion would *not* follow if we replace $\mathsf{PRF}$ with a watermarkable PRF. Here, it is not clear how to translate a decoder $D$ that is only able to distinguish between encryptions of two messages into an algorithm that is able to recover the full input/output behavior of the PRF on a noticeable fraction of the domain.

*Comparison with watermarkable PRFs.* One distinction between traceable PRFs and watermarkable PRFs is that in our definition of a traceable PRF, the tracing key is sampled jointly with the PRF key. In classic definitions of watermarking, it is possible to have a single (fixed) tracing key for an entire family of PRFs. This means that it is possible to sample a PRF key and decide to mark it at a later point in time. As we discuss in Section 3.1, having a tracing key that depends on the PRF key is essential to realizing the strong security notions in a traceable PRF. In most practical scenarios, if one wanted to take advantage of watermarking for software protection, it seems reasonable for them to sample the PRF key together with the marked key(s). Thus this distinction does not seem significant in practice, and we believe that the stronger and meaningful security

notions achieved by traceable PRFs makes it a far more suitable primitive than a watermarkable PRF in any realistic environment.

### 1.2 Related Work

*Watermarking.* Barak et al. [BGI+01, BGI+12] and Hopper et al. [HMW07] introduced the first rigorous mathematical frameworks for software watermarking that considered *arbitrary* adversarial strategies (i.e., the adversary is allowed to output an arbitrary circuit that preserves the input-output behavior of the original program). Cohen et al. [CHN+16] provided the first construction of a watermarking scheme for PRFs using indistinguishability obfuscation. Earlier works on watermarking [NSS99, YF11, Nis13] imposed additional restrictions on the adversary's capabilities. Several works have also studied watermarking public-key cryptographic primitives [CHN+16, BKS17, GKM+19, Nis20]. Here, the work of Goyal et al. [GKM+19] expanded watermarking security definitions (in the public-key setting) to include adversaries that are able to break the "semantics" of a scheme (as opposed to just the set of adversaries that preserve exact input/output behavior).

*Traitor tracing.* The notion of traitor tracing was first proposed by Chor et al. [CFN94] for solving the piracy problem in broadcast systems. Since then, numerous relaxations have been studied in order to achieve short ciphertexts. Broadly these can be categorized as follows: schemes where the traceability guarantees hold as long as the adversary corrupts an a priori bounded number of users and schemes where the guarantees hold as long as the adversary's decoder succeeds with probability greater than an a priori threshold. The former relaxation leads to traitor tracing schemes in the bounded collusion setting where we have numerous constructions via combinatorial tools [CFN94, SW98, CFNP00, SSW01, PST06, BP08] as well as a variety of cryptographic assumptions [KD98, BF99, KY02a, KY02b, CPP05, ADM+07, FNP07, LPSS14, NWZ16, ABP+17]. The latter schemes are typically referred to as "threshold traitor tracing" [NP98, CFNP00, BN08]. In another line of work, [GKRW18] considered schemes with a relaxed tracing guarantee: namely, the tracing algorithm does not need to be succeed in all cases. Recently, there has been significant progress on constructing fully collusion-resistant compact traitor tracing schemes from standard lattice assumptions [GKW18, GKW19a]. Since then, a sequence of works has built new traitor tracing systems with more functionality from standard cryptographic assumptions [CVW+18, GQWW19, GKW19b, KW20].

## 2 Preliminaries

We write PPT to denote probabilistic polynomial-time. We denote the set of all positive integers up to $n$ as $[n] \coloneqq \{1, \ldots, n\}$. Throughout this paper, unless specified otherwise, all polynomials we consider are positive polynomials. For any finite set $S$, $x \leftarrow S$ denotes a uniformly random element $x$ from the set

$S$. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element $x$ drawn from distribution $\mathcal{D}$. The distribution $\mathcal{D}^n$ is used to represent a distribution over vectors of $n$ components, where each component is drawn independently from the distribution $\mathcal{D}$.

For (possibly randomized) algorithms $\mathcal{A}$ and $D$, we use the notation $A^D$ to denote that algorithm $\mathcal{A}$ has oracle access to algorithm $D$. Here, if the algorithm $D$ is stateless, then on each query made by $\mathcal{A}$ to $D$, the oracle responds with a randomly drawn sample from the corresponding output distribution. If the algorithm $D$ is stateful, then whenever $\mathcal{A}$ queries the oracle $D$, it can choose to either suspend the current execution of the oracle $D$ or to continue executing $D$ while it maintains its state. If $D$ terminates after receiving an input, then it sends the final output of the computation as its query response to $\mathcal{A}$.

*Pseudorandom generator.* A pseudorandom generator $\mathsf{PRG}\colon \{0,1\}^\lambda \to \{0,1\}^\ell$ is secure if for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\mathcal{A}(t_b) = b \ : \ \begin{array}{c} s \leftarrow \{0,1\}^\lambda, t_0 \leftarrow \mathsf{PRG}(s) \\ t_1 \leftarrow \{0,1\}^\ell, b \leftarrow \{0,1\} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

## 3 Defining Traceable PRFs

In this section, we formally introduce our notion of a *traceable* PRF.

*Syntax.* A traceable PRF scheme, with input-output space $\mathcal{X} = \{\mathcal{X}_{\lambda,\kappa}\}_{\lambda,\kappa\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_{\lambda,\kappa}\}_{\lambda,\kappa\in\mathbb{N}}$[9], consists of the following four algorithms:

$\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{msk}, \mathsf{tk})$. The setup algorithm takes as input the security parameter $\lambda$, the "'identity space" parameter $\kappa$, and outputs a master PRF key $\mathsf{msk}$ and a tracing key $\mathsf{tk}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$. The key generation algorithm takes as input the master key and an identity $\mathsf{id} \in \{0,1\}^\kappa$. It outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.

$\mathsf{Eval}(\mathsf{sk}, x) \to y$. The decryption algorithm takes as input a secret key $\mathsf{sk}$ (which could be the master key), input $x \in \mathcal{X}$, and outputs $y \in \mathcal{Y}$.

$\mathsf{Trace}^D(\mathsf{tk}, 1^z) \to T \subseteq \{0,1\}^\kappa$. The tracing algorithm has oracle access to a program $D$, it takes as input the tracing key $\mathsf{tk}$, parameter $z$, and it outputs a set $T$ of identities.

*Weak pseudorandomness.* Below we define the weak pseudorandomness property for traceable PRFs.

[9]Throughout the paper, we drop the dependence of spaces $\mathcal{X}_{\lambda,\kappa}$ and $\mathcal{Y}_{\lambda,\kappa}$ on security parameter $\lambda$ and identity length parameter $\kappa$ whenever clear from context.

**Definition 3.1 (Weak pseudorandomness).** *A traceable PRF scheme* $\mathsf{Tr\text{-}PRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ *satisfies weak pseudorandomness property if for every stateful PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, the following holds*

$$\Pr\left[\mathcal{A}^{O_b(\mathsf{msk})} = b : \begin{array}{c} 1^\kappa \leftarrow \mathcal{A}(1^\lambda), b \leftarrow \{0,1\} \\ (\mathsf{msk}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

*where the oracle* $O_b(\mathsf{msk})$ *is defined as follows: if* $b = 0$*, then on each evaluation query made by adversary* $\mathcal{A}$*, the oracle samples random input* $x \leftarrow \mathcal{X}$ *and sends* $(x, \mathsf{Eval}(\mathsf{msk}, x))$ *to* $\mathcal{A}$*; otherwise, if* $b = 1$*, then on each evaluation query made by adversary* $\mathcal{A}$*, the oracle samples random input* $x \leftarrow \mathcal{X}$ *and sends* $(x, f(x))$ *to* $\mathcal{A}$ *where* $f : \mathcal{X} \to \mathcal{Y}$ *is a random function.*[10]

*Key-similarity property.* Informally, the key-similarity property says that the marked key is functionally equivalent to the original unmarked key on all but a negligible fraction of inputs. Formally, we define the property as follows:

**Definition 3.2 (Key similarity).** *A traceable PRF scheme* $\mathsf{Tr\text{-}PRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ *satisfies key-similarity if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda, \kappa \in \mathbb{N}$*, identity* $\mathsf{id} \in \{0,1\}^\kappa$*,* $(\mathsf{msk}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa)$*, the following holds*

$$\Pr\left[\mathsf{Eval}(\mathsf{msk}, x) \neq \mathsf{Eval}(\mathsf{sk}_{\mathsf{id}}, x) : \begin{array}{c} \mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \\ x \leftarrow \mathcal{X} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We note that while the marked keys agree with the unmarked key almost everywhere, it may still be easy for an adversary to efficiently find a point on which they differ. Thus, we can consider a strengthening of the property called "key indistinguishability" which we introduce next.

