Traceable PRFs: Full Collusion Resistance and Active Security

Sarasij Maitra
University of Virginia
sm3vg@virginia.edu

David J. Wu*
UT Austin
dwu4@cs.utexas.edu

Abstract

The main goal of traceable cryptography is to protect against unauthorized redistribution of cryptographic functionalities. Such schemes provide a way to embed identities (i.e., a "mark") within cryptographic objects (e.g., decryption keys in an encryption scheme, signing keys in a signature scheme). In turn, the tracing guarantee ensures that any "pirate device" that successfully replicates the underlying functionality can be successfully traced to the set of identities used to build the device.

In this work, we study traceable pseudorandom functions (PRFs). As PRFs are the workhorses of symmetric cryptography, traceable PRFs are useful for augmenting symmetric cryptographic primitives with strong traceable security guarantees. However, existing constructions of traceable PRFs either rely on strong notions like indistinguishability obfuscation or satisfy weak security guarantees like single-key security (i.e., tracing only works against adversaries that possess a *single* marked key).

In this work, we show how to use fingerprinting codes to upgrade a single-key traceable PRF into a *fully collusion resistant* traceable PRF, where security holds regardless of how many keys the adversary possesses. We additionally introduce a stronger notion of security where tracing security holds even against *active adversaries* that have oracle access to the tracing algorithm. In conjunction with known constructions of single-key traceable PRFs, we obtain the first fully collusion resistant traceable PRF from standard lattice assumptions. Our traceable PRFs directly imply new lattice-based secret-key traitor tracing schemes that are CCA-secure and where tracing security holds against active adversaries that have access to the tracing oracle.

1 Introduction

Traitor tracing [CFN94] and software watermarking schemes [BGI+01, BGI+12] are cryptographic primitives for protecting against the unauthorized distribution of software. In both settings, a content distributor can embed some special information (e.g., a "mark" or a "tag") into a program in a way that preserves the functionality of the program while ensuring that it is difficult for an adversary to remove the tag from the program without destroying its functionality. Schemes that provide strong security guarantees have typically focused on cryptographic programs. Specifically, traitor tracing schemes focus on protecting the decryption functionality in a (public-key) encryption scheme [CFN94, BSW06, BN08, GKW18, Zha20] while software watermarking has focused on symmetric primitives like pseudorandom functions (PRFs) [CHN+16, KW17, QWZ18, YAYX20] and on public-key primitives such as public-key encryption or digital signatures [GKM+19, Nis20].

Traceable PRFs. In this work, we study traceable PRFs, a notion recently introduced by Goyal et al. [GKWW21]. Recall first that a PRF [GGM84] is a keyed function $\mathsf{PRF}(k, \cdot)$ whose input/output behavior is computationally indistinguishable from a truly random function. As PRFs are the workhorses of symmetric cryptography, traceable PRFs are sufficient to augment a wide range of symmetric primitives with tracing capabilities: this

*Research supported by NSF CNS-1917414, CNS-2045180, and a Microsoft Research Faculty Fellowship.

can include notions such as symmetric encryption (which corresponds to secret-key traitor tracing), message authentication codes, or symmetric challenge-response authentication systems.

In a traceable PRF, the holder of the PRF key k can issue "marked" keys k_{id} associated with an identity id. First, the marked key k_{id} can be used to evaluate the PRF almost everywhere: namely, there is an efficient evaluation algorithm Eval where $\mathsf{Eval}(k_{\mathsf{id}}, x) = \mathsf{PRF}(k, x)$ for all but a negligible fraction of elements in the domain. Moreover, there is a tracing algorithm Trace that takes any "useful" program D and outputs at least one of the identity keys k_{id} that was used to construct D. More precisely, if an adversary that has keys $k_{\mathsf{id}_1}, \ldots, k_{\mathsf{id}_q}$ manages to create a "useful" program D, the tracing algorithm on program D should successfully output at least one of the identities $\mathsf{id}_1, \ldots, \mathsf{id}_q$.[1]

The question is how to define the "usefulness" of a program D. In the setting of watermarkable PRFs [CHN+16], a program D is considered useful only if $D(x) = \mathsf{PRF}(k, x)$ on at least a $(1/2 + \varepsilon)$-fraction of elements in the domain; in other words, programs are considered useful if they *exactly* preserve the output of the original PRF on most inputs. Goyal et al. [GKWW21] showed that this security notion is inadequate in settings where an adversarial program can break the security of a particular application without necessarily replicating the exact input/output behavior of the PRF. To address the weaknesses of the prevailing security notions for watermarking, Goyal et al. strengthened the "usefulness" definition on a program D to capture *all* programs that can successfully break (weak) pseudorandomness of the PRF. Specifically, any efficient program D that is able to distinguish a sequence $(x_1, \mathsf{PRF}(k, x_1)), \ldots, (x_n, \mathsf{PRF}(k, x_n))$ from $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$ with probability $1/2 + \varepsilon$, where $x_1, \ldots, x_n$ are random domain elements, f is a truly random function, and ε is non-negligible, is considered to be useful. In other words, the tracing algorithm should successfully extract an identity from any efficient distinguisher D that can *distinguish* PRF evaluations on random domain elements.

Collusion resistance. An important property in the study of traceable cryptography is collusion resistance, which requires that tracing security holds even if the adversary obtains *multiple* marked keys. We say a scheme is "fully collusion resistant" if security holds against adversaries that can obtain any unbounded polynomial number of keys.

Goyal et al. [GKWW21] gave two constructions of traceable PRFs: (1) a *single-key* construction from standard lattice assumptions where security holds against an adversary that holds a *single* marked key; and (2) a collusion resistant construction from indistinguishability obfuscation [BGI+01]. A natural question is whether we can obtain a collusion resistant traceable PRF from standard lattice assumptions. Such a construction would have the advantage of being plausibly post-quantum secure and also provides a more direct instantiation than going through the full power of indistinguishability obfuscation.

Fully collusion resistant constructions of related notions such as traitor tracing (i.e., traceable encryption) [GKW18, CVW+18] and watermarkable PRFs [YAYX20] are known from standard lattice assumptions.

Active security. Traceable PRFs and traitor tracing schemes come in several varieties. Some schemes support *public tracing* where anyone is able to run the tracing algorithm, while others only support *secret tracing* where knowledge of a secret key is needed to run the tracing algorithm. Existing lattice-based constructions of traceable PRFs and traitor tracing only support secret tracing.

In the secret tracing setting, existing security models only consider adversaries that do *not* have access to the tracing key. However, in practical scenarios where traitor tracing schemes may be deployed, it makes sense to consider *active adversaries* that may make multiple attempts to try and evade the tracing algorithm (or even worse, cause the tracing algorithm to falsely implicate an honest user). Certainly, any scheme that supports public tracing ensures robustness against such active adversaries, but the same does not hold in the secret-tracing setting. In this work, we model the capabilities of an active adversary by introducing a stronger security model in the secret tracing setting where we additionally allow the adversary to make queries to the tracing oracle. We view our notion to be an intermediate notion between secret tracing and public tracing. A similar intermediary notion was previously considered in the setting of watermarkable PRFs [QWZ18, KW19, YAL+19].

[1] While it might seem more natural to require that Trace outputs all of the identities $\mathsf{id}_1, \ldots, \mathsf{id}_q$, this requirement is impossible since an adversary can build its program D from just one of the keys it requested (and ignore all of the other ones).

This work. In this work, we show how to *generically* augment traceable PRFs with collusion resistance and active security through the use of fingerprinting codes [BS95].[2] We summarize our main results below and provide a more detailed technical overview in Section 1.1.

- **Collusion resistance:** We describe a generic transformation that transforms any single-key traceable PRF with domain $\mathcal{X}$, range $\{0,1\}^\rho$, and polynomial-size identity space $\mathcal{I}$ into a fully collusion resistant traceable PRF over the same domain, range, and identity space. A limitation of our construction is that the marked keys k_{id} are long: $|k_{\mathsf{id}}|$ scales polynomially with the size $|\mathcal{I}|$ of the identity space.

  We note that collusion resistance is meaningful and non-trivial to achieve even when the identity space is polynomial. For example, existing lattice-based traceable PRFs [GKWW21] are *completely* insecure if the adversary obtains just *two* marked keys (in fact, the adversary can recover the PRF secret key from any two marked keys). Moreover, in the closely-related setting of traitor tracing, many existing schemes only achieve full collusion resistance assuming a polynomial number of identities (e.g., [BSW06, GKSW10, Zha20, GQWW19]); in each of these examples, at least one of the scheme parameters grows *polynomially* with the number of identities, thus limiting the size of the identity space supported by the scheme.

- **Active security:** We describe a generic transformation that takes any single-key traceable PRF and compiles it into a traceable PRF with *active* security (i.e., where the adversary is allowed to have access to the tracing oracle). Combined with collusion resistant fingerprinting codes that support tracing queries [YAYX20], we obtain collusion resistant traceable PRFs with active security. We note that existing constructions of collusion resistant fingerprinting codes only support an a priori bounded polynomial number of tracing queries. The same limitation extends to our collusion resistant traceable PRFs with active security.

We capture these results in the following (informal) theorem:

Theorem 1.1 (Informal)**.** *Let λ be a security parameter and take any polynomial $n = n(\lambda)$. Let TPRF_0 be a single-key secretly-traceable PRF with domain $\mathcal{X}$, range $\{0,1\}^\rho$, and any identity space containing at least two identities. Then there exists a fully collusion resistant secretly-traceable PRF TPRF with domain $\mathcal{X}$, range $\{0,1\}^\rho$, and identity space $\{1,\ldots,n\}$. Moreover, for any polynomial $Q = Q(\lambda)$, TPRF is fully collusion resistant against an active adversary that makes up to Q queries to the tracing oracle. The size of the marked keys in TPRF is $\mathsf{poly}(\lambda, n, Q) \cdot |k_0|$, where $|k_0|$ denotes the size of a marked key in TPRF_0.*

Applying the above transformation to the single-key traceable PRF of Goyal et al. [GKWW21], we obtain the first fully collusion resistant traceable PRF from standard lattice assumptions. This puts traceable PRFs on par with the best-known results for watermarkable PRFs [YAYX20], while retaining the benefits of the significantly stronger tracing security provided by traceable PRFs. We summarize this instantiation in the following corollary to Theorem 1.1:

Corollary 1.2 (Collusion Resistant Traceable PRF)**.** *Under the sub-exponential hardness of LWE (with a sub-exponential modulus-to-noise ratio), there exists a fully collusion resistant traceable PRF with secret tracing and a polynomial identity space. The traceable PRF is secure against active adversaries making up to Q tracing queries, for any a priori bounded polynomial $Q = Q(\lambda)$.*

Applications to traitor tracing. As noted in Goyal et al. [GKWW21], traceable PRFs immediately give rise to symmetric traitor tracing schemes. Here, we note that if the underlying traceable PRFs provide active security, we obtain fully collusion resistant traitor tracing schemes with security against active adversaries. We also note that since PRFs can be directly used to construct a CCA-secure symmetric encryption scheme (and more generally, an authenticated encryption scheme [BN00]), our traceable PRF immediately implies a traitor tracing scheme for an authenticated encryption scheme. Previous constructions of traitor tracing (e.g.,

[2] The fingerprinting codes we rely on in this work [BS95, Tar03, YAL+19] are information-theoretic objects and do *not* require making additional computational assumptions.