*Key-indistinguishability property.* Informally, the key-indistinguishability property states that it should be hard for any PPT adversary to *find* inputs where the marked key (for an identity of the adversary's choosing) disagrees with the unmarked key. Formally, we define key indistinguishability as follows:

**Definition 3.3 (Key indistinguishability).** *A traceable PRF scheme* $\mathsf{Tr\text{-}PRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ *satisfies key indistinguishability if for every stateful PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, the following holds*

$$\Pr\left[\mathsf{Eval}(\mathsf{msk}, x) \neq \mathsf{Eval}(\mathsf{sk}^*, x) : \begin{array}{c} 1^\kappa \leftarrow \mathcal{A}(1^\lambda), (\mathsf{msk}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \\ (x, \mathsf{idx}, \mathsf{id}^*) \leftarrow \mathcal{A}^{\mathsf{Eval}(\mathsf{msk}, \cdot), \mathsf{KeyGen}(\mathsf{msk}, \cdot)} \end{array}\right] \leq \mathsf{negl}(\lambda),$$

[10] Note that instead of actually sampling a random function, the challenger simulates it by sampling random input-output pairs on the fly and storing them in a table.

*where* $\mathsf{sk}^*$ *is defined as*

$$\mathsf{sk}^* = \begin{cases} \mathsf{sk}^{(\mathsf{idx})} & \text{if } \mathsf{idx} \neq \bot \\ \mathsf{sk}_{\mathsf{id}^*} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}^*) & \text{otherwise,} \end{cases}$$

*and* $\mathsf{sk}^{(\ell)}$ *denotes the* $\ell^{th}$ *key* $\mathcal{A}$ *submits to the key-generation oracle.*

*Remark 3.4 (Key similarity vs. key indistinguishability).* It is easy to see that key indistinguishability is a *strictly stronger* property than key similarity. As a result, this property is only achievable in the *secret-tracing* setting. As we define more formally below, our tracing algorithm only has *oracle* access to the adversary's distinguishing circuit. If this tracing algorithm can be run publicly, then it must be the case that the tracing algorithm must be able to efficiently find *some* input where the unmarked key and the marked key differ. Otherwise, it cannot distinguish between the two keys given just oracle access to the evaluation algorithm.

In the full version of this paper [GKWW20], we also describe a weaker notion of key indistinguishability that we consider in some of our constructions.

*Secure tracing.* The secure tracing property states that if any PPT adversary creates a successful PRF distinguisher with respect to a master key, then the tracing algorithm, when provided with the PRF distinguisher, outputs the identity of at least one corrupted secret key, while never outputting the identity of an uncorrupt secret key. We define secure tracing as follows:

**Definition 3.5 (Secure tracing).** *Let* $\mathsf{Tr\text{-}PRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ *be a traceable PRF scheme. For any nonnegligible function* $\varepsilon(\cdot)$*, polynomial* $p(\cdot)$ *and PPT adversary* $\mathcal{A}$*, we define the tracing experiment* $\mathsf{ExptTPRF}^{\mathsf{Tr\text{-}PRF}}_{\mathcal{A},\varepsilon}(\lambda)$ *in Fig. 2. Based on* $\mathsf{ExptTPRF}^{\mathsf{Tr\text{-}PRF}}_{\mathcal{A},\varepsilon}$*, we define the following set of (probabilistic) events and their corresponding probabilities (which are a functions of* $\lambda$ *and parameterized by* $\mathcal{A}, \varepsilon$*):*

- $\mathsf{Good\text{-}Dis}$ *:* $\Pr\left[D^{O_b(\mathsf{msk})}(1^\lambda) = b \ : \ b \leftarrow \{0,1\}\right] \geq \frac{1}{2} + \varepsilon(\lambda)$*,*
  *where the probability is taken over the coins of* $D$*, and oracle* $O_b(\mathsf{msk})$ *is exactly as defined in Definition 3.1.*
  *Intuitively, this says that a distinguisher* $D$ *is an* $\varepsilon$*-good distinguisher if* $D$ *can break weak pseudorandomness of the underlying PRF with advantage* $\varepsilon = \varepsilon(\lambda)$*.*
  $\Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\varepsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Dis}]$*.*
- $\mathsf{Cor\text{-}Tr}$ *:* $T \neq \varnothing \wedge T \subseteq S_{\mathcal{ID}}$
  *This event corresponds to the tracing algorithm successfully outputting one or more of the keys the adversary possesses.*
  $\Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\varepsilon}(\lambda) = \Pr[\mathsf{Cor\text{-}Tr}]$*.*

**Experiment** $\mathsf{ExptTPRF}^{\mathsf{Tr\text{-}PRF}}_{\mathcal{A},\varepsilon}(\lambda)$

- $1^\kappa \leftarrow \mathcal{A}(1^\lambda)$.
- $(\mathsf{msk}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa)$.
- $D \leftarrow \mathcal{A}^{\mathsf{Eval}(\mathsf{msk},\cdot),\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SplEval}(\mathsf{msk},\cdot,\cdot)}$.
- $T \leftarrow \mathsf{Trace}^D(\mathsf{tk}, 1^{1/\varepsilon(\lambda)})$.

Let $S_{\mathcal{ID}}$ be the set of identities queried by $\mathcal{A}$ to the key generation oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$. Here, $\mathsf{SplEval}$ denotes a *special evaluation* algorithm that is defined as a randomized oracle algorithm that has $\mathsf{msk}$ hardwired, takes as input an identity $\mathsf{id} \in \{0,1\}^\kappa$, a string $x \in \mathcal{X}$, and outputs $y = \mathsf{Eval}(\mathsf{sk}_{\mathsf{id}}, x)$ where $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$. We discuss the rationale for this oracle in Remark 3.7.

Fig. 2: Experiment $\mathsf{ExptTPRF}$

- $\mathsf{Fal\text{-}Tr}$ : $T \not\subseteq S_{\mathcal{ID}}$
  *This event corresponds to the tracing algorithm outputting a key that the adversary did not request (i.e., falsely implicating an honest user).*
  $\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\varepsilon}(\lambda) = \Pr[\mathsf{Fal\text{-}Tr}]$.

*A traceable PRF scheme* $\mathsf{Tr\text{-}PRF}$ *(with secret-key tracing) is said to satisfy secure tracing property if for every PPT adversary* $\mathcal{A}$*, polynomial* $q(\cdot)$*, and non-negligible function* $\varepsilon(\cdot)$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *satisfying* $\varepsilon(\lambda) > 1/q(\lambda)$*, the following two properties hold:*

$$\Pr\text{-}\mathsf{Fal\text{-}Tr}_{\mathcal{A},\varepsilon}(\lambda) \leq \mathsf{negl}(\lambda) \text{ and } \Pr\text{-}\mathsf{Cor\text{-}Tr}_{\mathcal{A},\varepsilon}(\lambda) \geq \Pr\text{-}\mathsf{G\text{-}D}_{\mathcal{A},\varepsilon}(\lambda) - \mathsf{negl}(\lambda).$$

*Intuitively, the first property states that the tracing algorithm cannot falsely implicate an honest user with non-negligible probability and the second property requires that whenever* $D$ *is a* $\varepsilon$*-good distinguisher, then the tracing algorithm correctly traces at least one corrupt user.*

*Remark 3.6 (Security for publicly-traceable PRFs).* A traceable PRF scheme with public-tracing is defined identically to its secret-tracing counterpart, except now the adversary is additionally provided the tracing key $\mathsf{tk}$ in all of the security games. In the public-tracing setting, we require the scheme to satisfy weak pseudorandomness, key similarity, and the secure public tracing property (but not key indistinguishability; see Remark 3.4).

*Remark 3.7 (Special evaluation oracle* $\mathsf{SplEval}$*).* In our tracing experiment, we allow an attacker to not only corrupt keys for different users, but also query for PRF evaluations under keys of *non-corrupt* users on inputs of the adversary's choosing. Although providing access to this "special evaluation" oracle $\mathsf{SplEval}$ is not necessary for applications of traceable PRFs to traitor tracing systems, we include this as part of our definition to cover a broader class of adversarial strategies. For instance, this definition captures adversaries that may passively observe interactions between honest users using their respective identity keys.

Our definition says that even if the adversary can see (polynomially-many) such evaluations, they cannot construct a distinguisher that evades the tracing algorithm (nor can they cause the tracing algorithm to implicate one of the honest users). Thus, by allowing the adversary access to such an oracle, the definition provides security even against these much powerful adversaries.

Also, one could possibly have a seemingly stronger mechanism for capturing the special evaluation oracle where now it will be a stateful oracle and the adversary can ask the oracle to either sample (and store) a fresh key followed by evaluation with respect to the sampled key, or answer a evaluation query with respect to a previously-sampled key. Although this might seem like a stronger definition, this is not necessary since we can always assume without loss of generality that key generation algorithm is deterministic (by using a standard PRF for derandomization).

### 3.1 Note on Weak Pseudorandomness and Other Definitional Choices

In this section, we briefly discuss and motivate the definitional choices for our traceable PRF notion.