[GKW18, CVW+18]) typically only consider chosen plaintext security (CPA-security) for the underlying encryption scheme. At the same time, the existing lattice-based traitor tracing schemes have the advantage that they support public encryption and have short marked keys. We provide more details in Section 4.

1.1 Construction Overview

In this section, we provide a high-level overview of our generic transformations (Theorem 1.1). We provide the technical details in Section 3.

Fingerprinting codes. Our construction combines a single-key (i.e., non-collusion-resistant) traceable PRF with a collusion resistant fingerprinting code [BS95]. A fingerprinting code is an information-theoretic primitive defined over an alphabet Σ and an identity space $\mathcal{I}$. Here, we consider binary codes so $\Sigma = \{0,1\}$ and a polynomial-sized identity space $\mathcal{I} = [n] = \{1, \ldots, n\}$. A fingerprinting code is described by two main algorithms $(\mathsf{Gen}, \mathsf{Trace})$:

- The code generator algorithm Gen is a randomized algorithm that outputs a codebook $\Gamma = \{\bar{w}^{(i)}\}_{i \in [n]}$ together with a tracing key tk. We say that $\bar{w}^{(i)} \in \{0,1\}^{\ell}$ is the "codeword" associated with the i^{th} identity and we refer to its length ℓ as the length of the code.
- The trace algorithm Trace takes as input the tracing key tk and a word $\bar{w}^* \in \{0,1\}^{\ell}$ and outputs a subset $S \subseteq [n]$.

Given a collection of codewords $W = \{\bar{w}^{(\mathsf{id}_1)}, \ldots, \bar{w}^{(\mathsf{id}_t)}\} \subseteq \Gamma$, we say that a word $\bar{w} \in \{0,1\}^{\ell}$ is *feasible* for W if for all $i \in [\ell]$, there exists $j \in [t]$ such that $\bar{w}_i = \bar{w}_i^{(\mathsf{id}_j)}$. In words, every bit in $\bar{w}$ agrees with the corresponding bit in one of the codewords in W. We define the feasible set $F(W) \subseteq \{0,1\}^{\ell}$ for W to be the set of words that are feasible for W.

Security for a fingerprinting code is defined by the following game between an adversary and a challenger. The challenger starts by sampling a codebook Γ and a tracing key tk. The adversary is allowed to adaptively request for codewords $\bar{w}^{(\mathsf{id})} \in \Gamma$ on identities $\mathsf{id} \in \mathcal{I}$ of its choosing. Let $T \subseteq \mathcal{I}$ be the set of identities queried by the adversary and let $W = \{\bar{w}^{(\mathsf{id})}\}_{\mathsf{id} \in T} \subseteq \Gamma$ be the set of associated codewords the adversary receives. At the end of the game, the adversary outputs a word $\bar{w}^* \in F(W)$ and wins if $\mathsf{Trace}(\mathsf{tk}, \bar{w}^*)$ outputs a set S where either $S = \varnothing$ or $S \not\subseteq T$. We say that a fingerprinting code is secure if no adversary $\mathcal{A}$ can win this game with non-negligible probability (taken over the code-generation and tracing randomness).

We note that fingerprinting codes can be used to directly construct collusion resistant *traitor tracing* [BN08, BP08]. In these settings, the resulting scheme satisfies a weaker *threshold* notion of traitor tracing where tracing succeeds only if the adversary outputs a decoder that succeeds with probability at least $1/2 + \varepsilon$ for a *predetermined* and fixed ε. In contrast, the standard tracing definitions used for traceable PRFs and traitor tracing allows tracing a decoder that succeeds for arbitrary inverse polynomial ε. In this work, we show how to use fingerprinting codes to *upgrade* a non-collusion-resistant traceable PRF to a collusion resistant one *without* weakening the traceability guarantee. Recently, Zhandry [Zha20] introduced new techniques to compile a threshold traitor tracing scheme into one without the threshold limitation. It is not clear whether those techniques extend to the traceable PRF setting (in fact, it does not seem straightforward to even construct a traceable PRF with a threshold security notion directly from fingerprinting codes).

Collusion resistant traceable PRFs. Our main construction relies on the simple observation that the xor function is a "combiner" for PRFs. Namely, if $\mathsf{PRF}_1, \ldots, \mathsf{PRF}_\ell \colon \mathcal{K} \times \mathcal{X} \to \{0,1\}^{\rho}$ are PRF candidates, then $\mathsf{PRF}((k_1, \ldots, k_\ell), x) := \bigoplus_{i \in [\ell]} \mathsf{PRF}_i(k_i, x)$ is a secure PRF as long as *at least one* of the PRF_i is secure.

Let $\mathsf{TPRF}_{\mathsf{nc}}$ be a single-key (i.e., *non-collusion-resistant*) traceable PRF, and let $\Gamma = \{\bar{w}^{(i)}\}_{i \in [n]}$ be a fingerprinting code for the set $[n]$, where each codeword $\bar{w}^{(i)} \in \{0,1\}^{\ell}$. Our construction will use ℓ independent copies of $\mathsf{TPRF}_{\mathsf{nc}}$, where each copy is used to embed a *single* bit of the codeword. In more detail, the PRF key is a tuple of ℓ independent PRF keys $(\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell)$ for $\mathsf{TPRF}_{\mathsf{nc}}$. The PRF evaluation is defined to be

$$\mathsf{Eval}((\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell), x) = \bigoplus_{i \in [\ell]} \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_i, x).$$

A marked key for an identity id consists of a tuple of marked keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$ where each sk_i is msk_i marked with the bit $\bar{w}_i^{(\mathsf{id})}$. Pseudorandomness of this construction also follows from pseudorandomness of $\mathsf{TPRF}_{\mathsf{nc}}$.

To trace a distinguisher D, we use the above combiner property: any algorithm that can break (weak) pseudorandomness of $\bigoplus_{i \in [\ell]} \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_i, \cdot)$ can also break (weak) pseudorandomness of $\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_j, \cdot)$ for every $j \in [\ell]$. In particular, it is straightforward to take a distinguisher D and convert it into a distinguisher D_j for $\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_j, \cdot)$; recall here that in the secret-tracing setting, the tracing algorithm has the master secret key of the traceable PRF. The tracing algorithm runs the underlying single-key tracing algorithm on each distinguisher D_j to obtain sets $T_1, \ldots, T_\ell$. It uses the sets T_j to construct a codeword $\bar{w}^* \in \{0,1\}^\ell$ as follows: if $0 \in T_j$, then set $\bar{w}_j^* = 0$. Otherwise, set $\bar{w}_j^* = 1$. The final output of the tracing algorithm is obtained by running the decoding algorithm for the fingerprinting code on the extracted word $\bar{w}^*$.[3]

We argue that tracing security reduces to security of the underlying single-key traceable PRF and of the fingerprinting code. Suppose the adversary asks for keys on identities $\mathsf{id}_1, \ldots, \mathsf{id}_q$ and manages to produce a useful distinguisher D. The argument then proceeds as follows:

- Let $W = \{\bar{w}^{(\mathsf{id}_1)}, \ldots, \bar{w}^{(\mathsf{id}_q)}\}$ be the set of codewords for the fingerprinting code that are associated with the identities queried by the adversary. As long as the word $\bar{w}^*$ extracted by the tracing algorithm is contained in the feasible set of W, then security of the fingerprinting code guarantees that tracing security holds.

- By construction, $\bar{w}^* \in \{0,1\}^\ell$. This means that $\bar{w}^* \in F(W)$ as long as for every index $i \in [\ell]$ where $\bar{w}_i^{(\mathsf{id}_j)} = \bar{w}_i^{(\mathsf{id}_1)}$ for all $j \in [q]$, $\bar{w}_i^* = \bar{w}_i^{(\mathsf{id}_1)}$. In other words, if *all* of the codewords corresponding to identities requested by the adversary have the same bit in a particular position, then the corresponding bit in $\bar{w}^*$ must also match. But this property directly follows by *single-key* tracing security. Namely, if the codewords corresponding to identities requested by the adversary all match in a particular index $i \in [\ell]$, then the adversary only obtains *one* marked version of msk_i.[4] As described above, if D is a useful distinguisher, then it can be used to obtain a useful distinguisher for *any* of the underlying traceable PRFs. We use D to obtain a useful distinguisher D_i for the i^{th} traceable PRF. Since the adversary only possesses a single marked key for the i^{th} PRF (marked with the bit $\bar{w}_i^{(\mathsf{id}_1)}$), single-key tracing security ensures that tracing distinguisher D_i correctly recovers $\bar{w}_i^{(\mathsf{id}_1)}$.

Essentially, single-key security of the traceable PRF binds the adversary to strategies that conform to the restrictions of the fingerprinting code model. This in turn yields a fully collusion resistant traceable PRF.

Active security. The second security property we consider in this work is active security, where tracing security holds even if an adversary has oracle access to the tracing algorithm. We start by considering active security in the *single-key* setting. Intuitively, security in this setting should *almost* follow from single-key tracing security. This is because if a distinguisher is "useful," then security requires that the tracing algorithm outputs the single identity id that the adversary requested. Conversely, if the distinguisher is "useless" (e.g., outputs a random guess), the tracing algorithm should output $\varnothing$ to avoid false implication of an honest user.

In some sense then, the adversary in the single-key security game should be able to "predict" the output of the tracing function in advance. If this is true, then the tracing oracle is no longer useful to the adversary and security reduces to the setting without tracing queries. However, the catch is handling distinguishers which are somewhere in between "useful" and "useless." For instance, the adversary might start with a useful

[3] To avoid falsely implicating an honest user, we also run a statistical test to check that the distinguisher is "sufficiently good." We refer to Section 3 for more details.

[4] For this step to work, we need to first *derandomize* the key-generation algorithm. This can be done via the standard approach of deriving the key-generation randomness from a PRF. We refer to Section 3 for the full construction and analysis.

distinguisher and construct distinguishers with progressively decreasing distinguishing advantage until it observes a change in the behavior of the tracing algorithm; where this occurs can leak information about the secret tracing key.

More precisely, the tracing algorithm in a traceable PRF takes a distinguisher D and a threshold ε as input. The requirement is that if D has distinguishing advantage at least $1/2 + \varepsilon$, then running the tracing algorithm with threshold ε will correctly identify at least one corrupted user. However, if D's distinguishing advantage is less than $1/2 + \varepsilon$, then the only guarantee provided by the tracing algorithm is that it does not falsely implicate an honest user; in this case, it can either output a compromised identity or the empty set.