*On weak pseudorandomness.* In our definitional framework above, we focus on *weak pseudorandomness* as the target for both PRF security as well as the class of distinguishers against which we provide the tracing guarantee. There are a few technical reasons for the above choice. First, observe that it is impossible for a traceable PRF to be a secure PRF in the standard sense (i.e., appear pseudorandom on adversarially-chosen inputs) while also providing tracing guarantees against distinguishers that only break pseudorandomness in the standard sense. This is because in such a scenario, the adversary can construct an untraceable distinguisher by simply hardwiring a *single* PRF input-output pair (on a random input) and use that to claim that it is a valid distinguisher. Such a distinguisher can break the standard PRF game with advantage close to 1 by querying on its hard-wired input, and yet, no tracing algorithm can succeed here (since with overwhelming probability, the single input-output pair chosen by the adversary coincides with the real PRF evaluation on that input, and thus, cannot contain any information about an embedded identity).

Another possibility could be to allow the distinguisher to make arbitrary evaluation queries to the PRF, but the challenge point would still be chosen randomly. While this is a meaningful notion, this causes problems when defining publicly-traceable PRFs. Under this definition, the tracing algorithm would need the actual code of the distinguisher, as opposed to only requiring *oracle access* to the distinguisher. This is because under this definition, the tracing algorithm would need a way to respond to the distinguisher's evaluation queries (in order to use the distinguisher at tracing time). But if the distinguisher can make arbitrary PRF evaluation queries that the *public* tracing algorithm can answer, then the tracing algorithm can be used to break pseudorandomness. Consequently, this model is only achievable in the setting where the public tracing algorithm

has access to the code of the distinguisher. In this work, we focus on settings where tracing can be done just given *black-box* access to the decoder. Note that if we restrict ourselves to weak pseudorandomness, then there is no inherent contradiction; namely, a public tracing key only needs to provide an ability to sample random input-output evaluations of the PRF. Using indistinguishability obfuscation, we can realize this by publishing an obfuscated program that can sample input-output evaluations from a *sparse* pseudorandom subset of the domain and which does not compromise standard pseudorandomness.

A third possibility is to consider secure tracing against distinguishers which only break weak pseudorandomness, while requiring the PRF to achieve (regular) pseudorandomness security. Although this is not impossible (and our current constructions can be shown to satisfy this property), we decided to simply consider weak pseudorandomness security for PRF security since that yields a *unified* definitional framework and sufficed for many applications. Basically our intuition here is to avoid *unevenness* between the target pseudorandomness security for PRF security and the class of distinguishers against which we provide secure traceability.

*Joint sampling of tracing and PRF keys.* Lastly, our definitions assume that the tracing key is generated together with a PRF master key (via the Setup algorithm). That is, each PRF key is associated with a specific tracing key. An alternate definition could be to sample a *single* tracing key during the system setup, and then PRF keys could be sampled independent of the tracing key. This is the setting encountered in the context of watermarking PRFs [CHN+16]. However PRFs are a *symmetric-key* primitive. This means that in this scenario, the tracing algorithm would either need a description of the master PRF key to run the tracing algorithm, or the PRF setup must non-trivially depend on the tracing key itself. In the former case, it seems very restrictive since now the tracing party needs to know the full master key which may not be accessible in most applications. As to the latter case, it is not clear whether it provides any more functionality compared to our current definition. Therefore, we decided to consider a single joint setup for sampling the master PRF key as well as the tracing parameters as done in prior works on traitor tracing for public-key encryption systems [BSW06, NWZ16, GKRW18, GKW19a].

## 4 Traceable PRFs via Private Linear Constrained PRFs

In this section, we introduce an intermediate abstraction, that we call private linear constrained PRFs (PLCPRFs), towards building a traceable PRF scheme. This primitive mirrors the notion of private linear broadcast encryption (PLBE) [BSW06] from traitor tracing literature where PLBE was used as a useful abstraction for building general traitor tracing systems. We first present the syntax and security definitions for PLCPRFs, and later show that PLCPRFs lead to traceable PRFs.

### 4.1 Defining Private Linear CPRFs

*Syntax.* A private linear CPRF scheme with input-output space $\mathcal{X} = \{\mathcal{X}_{\lambda,\kappa}\}_{\lambda,\kappa\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_{\lambda,\kappa}\}_{\lambda,\kappa\in\mathbb{N}}$[11] consists of the following four algorithms:

$\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{msk}, \mathsf{tk})$. The setup algorithm takes as input the security parameter $\lambda$, the "identity space" parameter $\kappa$, and outputs a master PRF key $\mathsf{msk}$ and a tracing key $\mathsf{tk}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$. The key generation algorithm takes as input the master key and an identity $\mathsf{id} \in \{0,1\}^\kappa$. It outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.[12]

$\mathsf{Eval}(\mathsf{sk}, x) \to y$. The decryption algorithm takes as input a secret key $\mathsf{sk}$ (which could be the master key), input $x \in \mathcal{X}$, and outputs $y \in \mathcal{Y}$.

$\mathsf{Samp}(\mathsf{tk}, t) \to (x, y)$. The sampling algorithm takes as input the tracing key $\mathsf{tk}$, a threshold $t \in [0, 2^\kappa]$, and it outputs a input-output pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$.

*Key similarity and key indistinguishability.* We define key similarity and key indistinguishability for PLCPRFs to be identical to that for traceable PRFs as in Definitions 3.2 and 3.3.

*Security.* We now introduce some useful security properties for PLCPRFs that will be useful for constructing traceable PRFs. Intuitively, the properties can be stated as follows. The first property is called the *normal hiding* property which states that for any PPT adversary, it should be hard to distinguish whether an input string $x$ is sampled uniformly at random from the full domain $\mathcal{X}$, or if it is sampled uniformly at random using the sample algorithm for threshold 0 (that is, as $(x, y) \leftarrow \mathsf{Samp}(\mathsf{tk}, 0)$). The second property is called the *identity hiding* property which states that an input string $x$ should also hide the threshold $t$ corresponding to which it is sampled as long as the adversary cannot trivially learn it by simply evaluating at $x$ using its secret keys. Lastly, we define the *pseudorandomness* property which states that the PRF output on input strings sampled corresponding to threshold $2^\kappa$ are pseudorandom. Formally, we define each notion similar to the corresponding set of PLBE definitions from [BSW06, GKW18, GKW19a]. However, in our setting, we allow the adversary to make an *a priori unbounded* number of oracle queries, which will be essential to our construction of traceable PRFs from PLCPRFs. In the public-key setting, handling a single encryption query is sufficient to construct traitor tracing.

**Definition 4.1 (Normal hiding).** *A PLCPRF scheme is said to satisfy normal hiding if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible func-*

[11] As mentioned previously, we drop the dependence on $\lambda, \kappa$ whenever clear from context.

[12] This could also be viewed as a "constrain" algorithm (in the language of constrained PRFs [BW13, KPTZ13, BGI14]), but there are some semantic differences. As such, we refer to this algorithm as a "key-generation" algorithm instead.

*tion* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*, the following holds:*

$$\Pr\left[\mathcal{A}^{\mathsf{S}(\cdot),\mathsf{E}(\cdot),\mathsf{K}(\cdot),\mathsf{SIE}(\cdot,\cdot)}(x_b) = b \;:\; \begin{array}{c} 1^\kappa \leftarrow \mathcal{A}(1^\lambda); (\mathsf{msk},\mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \\ b \leftarrow \{0,1\};\; x_0 \leftarrow \mathcal{X} \\ (x_1, y_1) \leftarrow \mathsf{Samp}(\mathsf{tk}, 0) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

*where the oracles* $\mathsf{S}, \mathsf{E}, \mathsf{K}, \mathsf{SIE}$ *are defined as follows:*

- $\mathsf{S}(\cdot) = \mathsf{Samp}(\mathsf{tk}, \cdot)$ *is the sampling oracle with* $\mathsf{tk}$ *hardwired,*
- $\mathsf{E}(\cdot) = \mathsf{Eval}(\mathsf{msk}, \cdot)$ *is the evaluation oracle with* $\mathsf{msk}$ *hardwired,*
- $\mathsf{K}(\cdot) = \mathsf{KeyGen}(\mathsf{msk}, \cdot)$ *is the key-generation oracle with* $\mathsf{msk}$ *hardwired, and*
- $\mathsf{SIE}(\cdot,\cdot)$ *is a randomized oracle that has* $\mathsf{msk}$ *hardwired, takes as input an identity* $\mathsf{id} \in \{0,1\}^\kappa$*, a string* $x \in \mathcal{X}$*, and outputs* $y = \mathsf{Eval}(\mathsf{sk}_{\mathsf{id}}, x)$ *where* $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$*.*

**Definition 4.2 (Identity hiding).** *A PLCPRF scheme is said to satisfy identity hiding if for every stateful PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*, the following holds:*

$$\Pr\left[\mathcal{A}^{\mathsf{S}(\cdot),\mathsf{E}(\cdot),\mathsf{K}(\cdot),\mathsf{SIE}(\cdot,\cdot)}(x) = b \;:\; \begin{array}{c} 1^\kappa \leftarrow \mathcal{A}(1^\lambda); (\mathsf{msk},\mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \\ (t_0, t_1) \leftarrow \mathcal{A}^{\mathsf{S}(\cdot),\mathsf{E}(\cdot),\mathsf{K}(\cdot),\mathsf{SIE}(\cdot,\cdot)} \\ b \leftarrow \{0,1\};\; (x, y) \leftarrow \mathsf{Samp}(\mathsf{tk}, t_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

*where the oracles are defined as in Definition 4.1, and* $\mathcal{A}$ *must not query oracle* $\mathsf{SIE}$ *on the input-identity pair* $(x, \mathsf{id})$ *where* $\mathsf{id} \in [t_0, t_1 - 1]$*, and each query* $\mathsf{id}$ *made by* $\mathcal{A}$ *to the key-generation oracle* $\mathsf{K}$ *must satisfy the condition that* $\mathsf{id} \notin [t_0, t_1 - 1]$*.*[13] *We say the PLCPRF scheme satisfies* selective *identity hiding if the adversary has to commit to its challenge identities* $(t_0, t_1)$ *at the* beginning *of the game before it makes any oracle queries. Note that selective security implies adaptive security at the expense of a sub-exponential loss in the security reduction via a technique called complexity leveraging [BB04].*

**Definition 4.3 (Pseudorandomness).** *A PLCPRF scheme is said to satisfy pseudorandomness if for every stateful PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*, the following holds:*

$$\Pr\left[\mathcal{A}^{\mathsf{S}(\cdot),\mathsf{E}(\cdot),\mathsf{K}(\cdot),\mathsf{SIE}(\cdot,\cdot)}(x, y_b) = b \;:\; \begin{array}{c} 1^\kappa \leftarrow \mathcal{A}(1^\lambda); (\mathsf{msk},\mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\kappa) \\ b \leftarrow \{0,1\};\; (x, y_0) \leftarrow \mathsf{Samp}(\mathsf{tk}, 2^\kappa);\; y_1 \leftarrow \mathcal{Y} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

*where the oracles are defined as in Definition 4.1,* $\mathcal{A}$ *cannot query oracle* $\mathsf{SIE}$ *on the input-identity pair* $(x, 2^\kappa)$*, and* $\mathcal{A}$ *cannot query the evaluation oracle* $\mathsf{E}$ *on input* $x$*.*

[13] Here and throughout, the $\kappa$-bit identities are interpreted as non-negative integers between 0 and $2^\kappa - 1$ for comparison.

*Remark 4.4 (Multi-challenge security).* For security of PLCPRFs, we consider three properties (normal hiding, identity hiding, and pseudorandomness). Note that in each of Definitions 4.1 to 4.3, we consider a single-challenge variant which means that the adversary gets to see exactly one challenge element. For instance, in the normal hiding game it gets a single challenge $x_b$ which is either a random input or an input sampled corresponding to threshold 0.

Consider a multi-challenge variant of these security properties where the adversary instead gets unbounded access to a challenge oracle, where the challenge oracle on each query provides a fresh sample from the corresponding challenge distribution. For instance, in the multi-challenge version of normal hiding, the adversary gets oracle access to a challenge oracle where on every query, the challenger provides a freshly sampled input $x_b$ which is either a random input or an input sampled for threshold 0. (Here, the challenge bit $b$ is chosen only once.) In our transformation provided in Section 4.2, we will rely on this multi-challenge variant of the security game. Note that single-challenge and multi-challenge definitions are equivalent since the adversary is given unbounded oracle access to the sampling oracle $\mathsf{S}$ in these games already. This follows from a standard hybrid argument.

*Remark 4.5 (Security for publicly-sampleable PLCPRFs).* Similar to that for traceable PRFs, a PLCPRF with public-sampleability is defined identically to its secret-key counterpart, except now the attacker is additionally provided the tracing key $\mathsf{tk}$ in all the security games.

*Remark 4.6 (Single-key security).* In some settings, we will consider private linear constrained PRFs where the security properties (Definitions 4.1 and 4.3) only hold against adversaries that can make a *single* key-generation query. We refer to such schemes as single-key private linear constrained PRFs. In the single-key setting, we also consider the *selective* notion of security where the adversary is required to commit to its key-generation query at the beginning of the security game (before making any oracle queries or in the case of the public-tracing setting, seeing the tracing key). Note that selective single-key security implies the standard adaptive single-key security at the expense of making a stronger sub-exponential hardness assumption via complexity leveraging [BB04].

### 4.2 Building Traceable PRFs

In this section, we show how to build a traceable PRF scheme from a private linear CPRF scheme. First, we recall the 'jump-finding' problem introduced in the work of Nishimaki et al. [NWZ16]. Later on, we describe our construction.

**Definition 4.7 (Noisy jump finding problem [NWZ16, Definition 3.6]).** *The $(N, q, \delta, \varepsilon)$-jump-finding problem is defined as follows. An adversary chooses a set $C \subseteq [N]$ of $q$ unknown points. Then, the adversary provides an oracle $P\colon [0, N] \to [0, 1]_{\mathcal{R}}$ with the following properties:*

- $|P(N) - P(0)| \geq \varepsilon$.

– *For any* $x, y \in [0, N]$ *where* $x < y$ *and* $[x+1, y] \cap C = \varnothing$*, then* $|P(y) - P(x)| < \delta$*.*

*The* $(N, q, \delta, \varepsilon)$*-jump finding problem is to interact with the oracle* $P$ *and output an element in* $C$*. In the* $(N, q, \delta, \varepsilon)$-noisy jump finding problem*, the oracle* $P$ *is replaced with a randomized oracle* $Q\colon [0, N] \to \{0, 1\}$ *where on input* $x \in [0, N]$*,* $Q(x)$ *outputs* 1 *with probability* $P(x)$*. A fresh independent draw is chosen for each query to* $Q(x)$*.*

**Theorem 4.8 (Noisy jump finding algorithm [NWZ16, Theorem 3.7]).** *There is an efficient algorithm* $\mathsf{QTrace}^Q(\lambda, N, q, \delta, \varepsilon)$ *that runs in time* $t = \mathsf{poly}(\lambda, \log N, q, 1/\delta)$ *and makes at most* $t$ *queries to* $Q$ *that solves the* $(N, q, \delta, \varepsilon)$*-noisy-jump-finding problem whenever* $\varepsilon > \delta(5 + 2(\lceil \log N - 1 \rceil)q)$*. In particular,* $\mathsf{QTrace}^Q(\lambda, N, q, \delta, \varepsilon)$ *will output at least one element in* $C$ *with probability* $1 - \mathsf{negl}(\lambda)$ *and will never output an element outside* $C$*. Moreover, any element* $x$ *output by* $\mathsf{QTrace}^Q(\lambda, N, q, \delta, \varepsilon)$ *has the property that* $P(x) - P(x-1) > \delta$*, where* $P(x) = \Pr[Q(x) = 1]$*.*

*Remark 4.9 (Relaxed non-intersection property [NWZ16, Remark 3.8]).* The algorithm $\mathsf{QTrace}^Q$ in Theorem 4.8 succeeds in solving the $(N, q, \delta, \varepsilon)$-noisy-jump-finding problem even if the associated oracle $P$ does not satisfy the second property in Definition 4.7: namely, there may exist $x, y$ where $[x + 1, y] \cap C = \varnothing$ and $|P(y) - P(x)| \geq \delta$. As long as the property holds for all pairs $x, y$ queried by $\mathsf{QTrace}^Q$, Theorem 4.8 applies.