Our approach to achieving active security is through introducing an efficient statistical test $\mathsf{CheckDis}$ for deciding whether a distinguisher is "useful" (in which case the tracing algorithm always outputs the single corrupted identity) or "not useful" (in which case the tracing algorithm always outputs $\varnothing$). Importantly, this test can be run by the adversary itself, so the tracing oracle does not provide the adversary additional information. The $\mathsf{CheckDis}$ algorithm is very simple: it takes a distinguisher D and a threshold ε and estimates the distinguishing advantage of D. The algorithm satisfies two properties:

- If the distinguishing advantage of D is at least $1/2 + \varepsilon$, then $\mathsf{CheckDis}$ outputs 1 with overwhelming probability.
- If $\mathsf{CheckDis}$ outputs 1, then the distinguishing advantage of D is at least $1/2 + \varepsilon/4$ with overwhelming probability.

We now modify the tracing algorithm to first run $\mathsf{CheckDis}$ on the distinguisher. If $\mathsf{CheckDis}$ fails, then the tracing algorithm always outputs $\varnothing$; the first property guarantees that this will never happen to a "good" distinguisher. If $\mathsf{CheckDis}$ succeeds, then run the tracing algorithm with distinguishing threshold $\varepsilon/4$; the second property ensures that the tracing algorithm correctly outputs the compromised identity in this case. Finally, since $\mathsf{CheckDis}$ can be computed by the adversary, it is possible for the adversary to simulate for itself the output of the tracing queries without access to the tracing oracle. Thus, in the single-key setting, it is straightforward to achieve active security essentially for free.

To obtain a collusion resistant traceable PRF with active security, we can apply our generic transformation based on fingerprinting codes. While the general transformation still applies, security will require that the underlying fingerprinting code remains secure in the presence of tracing queries. Currently, there exist collusion resistant fingerprinting codes that are secure against adversaries that make an *a priori* bounded polynomial number of tracing queries [YAYX20]. In conjunction with our compilers, this yields a collusion resistant traceable PRF that is secure against adversaries that can make a bounded number of tracing queries. Constructing fingerprinting codes that are secure against an unbounded polynomial number of tracing queries is an interesting open problem, and a construction would immediately yield traceable PRFs (and correspondingly, traitor tracing schemes) with active security.

1.2 Additional Related Work

In this section, we discuss some additional results on traitor tracing and watermarking.

Traitor tracing. Traitor tracing has been studied extensively and numerous constructions of traitor tracing have been proposed based on combinatorial techniques [CFN94, NP98, SSW01, CFNP00, SSW01, BN08] as well as algebraic techniques [BSW06, GKSW10, LPSS14, KT15, NWZ16, GKW18, CVW$^+$18, GKW19, GQWW19]. Some of these schemes are secure against bounded collusions [CFN94, SSW01, LPSS14, KT15, NWZ16] while others are fully collusion resistant [BSW06, GKSW10, GKW18, CVW$^+$18, GQWW19, Zha20]. We refer to these works and the references therein for further information.

Traitor tracing from fingerprinting codes. Fingerprinting codes can be directly combined with public-key encryption to obtain traitor tracing schemes (though not traceable PRFs) [BN08]. As noted earlier, the resulting traitor tracing scheme satisfies a weaker threshold tracing guarantee. Our work shows that by combining fingerprinting codes with an existing (non-collusion-resistant) tracing scheme, it is possible to obtain full collusion resistance *without* the threshold restriction.

In the setting of watermarkable PRFs, Yang et al. [YAL+19] showed how to use fingerprinting codes to upgrade a non-collusion-resistant watermarkable PRF to a collusion resistant one. Their approach relies on concatenating the outputs of many watermarkable PRFs and only supports tracing adversaries that preserve the entirety of the PRF output (i.e., this precludes applications from truncating the PRF output). Overall, both the size of the marked keys and the length of the PRF output of their scheme scale polynomially with the number of identities. Our collusion resistant traceable PRF has long keys, but the length of the PRF output is independent of the number of identities. For instance, this property enables symmetric traitor tracing with short ciphertexts.

Watermarking. Barak et al. [BGI+01, BGI+12] and Hopper et al. [HMW07] provided the first rigorous definitions of software watermarking. Multiple works have subsequently studied constructions of watermarking for symmetric primitives [CHN+16, BLW17, KW17, YAL+18, QWZ18, KW19, YAL+19, YAYX20] and public-key primitives [GKM+19, Nis20]. Goyal et al. [GKWW21] recently highlighted some definitional issues with watermarking for PRFs and introduced the notion of traceable PRFs.

2 Preliminaries

Notation. We write λ (oftentimes implicitly) to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$. For a finite set S, we write $x \xleftarrow{\text{R}} S$ to denote that x is sampled uniformly from S. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is sampled from $\mathcal{D}$. For an event E, we write $\neg E$ to denote its complement. For finite sets $\mathcal{X}$ and $\mathcal{Y}$, we write $\mathsf{Funs}[\mathcal{X}, \mathcal{Y}]$ to denote the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$.

We say that a function f is negligible in the parameter λ if $f(\lambda) = o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We denote this by writing $f(\lambda) = \mathsf{negl}(\lambda)$. We write $\mathsf{poly}(\lambda)$ to denote a function bounded by a fixed polynomial in λ. We say an event E (parameterized by a security parameter λ) happens with negligible probability if $\Pr[E] = \mathsf{negl}(\lambda)$ and that it happens with overwhelming probability if $\Pr[\neg E] = \mathsf{negl}(\lambda)$. We say an algorithm $\mathcal{A}$ is efficient if it runs in probabilistic polynomial time in the length of its input. We say that two families of distributions $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}_2 = \{\mathcal{D}_{2,\lambda}\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable if no efficient adversary can distinguish samples from $\mathcal{D}_1$ and $\mathcal{D}_2$ except with negligible probability. We will also use standard Chernoff/Hoeffding bounds in our analysis:

Fact 2.1 (Hoeffding's Inequality [Hoe63])**.** Let $X_1, \ldots, X_n$ be independent random variables where $0 \le X_i \le 1$ for all $i \in [n]$. Let $S = \sum_{i \in [n]} X_i$ and let $\mathbb{E}[S]$ denote the expected value of S. Then, for any $t \ge 0$,

$$\Pr[|S - \mathbb{E}[S]| \ge nt] \le 2^{-\Omega(nt^2)}.$$

Finally, we recall the definition of a pseudorandom function [GGM84]:

Definition 2.2 (Pseudorandom Function [GGM84])**.** A pseudorandom function (PRF) with key-space $\mathcal{K}$, domain $\mathcal{X}$ and range $\mathcal{Y}$ is an efficiently-computable function $\mathsf{PRF} \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ with the property that for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathcal{A}^{O_b(\cdot)}(1^\lambda) = b : k \xleftarrow{\text{R}} \mathcal{K}, f \xleftarrow{\text{R}} \mathsf{Funs}[\mathcal{X}, \mathcal{Y}], b \xleftarrow{\text{R}} \{0,1\}] \le \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $O_b(x)$ outputs $\mathsf{PRF}(k, x)$ if $b = 0$ and $f(x)$ if $b = 1$.

2.1 Fingerprinting Codes

In this section, we recall the formal definition of a fingerprinting code from Boneh and Shaw [BS95]. To construct traceable PRFs with active security, we require the fingerprinting code to satisfy collusion resistance against adversaries that are allowed to make tracing queries [YAYX20].

Definition 2.3 (Feasible Set [BS95])**.** Let $W = \{\bar{w}^{(1)}, \ldots, \bar{w}^{(t)}\} \subseteq \{0,1\}^\ell$. We say that a word $\bar{w} \in \{0,1\}^\ell$ is *feasible* for W if for all $i \in [\ell]$, there exists $j \in [t]$ such that $\bar{w}_i = \bar{w}_i^{(j)}$. We define the *feasible set* $F(W) \subseteq \{0,1\}^\ell$ of W to be the set of all words in $\{0,1\}^\ell$ that are feasible for W.

Definition 2.4 (Fingerprinting Code [BS95, adapted])**.** A fingerprinting code FC with n codewords is a pair of efficient algorithms $(\mathsf{Gen}, \mathsf{Trace})$ with the following properties:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{tk}, \Gamma)$: On input the security parameter $\lambda \in \mathbb{N}$, the code-generation algorithm outputs a tracing key tk and a dictionary $\Gamma = \{\bar{w}^{(i)}\}_{i \in [n]}$. Here, $\bar{w}^{(i)} \in \{0,1\}^\ell$ for some parameter $\ell > 0$. We refer to ℓ as the *code length*.

- $\mathsf{Trace}(\mathsf{tk}, \bar{w}^*) \to S$: On input the tracing key tk and a word $\bar{w}^* \in \{0,1\}^\ell$, the decoding algorithm outputs a set $S \subseteq [n]$.

Definition 2.5 (Collusion Resistance with Tracing Queries [YAYX20, adapted])**.** Let $\mathsf{FC} = (\mathsf{Gen}, \mathsf{Trace})$ be a fingerprinting code with n codewords. For an adversary $\mathcal{A}$, we define the fingerprinting code experiment $\mathsf{ExptFC}_{\mathcal{A}}^{\mathsf{FC}}(\lambda)$ as follows:

Experiment $\mathsf{ExptFC}_{\mathcal{A}}^{\mathsf{FC}}(\lambda)$:

- The challenger starts by sampling $(\mathsf{tk}, \Gamma = \{\bar{w}^{(i)}\}_{i \in [n]}) \leftarrow \mathsf{Gen}(1^\lambda)$. It also initializes an empty set $W \leftarrow \varnothing$.
- The adversary is given access to the following oracles:
  - **Encode query:** On input an index $i \in [n]$, the challenger replies with $\bar{w}^{(i)} \in \{0,1\}^\ell$. The challenger adds $\bar{w}^{(i)}$ to W.
  - **Tracing query:** On input a word $\bar{w}^* \in \{0,1\}^\ell$, if $\bar{w}^* \notin F(W)$, the challenger replies with $\perp$. Otherwise, if $\bar{w}^* \in F(W)$, then the challenger computes $S \leftarrow \mathsf{Trace}(\mathsf{tk}, \bar{w}^*)$. If $S \neq \varnothing$ and $\bar{w}^{(\mathsf{id})} \in W$ for all $\mathsf{id} \in S$, the challenger replies with S. Otherwise, the experiment halts with output 1.
- After the adversary $\mathcal{A}$ finishes making its queries, the experiment halts with output 0 (if it has not already halted).

We say that FC is fully collusion resistant in the presence of Q tracing queries if for all security parameters $\lambda \in \mathbb{N}$ and all adversaries $\mathcal{A}$ making up to Q tracing queries, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{ExptFC}_{\mathcal{A}}^{\mathsf{FC}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

When we allow Q to be an arbitrary polynomial, we say that FC is fully collusion resistant in the presence of tracing queries.

Fact 2.6 (Fingerprinting Codes)**.** We recall the following results on the existence of collusion resistant fingerprinting codes (with and without tracing queries):

- For all $\lambda \in \mathbb{N}$ and $n \in \mathbb{N}$, there exists a fingerprinting code that is fully collusion resistant *without* tracing queries (i.e., $Q = 0$) with code length $\ell = \mathsf{poly}(n, \lambda)$ [BS95, Tar03]. Specifically, the Tardos instantiation [Tar03] yields a construction with code-length $\ell = O(\lambda n^2 \log n)$.

- For all $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, and $Q = \mathsf{poly}(\lambda)$ there exists a fingerprinting code that is fully collusion resistant in the presence of Q tracing queries with code length $\ell = \mathsf{poly}(n, \lambda, Q)$ [YAYX20].