**Construction 4.10** (Traceable PRF)**.** *Let* $\mathsf{PLCPRF} = (\mathsf{PL.Setup}, \mathsf{PL.KeyGen}, \mathsf{PL.Eval}, \mathsf{PL.Samp})$ *be a private linear CPRF scheme with input-output space* $\mathcal{X}$ *and* $\mathcal{Y}$*. Below we construct a traceable PRF scheme with identical input-output spaces. (Here we provide a transformation for PRF schemes with secret key tracing, but the construction can be easily extended to work in the public tracing setting if the special sampling algorithm in the underlying* PLCPRF *scheme is public key as well, that is tracing key* $\mathsf{tk}$ *is public).*

$\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{msk}, \mathsf{tk})$*. The setup algorithm runs the* PLCPRF *setup as* $(\mathsf{msk}, \mathsf{tk}) \leftarrow \mathsf{PL.Setup}(1^\lambda, 1^\kappa)$*, and outputs master secret-tracing key pair as* $(\mathsf{msk}, \mathsf{tk})$*.*

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$*. The key generation algorithm runs the* PLCPRF *key generation algorithm as* $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{PL.KeyGen}(\mathsf{msk}, \mathsf{id})$*, and outputs secret key* $\mathsf{sk}_{\mathsf{id}}$*.*

$\mathsf{Eval}(\mathsf{sk}, x) \to y$*. The evaluation algorithm runs the* PLCPRF *evaluation algorithm as* $y = \mathsf{PL.Eval}(\mathsf{sk}, x)$*, and outputs* $y$*.*

$\mathsf{Trace}^D(\mathsf{tk}, 1^z, q) \to T$*. The tracing algorithm runs the* QTrace *algorithm twice as* $T^{(\mathsf{real})} \leftarrow \mathsf{QTrace}^{Q_D^{(\mathsf{real})}}(\lambda, 2^\kappa, q, \delta, \varepsilon)$ *and* $T^{(\mathsf{rnd})} \leftarrow \mathsf{QTrace}^{Q_D^{(\mathsf{rnd})}}(\lambda, 2^\kappa, q, \delta, \varepsilon)$*, where* $\delta = \varepsilon/(5 + 2\kappa q)$*,* $\varepsilon = 1/z$*, and oracles* $Q_D^{(\mathsf{real})}$ *and* $Q_D^{(\mathsf{rnd})}$ *are described in Fig. 3. Finally, it outputs the set as* $T^{(\mathsf{real})} \cup T^{(\mathsf{rnd})}$*.*

*On input $\tau \in [0, 2^\kappa]$, the oracle $Q_D^{(\mathsf{mode})}$ proceeds as follows:*

- *Let $\mathsf{comp}_\tau$ denote the comparison function that on input $\mathsf{inp}$, outputs 1 if and only if $\mathsf{inp} \geq \tau$.*
- *Run the (stateful) oracle $D$, where on each query made by $D$ the oracle $Q_D$ samples an input-output pair as $(x, y) \leftarrow \mathsf{PL.Samp}(\mathsf{tk}, \tau)$. It samples a random output string $y' \leftarrow \mathcal{Y}$. If $\mathsf{mode} = \mathsf{real}$, it sends $(x, y)$ as the query response to $D$. Otherwise, it sends $(x, y')$ as the query response to $D$.*
- *Finally, $D$ outputs a bit $b$, and oracle $Q_D$ outputs the same bit $b$.*

Fig. 3: The distinguishing oracle $Q_D^{(\mathsf{mode})}$ for $\mathsf{mode} \in \{\mathsf{real}, \mathsf{rnd}\}$.

*Remark 4.11 (Additional parameter $q$).* Note that here the trace algorithm takes an additional parameter $q$. This is not an additional restriction since one could simply run the tracing algorithm increasingly with parameter $q$ growing as successive powers of two as long as the tracing algorithm outputs an empty set. A similar approach was taken in prior works such as [NWZ16, GKW19b].

### 4.3 Security

In this section, we prove security of our construction. Formally, we prove the following.

**Theorem 4.12 (Correctness).** *If the PLCPRF scheme* $\mathsf{PLCPRF} = (\mathsf{PL.Setup}, \mathsf{PL.KeyGen}, \mathsf{PL.Eval}, \mathsf{PL.Samp})$ *satisfies the key-similarity (resp., key-indistinguishability) property (Definitions 3.2 and 3.3, respectively), then the scheme* $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ *from Construction 4.10 also satisfies key-similarity (resp., key-indistinguishability).*

The above theorem follows directly from our construction. Next, we prove tracing security of our scheme.

**Theorem 4.13 (Security).** *If the scheme* $\mathsf{PLCPRF} = (\mathsf{PL.Setup}, \mathsf{PL.KeyGen}, \mathsf{PL.Eval}, \mathsf{PL.Samp})$ *satisfies normal hiding (Definition 4.1), identity hiding (Definition 4.2), and pseudorandomness (Definition 4.3) (resp., in the absence of* $\mathsf{SIE}$ *queries), then the scheme* $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ *from Construction 4.10 is a secure traceable PRF scheme as per Definition 3.5 (resp., in the absence of* $\mathsf{SIE}$ *queries).*

We provide an overview of the security proof below and provide the full proof in the full version of this paper [GKWW20].

*Proof overview.* We prove the theorem in two parts. First, we show that the false tracing probability is bounded by a negligible function. Next, we show the

correct tracing probability is close to the probability of adversary outputting an $\varepsilon$-good distinguisher for some non-negligible $\varepsilon$.

We begin by introducing some notation for the overview. Fix some master secret-tracing key pair $(\mathsf{msk}, \mathsf{tk})$. Given any pirate distinguisher $D$ and threshold $\tau \in [0, 2^\kappa]$, let

$$p^{\tau,D} = \Pr\left[D^{\mathsf{PL.Samp}(\mathsf{tk},\tau)}(1^\lambda) = 0\right] \text{ and } q^{\tau,D} = \Pr\left[D^{\widetilde{\mathsf{PL.Samp}}(\mathsf{tk},\tau)}(1^\lambda) = 0,\right]$$

where the oracle algorithm $\widetilde{\mathsf{PL.Samp}}$ is defined as the regular $\mathsf{PL.Samp}$ oracle algorithm, except the second tuple element (i.e., the output string) is sampled uniformly at random. Concretely, on each query to $\widetilde{\mathsf{PL.Samp}}(\mathsf{tk}, \tau)$, the oracle first samples $(x, y) \leftarrow \mathsf{PL.Samp}(\mathsf{tk}, \tau)$, $y' \leftarrow \mathcal{Y}$, and outputs $(x, y')$ as the response. Here the probability is taken over the random coins of distinguisher $D$ as well as the randomness used by the sample algorithm. Similarly, let

$$p^{\mathrm{nrml},D} = \Pr\left[D^{\mathsf{Real}(\mathsf{msk})}(1^\lambda) = 0\right], \text{ and } p^{\mathsf{rnd},D} = \Pr\left[D^{\mathsf{Rand}}(1^\lambda) = 0\right]$$

where oracle $\mathsf{Real}(\mathsf{msk})$ on each query, samples a random input $x \leftarrow \mathcal{X}$, and outputs $(x, \mathsf{Eval}(\mathsf{msk}, x))$ as the query response; whereas the oracle $\mathsf{Rand}$ is simulated by sampling by a random function $f : \mathcal{X} \to \mathcal{Y}$, and on each query, it samples a random input $x \leftarrow \mathcal{X}$, and outputs $(x, f(x)))$ as the query response. The above probabilities are also parameterized by the PLCPRF keys, but for simplicity of notation we do not include them as they are clear from context.

Now, suppose there exists a successful attacker $\mathcal{A}$. That is, $\mathcal{A}$ produces a distinguisher $D^*$, after making polynomially-many evaluation and key-generation queries, such that $p^{\mathrm{nrml},D^*} - p^{\mathsf{rnd},D^*} \geq 2\varepsilon$, and the tracing algorithm outputs either an empty set or an identity outside the set of identities queried by $\mathcal{A}$.[14] Let $\delta = \varepsilon/(5 + 2\kappa q)$ as used in the construction. Let $\gamma^*$ denote the probability that the distinguisher $D^*$ outputs 0 when given oracle access to a random function. Thus, we get that $p^{\mathrm{nrml},D^*} \geq \gamma^* + 2\varepsilon$. We first argue that it must also be the case that $p^{0,D^*} > \gamma^* + 2\varepsilon - \delta$, as otherwise we could use $\mathcal{A}$ to break the PLCPRF normal hiding property. Next, we also show that for any two thresholds $\tau_1 < \tau_2$, $p^{\tau_1,D^*} - p^{\tau_2,D^*} < \delta$ and $q^{\tau_2,D^*} - q^{\tau_1,D^*} < \delta$ as long as $\mathcal{A}$ does not make any key-generation query for an identity in the range $[\tau_1, \tau_2 - 1]$. This argument relies on the identity hiding property of the PLCPRFs. Next, we argue that $p^{2^\kappa,D^*} - q^{2^\kappa,D^*} < \delta$, as otherwise we could break the pseudorandomness security of the PLCPRFs. Lastly, we also argue that $q^{0,D^*} - p^{\mathsf{rnd},D^*} < \delta$, as otherwise we could break the PLCPRF normal hiding property. Combining these statements with the guarantees provided by the noisy jump finding algorithm (Theorem 4.8), we conclude that the tracing does not output an incorrect identity. □

[14] Recall that if $D^*$ is a $\varepsilon$-good distinguisher, then we have the bound $\Pr\left[D^{*O_b(\mathsf{msk})}(1^\lambda) = b \;:\; b \leftarrow \{0,1\}\right] \geq \varepsilon$. This can be rewritten as $p^{\mathrm{nrml},D^*} - p^{\mathsf{rnd},D^*} \geq 2\varepsilon$.

*Remark 4.14 (Public-tracing and handling* $\mathsf{SIE}$ *oracle queries).* In the proof of Theorem 4.13 above, we showed that Construction 4.10 gives a traceable PRF scheme with private tracing (which is secure in the absence of special evaluation oracle ($\mathsf{SIE}$) queries), as long as the underlying PLCPRF scheme is privately-sampleable and secure in the absence of $\mathsf{SIE}$ queries. However, if the underlying PLCPRF scheme is either publicly-sampleable or secure in presence of $\mathsf{SIE}$ queries, or both, then the reduction algorithm described above easily extends to prove the construction described above to be publicly-traceable, or secure in presence of $\mathsf{SIE}$ queries, or both, respectively.

*Remark 4.15 (Single-key security).* In the proof of Theorem 4.13, the number of key-generation queries each of the reduction algorithms needs to make to the underlying private linear constrained PRF is equal to the number of key-generation queries the tracing adversary makes. Thus, if we have a single-key private linear constrained PRF (Remark 4.6), that implies a traceable PRF with security against adversaries that can only make a single key-generation query. In Section 5, we show how to construct a single-key private linear constrained PRF from standard lattice assumptions (using single-key private constrained PRFs) as a starting point. It is an open problem to construct a many-key (i.e., collusion-resistant) private linear constrained PRF (or a traceable PRF) from standard lattice assumptions. We can construct a fully collusion-resistant private linear constrained PRF from indistinguishability obfuscation and injective one-way functions (see Section 6).

## 5 Privately-Traceable Private Linear Constrained PRFs

In this section, we show how to construct a single-key private linear constrained PRF from a private constrained PRF (for general circuit constraints) and an authenticated encryption scheme. Together with Construction 4.10, this yields a single-key traceable PRF in the private-tracing setting (and without access to the $\mathsf{SIE}$) from standard lattice assumptions (namely, on the sub-exponential hardness of LWE with a sub-exponential modulus-to-noise ratio). We define private constrained PRFs below and provide the formal definitions of authenticated encryption in the full version of this paper [GKWW20].

### 5.1 Private Constrained PRFs

*Syntax.* A private constrained PRF with input space $\mathcal{X}$, output-space $\mathcal{Y}$, and constraint family $\mathcal{F} = \{\mathcal{F}_{\lambda,\kappa}\}_{\lambda,\kappa\in\mathbb{N}}$ where $\mathcal{F}_{\lambda,\kappa} = \{f\colon \mathcal{X} \to \{0,1\}\}$ consists of the following algorithms:

$\mathsf{Setup}(1^\lambda, 1^\kappa) \to \mathsf{msk}$. The setup algorithm takes as input the security parameter $\lambda$ and a constraint-family parameter $\kappa$ and outputs a master PRF key $\mathsf{msk}$.

$\mathsf{Constrain}(\mathsf{msk}, f) \to \mathsf{sk}_f$. The constrain algorithm takes as input the master secret key $\mathsf{msk}$ and a constraint $f \in \mathcal{F}_{\lambda,\kappa}$ and outputs a constrained key $\mathsf{sk}_f$.

$\mathsf{Eval}(\mathsf{sk}, x) \to y$. The evaluation algorithm takes as input a secret key $\mathsf{sk}$ (which could be the master secret key $\mathsf{msk}$) and an input $x \in \mathcal{X}$ and outputs a value $y \in \mathcal{Y}$.

*Correctness and security.* We describe the correctness and security definitions for a private constrained PRF in the full version of this paper [GKWW20].

*Instantiations.* Private constrained PRFs (for general circuit constraints) satisfying the above properties can be built assuming sub-exponential hardness of LWE (with a sub-exponential modulus-to-noise ratio) [BTVW17, PS18].

### 5.2 Constructing a Private Linear Constrained PRF

We begin with a brief overview of our construction of a private linear constrained PRF. As discussed in Section 1.1, the domain of our PRF will be the ciphertext space $\{0,1\}^\ell$ for an authenticated encryption scheme with pseudorandom ciphertexts. A point corresponding to an index $t \in [0, 2^\kappa]$ (as would be output by the $\mathsf{Samp}$ algorithm) is an authenticated encryption of $t$. The PRF itself is implemented using a private constrained PRF, and the marked keys in our system correspond to a constrained key. Specifically, a marked key for an identity $\mathsf{id}$ consists of a constrained key for the function $f_{\mathsf{sk},\mathsf{id}}$ that has the secret key $\mathsf{sk}$ and the identity $\mathsf{id}$ hard-wired within in. The constraint $f_{\mathsf{sk},\mathsf{id}}$ has the property that $f_{\mathsf{sk},\mathsf{id}}(x) = 0$ whenever $x$ is a valid encryption under $\mathsf{sk}$ of some index $t' > \mathsf{id}$, and is 1 otherwise.

At a high-level, the security proof relies on the fact that a private constrained PRF hides the constraint function, which in this particular case, means that it hides the secret key $\mathsf{sk}$. Then, normal hiding and identity hiding follows from the fact that the ciphertexts in the underlying authenticated encryption scheme are pseudorandom, and pseudorandomness follows from constrained security of the underlying constrained PRF. We give our formal construction and security analysis below:

**Construction 5.1** (Private linear constrained PRF)**.** *Fix a security parameter $\lambda$ and an identity-space parameter $\kappa$. Our private linear constrained PRF relies on the following ingredients:*

- *A symmetric encryption scheme* $(\mathsf{SE.Setup}, \mathsf{SE.Enc}, \mathsf{SE.Dec})$ *with key-space $\mathcal{K}$, message-space $\mathcal{M} = \{\mathcal{M}_\kappa\}_{\kappa\in\mathbb{N}}$ where $\mathcal{M}_\kappa = [0, 2^\kappa]$, and ciphertext space $\mathcal{C} = \{\mathcal{C}_{\lambda,\kappa}\}_{\lambda,\kappa\in\mathbb{N}}$. Suppose that $\mathcal{C}_{\lambda,\kappa} \subseteq \{0,1\}^\ell$ where $\ell = \ell(\lambda, \kappa)$.*
- *For a symmetric encryption key $k \in \mathcal{K}$ and a threshold $t \in [0, 2^\kappa]$, let $f_{k,t} \colon \{0,1\}^\ell \to \{0,1\}$ be the following predicate:*

  > *On input* $\mathsf{ct} \in \{0,1\}^\ell$*:*
  >
  > 1. *Compute* $t' \leftarrow \mathsf{SE.Dec}(k, \mathsf{ct})$*. If* $\mathsf{SE.Dec}(k, \mathsf{ct})$ *does not have this form, output* 1*.*
  > 2. *Output* 1 *if* $t' \le t$ *and* 0 *otherwise.*

- *A private constrained PRF* $(\mathsf{PCPRF.Setup}, \mathsf{PCPRF.Eval}, \mathsf{PCPRF.Constrain})$ *with input space $\mathcal{X} = \{0,1\}^\ell$, output space $\mathcal{Y}$ and constraint family $\mathcal{F}_{\lambda,\kappa} = \{f_{k,t} \mid k \in \mathcal{K}, t \in [0, 2^\kappa]\}$.*

*We construct a private linear constrained PRF with input space $\mathcal{X} = \{0,1\}^\ell$, output space $\mathcal{Y}$ as follows:*

$\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{msk}, \mathsf{tk})$ *The setup algorithm samples a symmetric encryption key* $k \leftarrow \mathsf{SE.Setup}(1^\lambda, 1^\kappa)$ *and a private constrained PRF key* $\mathsf{pcprf.msk} \leftarrow \mathsf{PCPRF.Setup}(1^\lambda, 1^\kappa)$. *Then, it outputs* $\mathsf{msk} = \mathsf{tk} = (k, \mathsf{pcprf.msk})$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$. *The key-generation algorithm takes as input a master secret key* $\mathsf{msk} = (k, \mathsf{pcprf.msk})$ *and an identity* $\mathsf{id} \in [0, 2^\kappa]$ *and outputs* $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{PCPRF.Constrain}(\mathsf{pcprf.msk}, f_{k,\mathsf{id}})$.

$\mathsf{Eval}(\mathsf{sk}, x) \to y$. *The evaluation algorithm takes as input a secret key* $\mathsf{sk}$ *and an input* $x \in \{0,1\}^\ell$ *and output* $y \leftarrow \mathsf{PCPRF.Eval}(\mathsf{sk}, x)$.

$\mathsf{Samp}(\mathsf{tk}, t) \to (x, y)$. *The sampling algorithm takes as input the tracing key* $\mathsf{tk} = (k, \mathsf{pcprf.msk})$ *and a threshold* $t \in [0, 2^\kappa]$. *It computes* $x \leftarrow \mathsf{SE.Enc}(k, t)$ *and* $y \leftarrow \mathsf{PCPRF.Eval}(\mathsf{pcprf.msk}, x)$ *and outputs* $(x, y)$.