2.2 Traceable PRFs

In this section, we recall the formal definition of a traceable PRF from [GKWW21]. We note that our transformations will rely on a stronger notion of tracing security we call "strong tracing" (see Definition 2.12). Existing constructions of traceable PRFs [GKWW21] satisfy this security notion (Remark 2.13). Finally, we introduce our notion of tracing security against active adversaries that have oracle access to the tracing algorithm (Definition 2.14).

Definition 2.7 (Traceable PRFs [GKWW21])**.** Let λ be a security parameter. A traceable PRF scheme (in the secret-tracing setting) with domain $\mathcal{X}$, range $\mathcal{Y}$, and identity space $[n]$ where $n = n(\lambda)$ is a tuple of four algorithms $\mathsf{TPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ with the following properties:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{msk}$: The setup algorithm takes as input the security parameter λ and outputs a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$: The key generation algorithm takes as input the master secret key msk and an identity $\mathsf{id} \in [n]$, and outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.
- $\mathsf{Eval}(\mathsf{sk}, x) \to y$: The evaluation algorithm takes as input a secret key sk (which could be the master key msk), an input $x \in \mathcal{X}$, and outputs a value $y \in \mathcal{Y}$.
- $\mathsf{Trace}^D(\mathsf{msk}, 1^z) \to T$: The tracing algorithm has oracle access to an oracle-aided distinguisher D^O and takes as input the master secret key msk and a parameter z. It outputs a set of identities $T \subseteq [n]$. Note that the tracing algorithm must includes a description of how to implement the oracle O used by the oracle-aided distinguisher.

Correctness. The basic correctness requirement for a traceable PRF is that the behavior of the marked key agrees with the original key on all but a negligible fraction of the domain. We recall this below:

Definition 2.8 (Key Similarity)**.** A traceable PRF $\mathsf{TPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ with domain $\mathcal{X}$, range $\mathcal{Y}$, and identity space $[n]$ satisfies key similarity if for every security parameter $\lambda \in \mathbb{N}$, every identity $\mathsf{id} \in [n]$, there exists a negligible function $\mathsf{negl}(\cdot)$ where

$$\Pr\left[\mathsf{Eval}(\mathsf{msk}, x) \neq \mathsf{Eval}(\mathsf{sk}_{\mathsf{id}}, x) : \begin{array}{c} \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}), x \xleftarrow{\textsc{r}} \mathcal{X} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 2.9 (Stronger Notions of Correctness)**.** Definition 2.8 requires that marked keys agree with unmarked keys on all but a negligible fraction of the domain. Goyal et al. [GKWW21] also consider a stronger notion of key indistinguishability that requires that it is computationally difficult to *find* domain elements where the marked key and the unmarked key differ. We note that our generic transformations in Section 3 can be shown to preserve this stronger notion of correctness. For ease of exposition in this work, we focus on the simpler notion of key similarity.

Definition 2.10 (Weak Pseudorandomness)**.** A traceable PRF $\mathsf{TPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ with domain $\mathcal{X}$, range $\mathcal{Y}$, and identity space $[n]$ satisfies weak pseudorandomness if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\mathcal{A}^{O_b}(1^\lambda) = b : \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda), f \xleftarrow{\textsc{r}} \mathsf{Funs}[\mathcal{X}, \mathcal{Y}], b \xleftarrow{\textsc{r}} \{0,1\}\right] \leq \mathsf{negl}(\lambda),$$

where the weak PRF challenge oracle O_b samples $x \xleftarrow{\textsc{r}} \mathcal{X}$ and outputs $(x, \mathsf{Eval}(\mathsf{msk}, x))$ if $b = 0$ and $(x, f(x))$ if $b = 1$.

Remark 2.11 (On Weak Pseudorandomness)**.** Similar to Goyal et al. [GKWW21], we use weak pseudorandomness as our primary security notion for traceable PRFs. As discussed in [GKWW21, §3.1], tracing is only feasible against adversarial strategies that contain "global" information about the behavior of the PRF (i.e., adversaries that can break weak pseudorandomness). We do note that it is still possible for traceable PRFs to independently satisfy the usual notion of strong pseudorandomness (and indeed, the constructions of Goyal et al. do). All of the transformations developed in this work preserve strong pseudorandomness.

Definition 2.12 (Secure Tracing)**.** Let $\mathsf{TPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ be a traceable PRF with domain $\mathcal{X}$ and range $\mathcal{Y}$ and identity space $[n]$. For a function $\varepsilon = \varepsilon(\lambda)$ and adversary $\mathcal{A}$, we define the tracing experiment $\mathsf{ExptTPRF}^{\mathsf{TPRF}}_{\mathcal{A},\varepsilon}(\lambda)$ as follows:

Experiment $\mathsf{ExptTPRF}^{\mathsf{TPRF}}_{\mathcal{A},\varepsilon}(\lambda)$:

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

Let S_{id} be the set of identities $\mathcal{A}$ submits to the key-generation oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$. Based on the output of $\mathsf{ExptTPRF}^{\mathsf{TPRF}}_{\mathcal{A},\varepsilon}$, we define the following set of (probabilistic) events and their corresponding probabilities (which are a functions of λ and parameterized by $\mathcal{A}, \varepsilon$):

- $\mathsf{GoodDis}_{\mathcal{A},\varepsilon}$: This is the event where $\Pr[D^{O_b}(1^\lambda) = b \;:\; b \overset{\mathrm{R}}{\leftarrow} \{0,1\}, f \overset{\mathrm{R}}{\leftarrow} \mathsf{Funs}[\mathcal{X}, \mathcal{Y}]] \geq 1/2 + \varepsilon(\lambda)$, where the probability is taken over the coins of D, and the oracle O_b is the weak PRF challenge oracle: namely, O_b samples $x \overset{\mathrm{R}}{\leftarrow} \mathcal{X}$ and outputs $(x, \mathsf{Eval}(\mathsf{msk}, x))$ if $b = 0$ and $(x, f(x))$ if $b = 1$. Intuitively, this says that a distinguisher D is an ε-good distinguisher if D can break weak pseudorandomness of the underlying PRF with advantage $\varepsilon = \varepsilon(\lambda)$.
- $\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}$: This is the event where $T \neq \varnothing \wedge T \subseteq S_{\mathsf{id}}$. This event corresponds to the tracing algorithm successfully outputting one or more of the keys the adversary possesses.
- $\mathsf{BadTr}_{\mathcal{A},\varepsilon}$: This is the event where $T \not\subseteq S_{\mathsf{id}}$. This event corresponds to the tracing algorithm outputting a key that the adversary did not request (i.e., falsely implicating an honest user).

We say that an adversary $\mathcal{A}$ is admissible for the secure tracing experiment if the distinguisher D it outputs is efficiently-computable. A traceable PRF scheme TPRF satisfies *secure tracing* if for every $\lambda \in \mathbb{N}$, and every efficient and admissible adversary $\mathcal{A}$, and every inverse polynomial function $\varepsilon(\lambda) = 1/\mathsf{poly}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{BadTr}_{\mathcal{A},\varepsilon}] \leq \mathsf{negl}(\lambda) \text{ and } \Pr[\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] \geq \Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon}] - \mathsf{negl}(\lambda). \tag{2.1}$$

The first property states that the tracing algorithm cannot falsely implicate an honest user with non-negligible probability and the second property requires that the probability of the tracing algorithm correctly identifying at least one corrupt user be at least as high as the probability that the adversary outputs an ε-good distinguisher. We say that TPRF satisfies *strongly-secure tracing* if for every $\lambda \in \mathbb{N}$, every efficient and admissible adversary $\mathcal{A}$, and every inverse polynomial function $\varepsilon(\lambda) = 1/\mathsf{poly}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ where

$$\Pr[\mathsf{BadTr}_{\mathcal{A},\varepsilon}] \leq \mathsf{negl}(\lambda) \text{ and } \Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] \leq \mathsf{negl}(\lambda).$$

Remark 2.13 (Strong Tracing)**.** Strong tracing requires that the probability that the adversary outputs an ε-good distinguisher and yet, tracing fails, be negligible. This means that if the adversary outputs an ε-good distinguisher with non-negligible probability, then tracing succeeds with overwhelming probability. This is *not* required by the standard tracing definition. A simple calculation shows that strong tracing security implies standard secure tracing. First,

$$\Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon}] = \Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] + \Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}].$$

Strong secure tracing implies that $\Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] \leq \mathsf{negl}(\lambda)$. Thus,

$$\Pr[\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] \geq \Pr[\mathsf{CorrectTr}_{\mathcal{A},\varepsilon} \wedge \mathsf{GoodDis}_{\mathcal{A},\varepsilon}] \geq \Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon}] - \mathsf{negl}(\lambda).$$

Existing construction of traceable PRFs [GKWW21] all satisfy this stronger notion. In fact, the analysis of existing constructions show that $\Pr[\mathsf{CorrectTr}_{\mathcal{A},\varepsilon} \mid \mathsf{GoodDis}_{\mathcal{A},\varepsilon}] \geq 1 - \mathsf{negl}(\lambda)$; namely, whenever the adversary outputs a useful distinguisher, the tracing algorithm successfully recovers one of the identities.

Definition 2.14 (Secure Tracing against Active Adversaries)**.** We say a traceable PRF $\mathsf{TPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ satisfies secure tracing against active adversaries (i.e., is *actively secure*) if Definition 2.12 holds even if the adversary $\mathcal{A}$ in experiment $\mathsf{ExptTPRF}^{\mathsf{TPRF}}_{\mathcal{A},\varepsilon}$ has oracle access to a tracing oracle $\mathcal{O}(\mathsf{msk}, \cdot, \cdot)$ that takes as input the description of an efficiently-computable distinguisher D and the tracing parameter 1^z (encoded in unary) and outputs $\mathsf{Trace}^D(\mathsf{msk}, 1^z)$. We say TPRF satisfies secure tracing against Q-bounded active adversaries if Definition 2.12 holds against all efficient adversaries $\mathcal{A}$ that makes at most Q queries to the tracing oracle $\mathcal{O}(\mathsf{msk}, \cdot, \cdot)$ in $\mathsf{ExptTPRF}^{\mathsf{TPRF}}_{\mathcal{A},\varepsilon}$.

Remark 2.15 (Comparison with [GKWW21])**.** Definition 2.12 is slightly simpler than the corresponding definition from [GKWW21]. Namely, the definition in [GKWW21] required that for all efficient adversaries $\mathcal{A}$, every polynomial q, and non-negligible function ε, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ where $\varepsilon(\lambda) > 1/q(\lambda)$, Eq. (2.1) holds. Our formulation is equivalent; we refer to [Zha20, Remark 4] for a similar type of modification in the context of traitor tracing.

Remark 2.16 (Special Evaluation Queries)**.** The secure tracing definition from Goyal et al. [GKWW21] also allows the adversary to make *special evaluation queries* where the adversary can request evaluations on inputs $x \in \mathcal{X}$ under different identity keys. We do not focus on this setting since existing constructions of (non-collusion-resistant) traceable PRFs based on standard lattice assumptions do *not* support special evaluation queries. Special evaluation queries are not essential to realizing applications like traitor tracing from traceable PRFs.