As long as the underlying private constrained PRF PCPRF is single-key secure and the underlying authenticated encryption scheme is secure, Construction 5.1 is a single-key private linear constrained PRF (see Remarks 4.6 and 4.15 for a discussion of single-key security). We provide the proofs in the full version of this paper [GKWW20].

**Theorem 5.2 (Key indistinguishability).** *Suppose* PCPRF *satisfies correctness and single-key selective privacy, and that* SE *satisfies ciphertext integrity. Then, the private linear constrained PRF from Construction 5.1 satisfies single-key selective key indistinguishability where the adversary is only able to choose* $\mathsf{idx} = 1$ *(i.e., the adversary can only target the identity key* $\mathsf{sk}_{\mathsf{id}}$ *it requested).*

**Theorem 5.3 (Single-key normal hiding).** *Suppose* PCPRF *satisfies single-key selective privacy,* SE *is correct and satisfies ciphertext integrity and ciphertext pseudorandomness. Then, the private linear constrained PRF from Construction 5.1 satisfies selective single-key normal hiding security (without* SIE *queries).*

**Theorem 5.4 (Single-key identity hiding).** *Suppose* PCPRF *satisfies single-key selective privacy,* SE *is correct and satisfies ciphertext integrity and CPA-security. Then, the private linear constrained PRF from Construction 5.1 satisfies selective single-key identity hiding security (without* SIE *queries).*

**Theorem 5.5 (Single-key pseudorandomness).** *Suppose* PCPRF *satisfies constrained pseudorandomness and* SE *is correct and satisfies CPA-security. Then, the private linear constrained PRF from Construction 5.1 satisfies selective single-key pseudorandomness (without* SIE *queries).*

*Instantiating Construction 5.1.* Combining a private constrained PRF for circuit constraints [BTVW17, PS18] with an authenticated encryption scheme with pseudorandom ciphertexts (implied by any one-way function), we obtain a private linear constrained PRF from sub-exponential hardness of LWE with a sub-exponential modulus-to-noise ratio (by applying complexity leveraging [BB04] to the selectively secure construction above).

## 6 Publicly-Traceable Private Linear Constrained PRFs

In this section, we show how to construct a publicly-traceable private linear constrained PRF from indistinguishability obfuscation [BGI+01] together with a puncturable public-key encryption scheme [CHN+16]. We provide the formal definitions of these building blocks in the full version of this paper [GKWW20].

Our construction takes the same general approach as our previous construction based on private constrained PRFs in Section 5. Namely, the domain of the PRF is the ciphertext space for a sparse (puncturable) public-key encryption scheme with pseudorandom ciphertexts.[15] The special points associated with an index $t \in [0, 2^\kappa]$ used for tracing correspond to encryptions of $t$ under the public-key encryption scheme. A marked key for an identity $\mathsf{id}$ consists of an obfuscated program that has both the decryption key hard-wired within it (needed to identify special points) as well as the master PRF key (in order to compute valid PRF evaluations on non-special points). Similarly, the public sampling algorithm consists of an obfuscated program with the master PRF key hard-wired and which takes an index $t$ and randomness $r$, and samples an input-output pair for the PRF. In the security proof, we show that security holds as long as the public-key encryption scheme and the underlying PRF are *puncturable*, and the analysis is a standard application of the punctured programming paradigm of [SW14].

**Construction 6.1** (Private linear constrained PRF with public tracing)**.** *Fix a security parameter $\lambda$ and an identity-space parameter $\kappa$. We rely on the following ingredients:*

- *A puncturable public-key encryption scheme* $(\mathsf{PE.Setup}, \mathsf{PE.Enc}, \mathsf{PE.Dec}, \mathsf{PE.Puncture})$ *with message space* $\mathcal{M} = \{\mathcal{M}_\kappa\}_{\kappa\in\mathbb{N}}$ *where* $\mathcal{M}_\kappa = [0, 2^\kappa]$*, and ciphertext space* $\mathcal{C} = \{\mathcal{C}_{\lambda,\kappa}\}_{\lambda\in\mathbb{N},\kappa\in\mathbb{N}}$*, where* $\mathcal{C}_{\lambda,\kappa} \subseteq \{0,1\}^\ell$ *for some* $\ell = \ell(\lambda, \kappa)$*. Let* $\rho = \rho(\lambda)$ *be a bound on the number of bits of randomness* $\mathsf{PE.Enc}$ *takes.*
- *A length-doubling pseudorandom generator* $\mathsf{PRG}\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$*.*
- *A puncturable PRF*[16] $(\mathsf{PPRF.Setup}, \mathsf{PPRF.Eval}, \mathsf{PPRF.Puncture})$ *with domain* $\{0,1\}^{\ell+2\lambda}$ *and range* $\mathcal{Y}$*.*
- *An indistinguishability obfuscator $i\mathcal{O}$ for general circuits.*

*We construct our private linear constrained PRF family with domain* $\{0,1\}^{\ell+2\lambda}$ *and range* $\mathcal{Y}$ *as follows:*

$\mathsf{Setup}(1^\lambda, 1^\kappa) \to (\mathsf{msk}, \mathsf{tk})$. *Sample a public and secret key-pair* $(\mathsf{PE.pk}, \mathsf{PE.sk}) \leftarrow \mathsf{PE.Setup}(1^\lambda, 1^\kappa)$ *and a puncturable PRF key* $\mathsf{PPRF.msk} \leftarrow \mathsf{PPRF.Setup}(1^\lambda)$*. Let* $P_{\mathsf{Samp}}[\mathsf{PE.pk}, \mathsf{PPRF.msk}]$ *be the following program:*

[15] To implement the punctured programming ideas from [SW14] in the security analysis, we also adjoin a long pseudorandom string to the domain

[16] A puncturable PRF is a constrained PRF (see Section 5.1) is a constrained PRF for the family of "puncturing" constraints $\mathcal{F} = \{f_x\colon \mathcal{X} \to \{0,1\} : x \in \mathcal{X}\}$ where $f_x(y) = 1$ if $x \neq y$ and 0 if $x = y$. They can be built directly from one-way functions [GGM84, BW13, KPTZ13, BGI14].

***Hard-wired:*** *a* $\mathsf{PE}$ *public key* $\mathsf{PE.pk}$ *and a puncturable PRF key* $\mathsf{PPRF.msk}$

***Input:*** *an index* $t \in [0, 2^\kappa]$*, and randomness* $r \in \{0,1\}^{\rho+\lambda}$*.*

- *Parse* $r = r_0 \| r_1$ *where* $r_0 \in \{0,1\}^\rho$ *and* $r_1 \in \{0,1\}^\lambda$*. Compute* $\mathsf{ct} \leftarrow \mathsf{PE.Enc}(\mathsf{PE.pk}, t; r_0)$*, set* $z \leftarrow \mathsf{ct} \| \mathsf{PRG}(r_1) \in \{0,1\}^{\ell+2\lambda}$*, and output* $(z, \mathsf{PPRF.Eval}(\mathsf{PPRF.msk}, z))$*.*

Fig. 4: The program $P_{\mathsf{Samp}}[\mathsf{PE.pk}, \mathsf{PPRF.msk}]$

*The setup algorithm outputs* $\mathsf{msk} \leftarrow (\mathsf{PE.pk}, \mathsf{PE.sk}, \mathsf{PPRF.msk})$ *and* $\mathsf{tk} \leftarrow i\mathcal{O}(1^\lambda, P_{\mathsf{Samp}}[\mathsf{PE.pk}, \mathsf{PPRF.msk}])$*. Note that* $P_{\mathsf{Samp}}[\mathsf{PE.pk}, \mathsf{PPRF.msk}]$ *is padded to be the maximum size of all modified* $P'_{\mathsf{Samp}}$ *programs that appear in the security analysis.*

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$*. Let* $P_{\mathsf{Eval}}[\mathsf{id}, \mathsf{PE.sk}, \mathsf{PPRF.msk}]$ *be the following program:*

***Hard-wired:*** *an identity* $\mathsf{id} \in \{0,1\}^\kappa$*, a* $\mathsf{PE}$ *secret key* $\mathsf{PE.sk}$*, and a PRF key* $\mathsf{PPRF.msk}$

***Input:*** *an input* $x \in \{0,1\}^{\ell+2\lambda}$

- *Parse* $x$ *as* $\mathsf{ct} \| x'$ *where* $\mathsf{ct} \in \{0,1\}^\ell$ *and* $x' \in \{0,1\}^{2\lambda}$*. Compute* $t' \leftarrow \mathsf{PE.Dec}(\mathsf{PE.sk}, \mathsf{ct})$*. If* $t' = \bot$*, output* $\mathsf{PPRF.Eval}(\mathsf{PPRF.msk}, x)$*.*
- *Otherwise, output* $\mathsf{PPRF.Eval}(\mathsf{PPRF.msk}, x)$ *if* $t' \leq \mathsf{id}$ *and* $\bot$ *if* $t' > \mathsf{id}$*.*