3 Traceable PRF Constructions

In this section, we introduce our generic transformations (Constructions 3.3 and 3.9) for constructing traceable PRFs with active security and full collusion resistance (based on any single-key traceable PRF). In both of our constructions, we need an algorithm to estimate the success probability of a distinguisher. We use a standard approach based on Chernoff/Hoeffding bounds (Fact 2.1):

Definition 3.1 ($\mathsf{CheckDis}$)**.** Let λ be a security parameter, and let $\mathsf{TPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ be a traceable PRF with domain $\mathcal{X}$, range $\mathcal{Y}$, and identity space $[n]$. Given a distinguisher D, we define the algorithm $\mathsf{CheckDis}$:

- $\mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$: On input the master secret key msk, a parameter $z \in \mathbb{N}$, and given oracle access to a distinguisher D, the $\mathsf{CheckDis}$ algorithm proceeds as follows:
  - Let $N = \lambda z^2$. For each $i \in [N]$, sample $b_i \xleftarrow{\text{R}} \{0,1\}$, initialize an empty table T, and compute $b'_i \leftarrow D^{O_{b_i}(\mathsf{msk})}(1^\lambda)$, where the oracle O_{b_i} is implemented as follows:
    - If $b_i = 0$, sample $x \xleftarrow{\text{R}} \mathcal{X}$, compute $y \leftarrow \mathsf{Eval}(\mathsf{msk}, x)$, and output (x, y).
    - If $b_i = 1$, sample $x \xleftarrow{\text{R}} \mathcal{X}$ and check if there is already a mapping of the form $x \mapsto y$ in T. If so, output (x, y). Otherwise, sample $y \xleftarrow{\text{R}} \mathcal{Y}$, add (x, y) to T, and output (x, y).
  - Let t be the number of indices $i \in [N]$ where $b_i = b'_i$. If $t > N(1/2 + 1/(2z))$ occurs, then output 1. Otherwise, output 0.

Lemma 3.2 (Distinguisher Success Probability)**.** *Take any $z = z(\lambda)$. Let $\mathsf{TPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ be a traceable PRF, and sample $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$. Take any candidate distinguisher D, and let O_b be the weak PRF challenge oracle from Definition 2.12. Then the following properties hold:*

- *Suppose* $\Pr[D^{O_b(\mathsf{msk})}(1^\lambda) = b \colon b \xleftarrow{\text{R}} \{0,1\}] \geq 1/2 + 1/z$. *Then,* $\Pr[\mathsf{CheckDis}^D(\mathsf{msk}, 1^z) = 1] \geq 1 - \mathsf{negl}(\lambda)$.
- *Suppose* $\mathsf{CheckDis}^D(\mathsf{msk}, 1^{1/\varepsilon}) = 1$. *Then, with overwhelming probability over the randomness of* $\mathsf{CheckDis}$*, we have that* $\Pr[D^{O_b(\mathsf{msk})}(1^\lambda) = b \colon b \xleftarrow{\text{R}} \{0,1\}] \geq 1/2 + 1/(4z)$.

Proof. Both properties follow via Chernoff/Hoeffding bounds (Fact 2.1). □

3.1 Tracing Security with Active Adversaries

We first show how to generically transform any single-key traceable PRF satisfying strong tracing security into a single-key traceable PRF with strong tracing security against active adversaries (Definition 2.14).

Construction 3.3 (Actively Secure Single-Key Traceable PRF)**.** Let $\lambda \in \mathbb{N}$ be a security parameter. Let $\mathsf{TPRF}_0 = (\mathsf{Setup}_0, \mathsf{KeyGen}_0, \mathsf{Eval}_0, \mathsf{Trace}_0)$ be a secret-key traceable PRF with domain $\mathcal{X}$, range $\mathcal{Y}$ and identity space $[n]$. We construct a traceable PRF $\mathsf{TPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ with the same domain, range, and identity space as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{msk}$: On input the security parameter λ, the setup algorithm samples $\mathsf{msk} \leftarrow \mathsf{Setup}_0(1^\lambda)$.
- $\mathsf{Eval}(\mathsf{sk}, x) \to y$: Output $y \leftarrow \mathsf{Eval}_0(\mathsf{sk}, x)$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$: Output $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}_0(\mathsf{msk}, \mathsf{id})$.
- $\mathsf{Trace}^D(\mathsf{msk}, 1^z) \to T$: On input the master secret key msk and the parameter z, the tracing algorithm outputs $\varnothing$ if $\mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$ outputs 0. Otherwise, output $\mathsf{Trace}_0^D(\mathsf{msk}, 1^{4z})$.

Theorem 3.4 (Correctness and Weak Pseudorandomness)**.** *If* TPRF_0 *satisfies weak pseudorandomness (resp., key similarity), then* TPRF *in Construction 3.3 also satisfies weak pseudorandomness (resp., key similarity).*

Proof. This is immediate since Setup, KeyGen, and Eval simply invokes the corresponding algorithm in TPRF_0. □

Theorem 3.5 (Tracing Security)**.** *If* TPRF_0 *is a single-key strongly-secure traceable PRF, then* TPRF *in Construction 3.3 is a single-key strongly-secure traceable PRF with active security.*

Proof. Take any inverse polynomial function $\varepsilon(\lambda) = 1/\mathsf{poly}(\lambda)$ and any efficient adversary $\mathcal{A}$ for $\mathsf{ExptTPRF}_{\mathcal{A},\varepsilon}^{\mathsf{TPRF}}$. Let $Q = \mathsf{poly}(\lambda)$ be an upper bound on the number of queries that $\mathcal{A}$ makes. We proceed via a hybrid argument. Let Hyb_0 be the experiment $\mathsf{ExptTPRF}_{\mathcal{A},\varepsilon}^{\mathsf{TPRF}}(\lambda)$. For $i \in [Q]$, define Hyb_i and Hyb'_i as follows:

- Hyb_i: Same as $\mathsf{ExptTPRF}_{\mathcal{A},\varepsilon}^{\mathsf{TPRF}}(\lambda)$ except the first i queries $(D, 1^z)$ that $\mathcal{A}$ makes to the tracing oracle are answered using the following modified procedure:
  - If $\mathcal{A}$ has not made any key-generation queries, then output $\varnothing$.
  - Otherwise, suppose $\mathcal{A}$ has made a key-generation query on an identity $\mathsf{id} \in [n]$. Let $\mathsf{sk}_{\mathsf{id}}$ be the challenger's response to the key-generation query.
  - Compute $b \leftarrow \mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$, except use $\mathsf{Eval}(\mathsf{sk}_{\mathsf{id}}, x)$ in place of $\mathsf{Eval}(\mathsf{msk}, x)$ when simulating the oracle O_0.
  - If $b = 0$, output $\perp$. If $b = 1$, output $\{\mathsf{id}\}$.
- Hyb'_i: Same as Hyb_i except when using the modified procedure to respond to the i^{th} tracing query, the challenger computes $\mathsf{Eval}(\mathsf{msk}, x)$ in place of $\mathsf{Eval}(\mathsf{sk}_{\mathsf{id}}, x)$ when answering queries to O_0 (i.e., the challenger computes $b \leftarrow \mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$ when responding to the i^{th} tracing query). The first $i - 1$ tracing queries and the queries after the i^{th} query are handled as in Hyb_i.

Importantly, the modified procedure used in the first i queries of Hyb_i can be *implemented* entirely by the adversary itself (*without* needing to query the challenger). This means that an execution of Hyb_Q is equivalent to an execution of $\mathsf{ExptTPRF}_{\mathcal{A},\varepsilon}^{\mathsf{TPRF}}$ where $\mathcal{A}$ does not make any queries to the tracing oracle (the adversary can answer those queries itself). For an event E, we write $\mathsf{Hyb}_i[E]$ to denote the indicator random variable that is 1 if event E occurs in an execution of Hyb_i and 0 otherwise. We define $\mathsf{Hyb}'_i[E]$ analogously. In the following, we will consider events E that are functions of the master secret key msk and the adversary's view of the experiment (i.e., the event E can be expressed as a function of msk, the adversary's queries, the challenger's responses, and the adversary's output in the experiment).

Lemma 3.6. *If TPRF_0 satisfies key indistinguishability, then for all adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathsf{Hyb}'_i[E] = 1] - \Pr[\mathsf{Hyb}_i[E] = 1]\right| \le \mathsf{negl}(\lambda).$$

Proof. The only difference between Hyb'_i and Hyb_i is in the challenger's response to the i^{th} tracing query. Specifically, in Hyb'_i, the challenger computes $y \leftarrow \mathsf{Eval}_0(\mathsf{msk}, x)$ when simulating the behavior of O_0 while in Hyb_i, the challenger computes $y \leftarrow \mathsf{Eval}_0(\mathsf{sk}_{\mathsf{id}}, x)$ when simulating the behavior of O_0 in the computation of $\mathsf{CheckDis}$. In both experiments, the challenger samples $\mathsf{msk} \leftarrow \mathsf{Setup}_0(1^\lambda)$ and $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}_0(\mathsf{msk}, \mathsf{id})$. Let $M = \mathsf{poly}(\lambda)$ be a bound on the number of queries to O_0 that the challenger simulates when responding to the i^{th} tracing query. Let $x_1, \ldots, x_M \xleftarrow{\text{R}} \mathcal{X}$ be the domain elements the challenger samples when responding to O_0 queries. By key-indistinguishability of TPRF_0, we have that $\Pr[\mathsf{Eval}_0(\mathsf{sk}_{\mathsf{id}}, x_j) \neq \mathsf{Eval}_0(\mathsf{msk}, x_j)] \le \mathsf{negl}(\lambda)$ for all $j \in [M]$. By a union bound, the probability that there exists $j \in [M]$ such that $\mathsf{Eval}_0(\mathsf{sk}_{\mathsf{id}}, x_j) \neq \mathsf{Eval}_0(\mathsf{msk}, x_j)$ is negligible. Thus, with overwhelming probability, the adversary's view in Hyb'_i and Hyb_i is identically distributed. □

Lemma 3.7. *If TPRF_0 satisfies single-key strongly secure tracing, then for all efficient adversaries $\mathcal{A}$, $\left|\Pr[\mathsf{Hyb}_{i-1}[E] = 1] - \Pr[\mathsf{Hyb}'_i[E] = 1]\right| \le \mathsf{negl}(\lambda)$.*

Proof. Suppose that $\left|\Pr[\mathsf{Hyb}_{i-1}[E] = 1] - \Pr[\mathsf{Hyb}'_i[E] = 1]\right| = \varepsilon'$ for some non-negligible ε'. The only difference between Hyb_{i-1} and Hyb'_i is the challenger's behavior in response to the i^{th} tracing query. Let $(D_i, 1^{z_i})$ be the i^{th} tracing query $\mathcal{A}$ makes in an execution of Hyb_{i-1} and Hyb'_i. In Hyb_{i-1}, the challenger responds with $\mathsf{Trace}^{D_i}(\mathsf{msk}, 1^{z_i})$. In Hyb'_i, the challenger uses the modified procedure described above. This is the only difference between Hyb_{i-1} and Hyb'_i. Thus, if the outputs of $\mathsf{Hyb}_{i-1}[E]$ and $\mathsf{Hyb}'_i[E]$ differ by ε', one of the following two events must occur with probability at least ε':

- E_1: Algorithm $\mathcal{A}$ has not made any key-generation queries and $\mathsf{Trace}^{D_i}(\mathsf{msk}, 1^{z_i}) \neq \varnothing$. In this case, the challenger in Hyb_{i-1} responds with $\mathsf{Trace}^{D_i}(\mathsf{msk}, 1^{z_i})$ while the challenger in Hyb'_i responds with $\varnothing$.

- E_2: Algorithm $\mathcal{A}$ has made a key-generation query on an identity $\mathsf{id} \in [n]$, $\mathsf{CheckDis}^{D_i}(\mathsf{msk}, 1^{z_i})$ outputs 1, and $\mathsf{Trace}_0^{D_i}(\mathsf{msk}, 1^{4z_i}) \neq \{\mathsf{id}\}$. In this case, the challenger in Hyb_{i-1} responds with $\mathsf{Trace}_0^{D_i}(\mathsf{msk}, 1^{4z_i})$ while the challenger in Hyb'_i responds with $\{\mathsf{id}\}$.

In all other cases, the challenger's response in the two experiments is identical. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for experiment $\mathsf{ExptTPRF}^{\mathsf{TPRF}_0}_{\mathcal{B},1/(4z_i)}$:

1. Whenever algorithm $\mathcal{A}$ makes an evaluation query $\mathsf{Eval}(\mathsf{msk}, \cdot)$ or a key-generation query $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$, algorithm $\mathcal{B}$ forwards the query to the secure tracing challenger and forwards the response to $\mathcal{A}$.

2. For the first $i - 1$ queries to the tracing oracle, algorithm $\mathcal{B}$ responds using the modified procedure in Hyb_{i-1} and Hyb'_i. This procedure only depends on quantities that are known to the challenger.

3. When algorithm $\mathcal{A}$ makes its i^{th} query D_i to the tracing oracle, algorithm $\mathcal{B}$ aborts the simulation and outputs D_i as its distinguisher.

By construction, algorithm $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in an execution of Hyb_{i-1} and Hyb'_i (up to the point where $\mathcal{A}$ submits its i^{th} tracing query). Thus, either event E_1 or E_2 must happen with probability at least $\varepsilon'/2$. We now consider each possibility:

- Suppose $\Pr[\mathsf{E}_1] \ge \varepsilon'/2$. If E_1 occurs, this means that algorithm $\mathcal{B}$ has not made any key-generation queries and $\mathsf{Trace}^{D_i}(\mathsf{msk}, 1^{z_i})$ outputs a non-empty set. This is only possible if $\mathsf{Trace}_0^{D_i}(\mathsf{msk}, 1^{4z_i})$ outputs a non-empty set, in which case, the event $\mathsf{BadTr}_{\mathcal{B},1/(4z_i)}$ occurs. This means that $\Pr[\mathsf{BadTr}_{\mathcal{B},1/(4z_i)}] \ge \Pr[\mathsf{E}_1] \ge \varepsilon'/2$, which is non-negligible. Correspondingly, $\mathcal{B}$ breaks the secure tracing property of TPRF_0.

- Suppose $\Pr[\mathsf{E}_2] \geq \varepsilon'/2$. If E_2 occurs, then $\mathsf{CheckDis}^{D_i}(\mathsf{msk}, 1^{z_i})$ outputs 1. By Lemma 3.2, this means that with probability $1 - \mathsf{negl}(\lambda)$, $\Pr\left[D_i^{\mathcal{O}_b}(1^\lambda) = b : b \xleftarrow{\mathrm{R}} \{0,1\}\right] \geq 1/2 + 1/(4z_i)$, so event $\mathsf{GoodDis}_{\mathcal{B},1/(4z_i)}$ occurs. Moreover, if E_2 occurs, then $\mathsf{Trace}_0^{D_i}(\mathsf{msk}, 1^{4z_i}) \neq \{\mathsf{id}\}$. Since the only identity queried by $\mathcal{B}$ is id, this means that $\mathsf{CorrectTr}_{\mathcal{B},1/(4z_i)}$ did not occur. Thus,

  $$\Pr[\mathsf{GoodDis}_{\mathcal{B},1/(4z_i)} \wedge \neg\mathsf{CorrectTr}_{\mathcal{B},1/(4z_i)}] \geq \Pr[\mathsf{E}_2] - \mathsf{negl}(\lambda) \geq \varepsilon'/2 - \mathsf{negl}(\lambda),$$

  which is non-negligible. Thus, $\mathcal{B}$ breaks the strong tracing property of TPRF_0.

In both cases, $\mathcal{B}$ breaks strong tracing property of TPRF_0. □

Lemma 3.8. *If TPRF_0 satisfies strong secure tracing, then for all efficient adversaries $\mathcal{A}$ in Hyb_Q, and all inverse polynomials $\varepsilon = 1/\mathsf{poly}(\lambda)$,*

$$\Pr[\mathsf{BadTr}_{\mathcal{A},\varepsilon}] \leq \mathsf{negl}(\lambda) \ \textit{and} \ \Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] \leq \mathsf{negl}(\lambda), \tag{3.1}$$

where the events $\mathsf{BadTr}_{\mathcal{A},\varepsilon}$, $\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}$, and $\mathsf{GoodDis}_{\mathcal{A},\varepsilon}$ are as defined in Definition 2.12.

Proof. Suppose there is an adversary $\mathcal{A}$ in Hyb_Q where either

$$\Pr[\mathsf{BadTr}_{\mathcal{A},\varepsilon}] = \varepsilon' \ \text{or} \ \Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] = \varepsilon', \tag{3.2}$$

for some non-negligible ε'. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for $\mathsf{ExptTPRF}^{\mathsf{TPRF}_0}_{\mathcal{B},\varepsilon/4}$. Algorithm $\mathcal{B}$ simulates an execution of Hyb_Q for $\mathcal{A}$ as follows:

1. Whenever $\mathcal{A}$ makes a key-generation or evaluation query, it forwards the query to its challenger and forwards the challenger's response to $\mathcal{A}$. Whenever $\mathcal{A}$ makes a tracing query, algorithm $\mathcal{B}$ answers the query using the modified tracing procedure in Hyb_Q.
2. After algorithm $\mathcal{A}$ outputs a distinguisher D, algorithm $\mathcal{B}$ outputs the same distinguisher D.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_Q for $\mathcal{A}$, so Eq. (3.2) holds. We consider the two possibilities:

- Suppose $\Pr[\mathsf{BadTr}_{\mathcal{A},\varepsilon}] = \varepsilon'$. By definition of Trace, this means that $\mathsf{Trace}_0^D(\mathsf{msk}, 1^{4/\varepsilon})$ outputs a set T where $T \not\subseteq S_{\mathsf{id}}$, where S_{id} is the set of identities algorithm $\mathcal{A}$ submitted to the key-generation oracle. This means that event $\mathsf{BadTr}_{\mathcal{B},\varepsilon/4}$ occurs in $\mathsf{ExptTPRF}^{\mathsf{TPRF}_0}_{\mathcal{B},\varepsilon/4}$ with non-negligible probability ε'.

- Suppose $\Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] = \varepsilon'$. Let S_{id} be the set of identities $\mathcal{A}$ (and by construction, $\mathcal{B}$) submits to the key-generation oracle. Let $T \leftarrow \mathsf{Trace}^D(\mathsf{msk}, 1^{1/\varepsilon})$. Suppose events $\mathsf{GoodDis}_{\mathcal{A},\varepsilon}$ and $\neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}$ occur. By Lemma 3.2, this means that $\mathsf{CheckDis}^D(\mathsf{msk}, 1^{1/\varepsilon})$ outputs 1 with overwhelming probability. In this case, the output T from $\mathsf{Trace}^D(\mathsf{msk}, 1^{1/\varepsilon})$ is computed by evaluating $T \leftarrow \mathsf{Trace}_0^D(\mathsf{msk}, 1^{4/\varepsilon})$. Since $\neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}$ occurs, either $T = \varnothing$ or $T \not\subseteq S_{\mathsf{id}}$. This means that $\neg\mathsf{CorrectTr}_{\mathcal{B},\varepsilon/4}$ also occurs.

  Moreover, if $\mathsf{GoodDis}_{\mathcal{A},\varepsilon}$ occurs, then $\Pr[D^{\mathcal{O}_b(\mathsf{msk})}(1^\lambda) = b : b \xleftarrow{\mathrm{R}} \{0,1\}] \geq 1/2 + \varepsilon$. By construction, since $\mathsf{Eval}(\mathsf{sk}, \cdot)$ is implemented by $\mathsf{Eval}_0(\mathsf{sk}, \cdot)$, this means that event $\mathsf{GoodDis}_{\mathcal{B},\varepsilon/4}$ also occurs. Taken together, we conclude that

  $$\begin{aligned}\Pr[\mathsf{GoodDis}_{\mathcal{B},\varepsilon/4} \wedge \neg\mathsf{CorrectTr}_{\mathcal{B},\varepsilon/4}] &\geq \Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] - \mathsf{negl}(\lambda) \\ &= \varepsilon' - \mathsf{negl}(\lambda),\end{aligned}$$

  which is non-negligible and breaks strong secure tracing of TPRF_0. □

Combining Lemmas 3.6 to 3.8, and applying them to the events $\mathsf{BadTr}_{\mathcal{A},\varepsilon}$ and $(\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon})$, we have that Eq. (3.1) also holds in $\mathsf{Hyb}_0 \equiv \mathsf{ExptTPRF}^{\mathsf{TPRF}}_{\mathcal{A},\varepsilon}$. Thus, TPRF from Construction 3.3 satisfies strong secure tracing with active adversaries. □

3.2 Collusion Resistant Traceable PRFs

We now introduce our main construction of a fully collusion resistant traceable PRF from any single-key traceable PRF (in conjunction with a fingerprinting code). We refer to Section 1.1 for an overview of the construction.

Construction 3.9 (Collusion Resistant Traceable PRF)**.** Let $\lambda \in \mathbb{N}$ be a security parameter and $n = n(\lambda)$ be the number of identities. Our construction relies on the following ingredients:

- Let $\mathsf{TPRF}_{\mathsf{nc}} = (\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Setup}, \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{KeyGen}, \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}, \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace})$ be a (single-key) secret-key traceable PRF with domain $\mathcal{X}$, range $\{0,1\}^\rho$ and identity space $\{0,1\}$.
- Let $\mathsf{FC} = (\mathsf{FC}.\mathsf{Gen}, \mathsf{FC}.\mathsf{Trace})$ be a fingerprinting code with n codewords and code length ℓ.
- Let $\mathcal{R}$ be the randomness space for $\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{KeyGen}$ and let $\mathsf{PRF} \colon \mathcal{K} \times ([\ell] \times \{0,1\}) \to \mathcal{R}$ be a pseudorandom function (with key-space $\mathcal{K}$ and domain $[\ell] \times \{0,1\}$).