Fig. 5: The program $P_{\mathsf{Eval}}[\mathsf{id}, \mathsf{PE.sk}, \mathsf{PPRF.msk}]$

*Output* $\mathsf{sk}_{\mathsf{id}} \leftarrow i\mathcal{O}(1^\lambda, P_{\mathsf{Eval}}[\mathsf{id}, \mathsf{PE.sk}, \mathsf{PPRF.msk}])$*. Similar to* $\mathsf{Setup}$*, the program* $P_{\mathsf{Eval}}[\mathsf{id}, \mathsf{PE.sk}, \mathsf{PPRF.msk}]$ *is padded to be the maximum size of all modified* $P'_{\mathsf{Eval}}$ *programs that appear in the security analysis.*

$\mathsf{Eval}(\mathsf{sk}, x) \to y$*. If the secret key* $\mathsf{sk}$ *has the form* $(\mathsf{PE.pk}, \mathsf{PE.sk}, \mathsf{PPRF.msk})$*, then output* $\mathsf{PPRF.Eval}(\mathsf{PPRF.msk}, x)$*. Otherwise, if* $\mathsf{sk}_{\mathsf{id}}$ *is a description of an obfuscated program, output* $\mathsf{sk}_{\mathsf{id}}(x)$*.*

$\mathsf{Samp}(\mathsf{tk}, t) \to (x, y)$*. Sample a random* $r \leftarrow \{0,1\}^{\rho+\lambda}$ *and output* $\mathsf{tk}(t, r)$*.*

Due to space limitations, we defer the formal theorem statements and their proofs to the full version of this paper [GKWW20].

## References


ABP+17. Shweta Agrawal, Sanjay Bhattacherjee, Duong Hieu Phan, Damien Stehlé, and Shota Yamada. Efficient public trace and revoke from standard assumptions: Extended abstract. In *ACM CCS*, pages 2277–2293, 2017.

ADM+07. Michel Abdalla, Alexander W. Dent, John Malone-Lee, Gregory Neven, Duong Hieu Phan, and Nigel P. Smart. Identity-based traitor tracing. In *PKC*, pages 361–376, 2007.

BB04. Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.

BF99. Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In *CRYPTO*, pages 338–353, 1999.

BGI+01. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

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

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

BKS17. Foteini Baldimtsi, Aggelos Kiayias, and Katerina Samari. Watermarking public-key cryptographic functionalities and implementations. In *ISC*, pages 173–191, 2017.

BLW17. Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In *PKC*, pages 494–524, 2017.

BN08. Dan Boneh and Moni Naor. Traitor tracing with constant size ciphertext. In *ACM CCS*, pages 501–510, 2008.

BP08. Olivier Billet and Duong Hieu Phan. Efficient traitor tracing from collusion secure codes. In *ICITS*, pages 171–182, 2008.

BSW06. Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006.

BTVW17. Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained prfs (and more) from LWE. In *TCC*, pages 264–302, 2017.

BV15. Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC*, pages 1–30, 2015.

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

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

CFN94. Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, pages 257–270, 1994.

CFNP00. Benny Chor, Amos Fiat, Moni Naor, and Benny Pinkas. Tracing traitors. *IEEE Trans. Information Theory*, 46(3):893–910, 2000.

CHN+16. Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *STOC*, pages 1115–1127, 2016.

CPP05. Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In *EUROCRYPT*, pages 542–558, 2005.

CVW+18. Yilei Chen, Vinod Vaikuntanathan, Brent Waters, Hoeteck Wee, and Daniel Wichs. Traitor-tracing from LWE made simple and attribute-based. In *TCC*, pages 341–369, 2018.

FNP07. Nelly Fazio, Antonio Nicolosi, and Duong Hieu Phan. Traitor tracing with optimal transmission rate. In *ISC*, pages 71–88, 2007.

GGM84. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.

GKM+19. Rishab Goyal, Sam Kim, Nathan Manohar, Brent Waters, and David J. Wu. Watermarking public-key cryptographic primitives. In *CRYPTO*, pages 367–398, 2019.

GKRW18. Rishab Goyal, Venkata Koppula, Andrew Russell, and Brent Waters. Risky traitor tracing and new differential privacy negative results. In *CRYPTO*, pages 467–497, 2018.

GKSW10. Sanjam Garg, Abishek Kumarasubramanian, Amit Sahai, and Brent Waters. Building efficient fully collusion-resilient traitor tracing and revocation schemes. In *ACM CCS*, pages 121–130, 2010.

GKW18. Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In *STOC*, pages 660–670, 2018.

GKW19a. Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. *SIAM Journal on Computing*, (0):STOC18–94, 2019.

GKW19b. Rishab Goyal, Venkata Koppula, and Brent Waters. New approaches to traitor tracing with embedded identities. In *TCC*, pages 149–179, 2019.

GKWW20. Rishab Goyal, Sam Kim, Brent Waters, and David J. Wu. Beyond software watermarking: Traitor-tracing for pseudorandom functions. *IACR Cryptol. ePrint Arch.*, 2020:316, 2020.

GM84. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

GQWW19. Rishab Goyal, Willy Quach, Brent Waters, and Daniel Wichs. Broadcast and trace with n^$\epsilon$ ciphertext size from standard assumptions. In *CRYPTO*, pages 826–855, 2019.

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

JLS21. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. 2021.

KD98. Kaoru Kurosawa and Yvo Desmedt. Optimum traitor tracing and asymmetric schemes. In *EUROCRYPT*, pages 145–157, 1998.

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

KW17. Sam Kim and David J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In *CRYPTO*, pages 503–536, 2017.

KW19. Sam Kim and David J. Wu. Watermarking PRFs from lattices: Stronger security via extractable PRFs. In *CRYPTO*, pages 335–366, 2019.

KW20. Sam Kim and David J. Wu. Collusion resistant trace-and-revoke for arbitrary identities from standard assumptions. In *ASIACRYPT*, 2020.

KY02a. Aggelos Kiayias and Moti Yung. Traitor tracing with constant transmission rate. In *EUROCRYPT*, pages 450–465, 2002.

KY02b. Kaoru Kurosawa and Takuya Yoshida. Linear code implies public-key traitor tracing. In *PKC*, pages 172–187, 2002.

LPSS14. San Ling, Duong Hieu Phan, Damien Stehlé, and Ron Steinfeld. Hardness of k-LWE and applications in traitor tracing. In *CRYPTO*, pages 315–334, 2014.

Nis13. Ryo Nishimaki. How to watermark cryptographic functions. In *EUROCRYPT*, pages 111–125, 2013.

Nis20. Ryo Nishimaki. Equipping public-key cryptographic primitives with watermarking (or: A hole is to watermark). In *TCC*, 2020.

NP98. Moni Naor and Benny Pinkas. Threshold traitor tracing. In *CRYPTO*, pages 502–517, 1998.

NSS99. David Naccache, Adi Shamir, and Julien P. Stern. How to copyright a function? In *PKC*, pages 188–196, 1999.

NWZ16. Ryo Nishimaki, Daniel Wichs, and Mark Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In *EUROCRYPT*, pages 388–419, 2016.

PS18. Chris Peikert and Sina Shiehian. Privately constraining and programming PRFs, the LWE way. In *PKC*, pages 675–701, 2018.

PST06. Duong Hieu Phan, Reihaneh Safavi-Naini, and Dongvu Tonien. Generic construction of hybrid public key traitor tracing with full-public-traceability. In *ICALP*, pages 264–275, 2006.

QWZ18. Willy Quach, Daniel Wichs, and Giorgos Zirdelis. Watermarking prfs under standard assumptions: Public marking and security with extraction queries. In *TCC*, pages 669–698, 2018.

SSW01. Jessica Staddon, Douglas R. Stinson, and Ruizhong Wei. Combinatorial properties of frameproof and traceability codes. *IEEE Trans. Information Theory*, 47(3):1042–1049, 2001.

SW98. Douglas R. Stinson and Ruizhong Wei. Combinatorial properties and constructions of traceability schemes and frameproof codes. *SIAM J. Discrete Math.*, 11(1):41–53, 1998.

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

YAL+18. Rupeng Yang, Man Ho Au, Junzuo Lai, Qiuliang Xu, and Zuoxia Yu. Unforgeable watermarking schemes with public extraction. In *SCN*, pages 63–80, 2018.

YAL+19. Rupeng Yang, Man Ho Au, Junzuo Lai, Qiuliang Xu, and Zuoxia Yu. Collusion resistant watermarking schemes for cryptographic functionalities. In *ASIACRYPT*, pages 371–398, 2019.

YAYX20. Rupeng Yang, Man Ho Au, Zuoxia Yu, and Qiuliang Xu. Collusion resistant watermarkable prfs from standard assumptions. In *CRYPTO*, pages 590–620, 2020.

YF11. Maki Yoshida and Toru Fujiwara. Toward digital watermarking for cryptographic data. *IEICE Transactions*, 94-A(1), 2011.