We construct a fully collusion resistant secret-key traceable PRF $\mathsf{TPRF} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Trace})$ with domain $\mathcal{X}$, range $\{0,1\}^\rho$, and identity space $[n]$ as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{msk}$: On input the security parameter λ, the setup algorithm starts by sampling $\mathsf{msk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Setup}(1^\lambda)$ for each $i \in [\ell]$. In addition, it samples $k \xleftarrow{\text{R}} \mathcal{K}$ and $(\mathsf{tk}_{\mathsf{FC}}, \Gamma) \leftarrow \mathsf{FC}.\mathsf{Gen}(1^\lambda)$. It outputs $\mathsf{msk} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell, \Gamma, \mathsf{tk}_{\mathsf{FC}}, k)$.
- $\mathsf{Eval}(\mathsf{sk}, x) \to y$: On input a secret key $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell, \Gamma, \mathsf{tk}_{\mathsf{FC}}, k)$ and an input $x \in \mathcal{X}$, the evaluation algorithm computes $y_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{sk}_i, x)$ for each $i \in [\ell]$, and outputs $y \leftarrow \bigoplus_{i \in [\ell]} y_i$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$: On input the master secret key $\mathsf{msk} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell, \Gamma = \{\bar{w}^{(i)}\}_{i \in [n]}, \mathsf{tk}_{\mathsf{FC}}, k)$ and an identity $\mathsf{id} \in [n]$, the key-generation algorithm computes randomness $r_i \leftarrow \mathsf{PRF}(k, (i, \bar{w}_i^{(\mathsf{id})}))$ and samples $\mathsf{sk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{KeyGen}(\mathsf{msk}_i, \bar{w}_i^{(\mathsf{id})}; r_i)$ for each $i \in [\ell]$. It outputs $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell, \perp, \perp, \perp)$.[5]
- $\mathsf{Trace}^D(\mathsf{msk}, 1^z) \to T$: On input the master secret key $\mathsf{msk} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell, \Gamma, \mathsf{tk}_{\mathsf{FC}}, k)$ and the parameter z, the tracing algorithm proceeds as follows:
  - If $\mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$ outputs 0, output $\varnothing$.
  - Otherwise, define the oracle-aided distinguisher D_i^O as follows:
    - On input the security parameter λ, start running algorithm $D^{O'}(1^\lambda)$.
    - Whenever D makes a query to its oracle O', the distinguisher D_i makes a query to its own oracle O to obtain a sample (x, y). Algorithm D_i computes $y' \leftarrow y \oplus \Big(\bigoplus_{j \neq i} \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_j, x)\Big)$ and replies to D with the sample (x, y').
  - For each $i \in [\ell]$, run $T_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace}^{D_i}(\mathsf{msk}_i, 1^{4z})$. If $0 \in T_i$, set $\bar{w}_i^* = 0$; otherwise, set $\bar{w}_i^* = 1$.
  - Output $\mathsf{FC}.\mathsf{Trace}(\mathsf{tk}_{\mathsf{FC}}, \bar{w}^*)$.

Theorem 3.10 (Weak Pseudorandomness)**.** *If $\mathsf{TPRF}_{\mathsf{nc}}$ satisfies weak pseudorandomness, then TPRF in Construction 3.9 also satisfies weak pseudorandomness.*

Proof. This follows from the fact that xor-ing the outputs of a (weak) PRF preserves (weak) pseudorandomness. More formally, suppose there exists an efficient adversary $\mathcal{A}$ that breaks weak pseudorandomness of Construction 3.9. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the weak pseudorandomness of $\mathsf{TPRF}_{\mathsf{nc}}$ as follows:

1. For $i \in [\ell - 1]$, algorithm $\mathcal{B}$ samples a key $\mathsf{msk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Setup}(1^\lambda)$.

[5] The $\perp$'s are added so that msk and sk have the same format (and can both be used as an input to the evaluation algorithm).

2. Whenever $\mathcal{A}$ makes an oracle query, algorithm $\mathcal{B}$ queries its own oracle to obtain an output (x, y). It compute $y' \leftarrow y \oplus \left(\bigoplus_{i \in [\ell-1]} \mathsf{Eval}(\mathsf{msk}_i, x)\right)$ and replies to $\mathcal{A}$ with (x, y).

The weak PRF challenger is used to simulate the evaluations of the ℓ^{th} copy of $\mathsf{TPRF}_{\mathsf{nc}}$. If the challenger replies with PRF evaluations, then $\mathcal{B}$ perfectly simulates the pseudorandom distribution for $\mathcal{A}$ while if the challenger replies with uniform random value, then $\mathcal{B}$ perfectly simulates the truly random distribution. □

Theorem 3.11 (Key Similarity)**.** *If* $\mathsf{TPRF}_{\mathsf{nc}}$ *satisfies key similarity, then* TPRF *in Construction 3.9 also satisfies key similarity.*

Proof. Take any identity $\mathsf{id} \in [n]$, and sample $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$, $x \xleftarrow{\text{R}} \mathcal{X}$. In this case, $\mathsf{msk} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell, \Gamma, \mathsf{tk}_{\mathsf{FC}}, k)$ where $\mathsf{msk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{sk}_{\mathsf{id}} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell, \perp, \perp, \perp)$ where $\mathsf{sk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{KeyGen}(\mathsf{msk}_i, \mathsf{id})$. Key similarity of $\mathsf{TPRF}_{\mathsf{nc}}$ implies that

$$\Pr[\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_i, x) \neq \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{sk}_i, x)] \leq \mathsf{negl}(\lambda).$$

By a union bound, with probability $1 - \mathsf{negl}(\lambda)$, $\mathsf{Eval}(\mathsf{msk}_i, x) = \mathsf{Eval}(\mathsf{sk}_i, x)$ for all $i \in [\ell]$, and the claim follows. □

Theorem 3.12 (Tracing Security)**.** *Let $Q = Q(\lambda)$ be an arbitrary polynomial. If* $\mathsf{TPRF}_{\mathsf{nc}}$ *is a strongly-secure single-key traceable PRF with security against Q-bounded active adversaries,* FC *is a fully collusion resistant fingerprinting code in the presence of Q tracing queries, and* PRF *is a secure PRF, then Construction 3.9 is a fully collusion resistant strongly-secure traceable PRF with security against Q-bounded active adversaries. If* $\mathsf{TPRF}_{\mathsf{nc}}$ *and* FC *are both secure against adversaries that can make an unbounded number of tracing queries, then the same holds for Construction 3.9.*

Proof. Fix a security parameter $\lambda \in \mathbb{N}$ and take any inverse polynomial function $\varepsilon(\lambda) = 1/\mathsf{poly}(\lambda)$. Consider an execution of experiment $\mathsf{ExptTPRF}_{\mathcal{A},\varepsilon}^{\mathsf{TPRF}}$. We now define the following sequence of hybrid experiments:

- Hyb_0: This is the real security experiment $\mathsf{ExptTPRF}_{\mathcal{A},\varepsilon}^{\mathsf{TPRF}}(\lambda)$.
- Hyb_1: Same as Hyb_0 except the challenger samples $f \xleftarrow{\text{R}} \mathsf{Funs}[[\ell] \times \{0,1\}, \mathcal{R}]$ and computes $f(\cdot)$ instead of $\mathsf{PRF}(k, \cdot)$.
- Hyb_2: Same as Hyb_1, except on every tracing query and at the end of the game when $\mathcal{A}$ outputs its distinguisher, the experiment additionally checks the following two conditions. Let $(D, 1^z)$ be the distinguisher and tracing parameter the adversary submits in its tracing query (or outputs at the end of the experiment[6]).
  - $\mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$ outputs 1.
  - The word $\bar{w}^* \in \{0,1\}^\ell$ computed by $\mathsf{Trace}^D(\mathsf{msk}, 1^z)$ satisfies $\bar{w}^* \notin F(W)$, where $F(W)$ is the feasible set of $W = \{\bar{w}^{(\mathsf{id}_j)}\}_{j \in [Q]}$, $\mathsf{id}_1, \ldots, \mathsf{id}_Q \in [n]$ are the identities $\mathcal{A}$ submitted to the key-generation oracle *prior* to outputting D, and $\Gamma = \{\bar{w}^{(i)}\}_{i \in [n]}$ is the set of codewords sampled by Setup.

  If both conditions hold, then the experiment sets the Bad flag and aborts with output $\perp$.

For an event E, we write $\mathsf{Hyb}_i[E]$ to denote the indicator random variable that is 1 if event E occurs in an execution of Hyb_i and 0 otherwise. In the following, we will consider events E that are functions of the "experiment's messages:" these include the adversary's queries, the challenger's responses, and the adversary's output in the experiment.

Lemma 3.13. *Let E be an efficiently-checkable event that is a function of* $(\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell, \Gamma, \mathsf{tk}_{\mathsf{FC}})$ *and the experiment's messages in* $\mathsf{ExptTPRF}_{\mathcal{A},\varepsilon}^{\mathsf{TPRF}}$*. If* PRF *is secure, then for all efficient adversaries $\mathcal{A}$, we have that* $|\Pr[\mathsf{Hyb}_0[E] = 1] - \Pr[\mathsf{Hyb}_1[E] = 1]| \leq \mathsf{negl}(\lambda)$.

[6] The tracing parameter for the final output is set to be $z = 1/\varepsilon$

Proof. Suppose there exists efficient $\mathcal{A}$ where $|\Pr[\mathsf{Hyb}_0[E] = 1] - \Pr[\mathsf{Hyb}_1[E] = 1]| \geq \varepsilon'$ for some non-negligible ε'. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks security of PRF as follows:

1. Algorithm $\mathcal{B}$ starts by sampling a key $\mathsf{msk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Setup}(1^\lambda)$ for each $i \in [\ell]$. It also samples $(\mathsf{tk}_{\mathsf{FC}}, \Gamma = \{\bar{w}^{(i)}\}_{i \in [\ell]}) \leftarrow \mathsf{FC}.\mathsf{Gen}(1^\lambda)$.

2. Algorithm $\mathcal{B}$ starts running $\mathcal{A}$. Whenever $\mathcal{A}$ makes an evaluation or a trace query, algorithm $\mathcal{B}$ responds according to the specification of the real scheme (Construction 3.9). Observe that neither of these queries depend on the PRF key k.

3. When $\mathcal{A}$ makes a key-generation query on an identity $\mathsf{id} \in [n]$, algorithm $\mathcal{B}$ queries the PRF challenger on input $(i, \bar{w}_i^{(\mathsf{id})})$ and obtains output $r_i \in \mathcal{R}$ for each $i \in [\ell]$. It then computes $\mathsf{sk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{KeyGen}\big(\mathsf{msk}_i, \bar{w}_i^{(\mathsf{id})}; r_i\big)$ and replies to $\mathcal{A}$ with $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell, \perp, \perp, \perp)$.

4. At the end of the game, algorithm $\mathcal{B}$ outputs 1 if event E occurs and 0 otherwise.

Algorithm $\mathcal{B}$ is efficient since deciding E can be efficiently computed as a function of $(\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell, \Gamma, \mathsf{tk}_{\mathsf{FC}})$ and the experiment's messages. All of these quantities are known to $\mathcal{B}$. By construction, if $r_i \leftarrow \mathsf{PRF}(k, (i, \bar{w}_i^{(\mathsf{id})}))$ where $k \xleftarrow{\text{R}} \mathcal{K}$, then $\mathcal{B}$ perfectly simulates the distribution in Hyb_0. If $r_i \leftarrow f(i, \bar{w}_i^{(\mathsf{id})})$ where $f \xleftarrow{\text{R}} \mathsf{Funs}[[\ell] \times \{0,1\}, \mathcal{R}]$, then $\mathcal{B}$ perfectly simulates the distribution in Hyb_1. Thus, algorithm $\mathcal{B}$'s distinguishing advantage is exactly ε'. □

Lemma 3.14. *Let E be any event that depends only on the experiment's messages and* msk. *If* $\mathsf{TPRF}_{\mathsf{nc}}$ *is a strongly secure single-key traceable PRF with security against Q-bounded active adversaries, then for all efficient Q-bounded active adversaries $\mathcal{A}$,* $|\Pr[\mathsf{Hyb}_1[E] = 1] - \Pr[\mathsf{Hyb}_2[E] = 1]| \leq \mathsf{negl}(\lambda)$.

Proof. Suppose there exists efficient $\mathcal{A}$ where $|\Pr[\mathsf{Hyb}_1[E] = 1] - \Pr[\mathsf{Hyb}_2[E] = 1]| = \varepsilon'$ for some non-negligible ε'. Since the only difference between Hyb_1 and Hyb_2 is the additional checks in Hyb_2, it must be the case that in an execution of Hyb_1 or Hyb_2, algorithm $\mathcal{A}$ outputs a distinguisher D and a tracing parameter 1^z (either as part of a tracing query or at the end of the experiment) that causes Hyb_2 to set the Bad flag. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ for experiment $\mathsf{ExptTPRF}_{\mathcal{B},1/(4z)}^{\mathsf{TPRF}_{\mathsf{nc}}}$:

1. Algorithm $\mathcal{B}$ begins by sampling an index $i^* \xleftarrow{\text{R}} [\ell]$ and a bit $b^* \xleftarrow{\text{R}} \{0,1\}$. It makes a key-generation query to its challenger on the bit b^* to obtain a key sk^*.

2. For all $i \neq i^*$, algorithm $\mathcal{B}$ samples a key $\mathsf{msk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Setup}(1^\lambda)$. It also samples $(\mathsf{tk}_{\mathsf{FC}}, \Gamma = \{\bar{w}^{(i)}\}_{i \in [\ell]}) \leftarrow \mathsf{FC}.\mathsf{Gen}(1^\lambda)$.

3. Algorithm $\mathcal{B}$ initializes an empty table T and starts running $\mathcal{A}$. Whenever $\mathcal{A}$ makes an oracle query, algorithm $\mathcal{B}$ responds as follows:

   - **Evaluation queries:** On input $x \in \mathcal{X}$, algorithm $\mathcal{B}$ computes $y_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_i, x)$ for all $i \neq i^*$. It makes an evaluation query to its challenger on input x to obtain a value $y_{i^*} \in \{0,1\}^\rho$. It replies to $\mathcal{A}$ with $y_{i^*} \oplus \big(\bigoplus_{i \neq i^*} y_i\big)$.

   - **Key-generation queries:** If $\bar{w}_{i^*}^{(\mathsf{id})} \neq b^*$, then algorithm $\mathcal{B}$ aborts the experiments and outputs $\perp$. Otherwise, algorithm $\mathcal{B}$ sets $\mathsf{sk}_{i^*} \leftarrow \mathsf{sk}^*$. Next, for each $i \neq i^*$, $\mathcal{B}$ checks if there exists a mapping $(i, \bar{w}_i^{(\mathsf{id})}) \mapsto r_{i, \bar{w}_i^{(\mathsf{id})}}$ in T. If not, it samples a random $r_{i, \bar{w}_i^{(\mathsf{id})}} \xleftarrow{\text{R}} \mathcal{R}$ and adds $(i, \bar{w}_i^{(\mathsf{id})}) \mapsto r_{i, \bar{w}_i^{(\mathsf{id})}}$ to T. Algorithm $\mathcal{B}$ then computes $\mathsf{sk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{KeyGen}(\mathsf{msk}_i, \bar{w}_i^{(\mathsf{id})}; r_{i, \bar{w}_i^{(\mathsf{id})}})$ for each $i \neq i^*$. Algorithm $\mathcal{B}$ gives $\mathcal{A}$ the tuple $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell, \perp, \perp, \perp)$.

   - **Tracing queries:** On input a distinguisher D and a tracing parameter 1^z, algorithm $\mathcal{B}$ starts by computing $\mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$, where it uses the procedure for simulating evaluation queries to compute $\mathsf{Eval}(\mathsf{msk}, x)$ in $\mathsf{CheckDis}$. If $\mathsf{CheckDis}$ outputs 0, then $\mathcal{B}$ replies to $\mathcal{A}$ with $\varnothing$.

Otherwise, if $\mathsf{CheckDis}$ outputs 1, for $i \neq i^*$, algorithm $\mathcal{B}$ computes T_i by emulating an execution of $\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace}^{D_i}(\mathsf{msk}_i, 1^{4z})$. Whenever $\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace}^{D_i}$ makes a query to the oracle-aided algorithm D_i^O, algorithm $\mathcal{B}$ implements the logic as follows:

  - When $\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace}$ makes an oracle query to D_i on input 1^λ, algorithm $\mathcal{B}$ starts running $D^{O'}(1^\lambda)$.
  - When D makes an oracle query to its oracle O', algorithm $\mathcal{B}$ treats it as if D_i made a query to O, and computes the oracle's response (x, y) according to the specification in $\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace}$. Algorithm $\mathcal{B}$ then makes an evaluation query to its challenger on input x to receive y_{i^*} and computes $y' \leftarrow y \oplus \big(\bigoplus_{j \neq i, i^*} \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_j, x)\big) \oplus y_{i^*}$, and gives the pair (x, y') to D as the response from O'.

Let T_i be the output of this emulated execution of $\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace}^{D_i}(\mathsf{msk}_i, 1^{4z})$. Next, to compute T_{i^*}, algorithm $\mathcal{B}$ defines the oracle-aided algorithm $D_{i^*}^O$ according to the specification of the real tracing algorithm:

  - On input the security parameter λ, run $D^{O'}(1^\lambda)$.
  - Whenever D makes a query to its oracle O', the distinguisher D_{i^*} makes a query to its own oracle to obtain a sample (x, y). Algorithm D_{i^*} computes $y' \leftarrow y \oplus \big(\bigoplus_{j \neq i^*} \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_j, x)\big)$ and replies to D with the same (x, y').

Algorithm $\mathcal{B}$ submits a tracing query on $D_{i^*}^O$ to its challenger to obtain a set T_{i^*}. If $T_{i^*} \neq \{b^*\}$, algorithm $\mathcal{B}$ halts and outputs $D_{\mathsf{nc}} = D_{i^*}$. Otherwise, for each $i \in [\ell]$, if $0 \in T_i$, it sets $\bar{w}_i^* = 0$, and otherwise, it sets $\bar{w}_i^* = 1$. It replies to $\mathcal{A}$ with $\mathsf{FC}.\mathsf{Trace}(\mathsf{tk}_{\mathsf{FC}}, \bar{w}^*)$.

4. After $\mathcal{A}$ has finished making queries (and assuming $\mathcal{B}$ has not yet aborted), then $\mathcal{A}$ outputs a distinguisher D. Algorithm $\mathcal{B}$ constructs the oracle-aided algorithm D_{nc}^O using the same procedure as in $D_{i^*}^O$ in the above description for simulating tracing queries. Finally, it outputs D_{nc} as its distinguisher.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of experiment Hyb_1 for $\mathcal{A}$ unless $\mathcal{A}$ makes a key-generation query that causes $\mathcal{B}$ to abort. By assumption, in an execution of Hyb_1, algorithm $\mathcal{A}$ will output a distinguisher D and a tracing parameter 1^z (either as part of a tracing query or at the end of the experiment) that satisfies the following properties:

- $\mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$ outputs 1.
- Let $\mathsf{id}_1, \ldots, \mathsf{id}_q \in [n]$ be the identities $\mathcal{A}$ submitted to the key-generation oracle prior to outputting $(D, 1^z)$ and let $\Gamma = \{\bar{w}^{(i)}\}_{i \in [n]}$ be the set of codewords $\bar{w}^{(i)} \in \{0,1\}^\ell$ sampled by Setup. Then, there exists an index $j \in [\ell]$ with the following two properties:
  - $\bar{w}_j^{(\mathsf{id}_i)} = \bar{w}_j^{(\mathsf{id}_1)}$ for all $i \in [q]$; and
  - $T_j \neq \{\bar{w}_j^{(\mathsf{id}_1)}\}$, where $T_j \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace}^{D_j}(\mathsf{msk}_j, 1^{4z})$, and D_j is the oracle-aided distinguisher as defined in Construction 3.9.

Algorithm $\mathcal{B}$ samples the index i^* and the bit b^* uniformly at random (and independently of the view of the adversary). Observe that if $i^* = j$ and $b^* = \bar{w}_j^{(\mathsf{id}_1)}$, algorithm $\mathcal{B}$ does not abort the simulation, and instead, outputs the distinguisher D_{i^*}. Thus, with probability at least $\varepsilon'/(2\ell)$, algorithm $\mathcal{B}$ does not abort and successfully outputs a distinguisher D_{nc}. We now argue that this implies events $\mathsf{GoodDis}_{\mathcal{B},1/(4z)}$ and $\neg\mathsf{CorrectTr}_{\mathcal{B},1/(4z)}$. In the following, we write msk_{i^*} to denote the master secret key sampled by the challenger in $\mathsf{ExptTPRF}_{\mathcal{B},1/(4z)}^{\mathsf{TPRF}_{\mathsf{nc}}}$. By construction, algorithm $\mathcal{B}$ simulates an execution of Hyb_1 with $\mathsf{msk} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell, \Gamma, \mathsf{tk}_{\mathsf{FC}}, \bot)$.

- Let $O_{\mathcal{A},b}(\mathsf{msk})$ and $O_{\mathcal{B},b}(\mathsf{msk}_{i^*})$ be the weak PRF challenge oracles from Definition 2.12 in $\mathsf{ExptTPRF}_{\mathcal{A},\varepsilon}^{\mathsf{TPRF}}$ and $\mathsf{ExptTPRF}_{\mathcal{B},1/(4z)}^{\mathsf{TPRF}_{\mathsf{nc}}}$, respectively. Since $\mathsf{CheckDis}^D(\mathsf{msk}, 1^z) = 1$, we appeal to Lemma 3.2 and conclude that with probability $1 - \mathsf{negl}(\lambda)$,

$$\Pr[D^{O_{\mathcal{A},b}}(1^\lambda) = b : b \xleftarrow{\text{R}} \{0,1\}] \geq 1/2 + 1/(4z).$$

Consider the probability that $\Pr[D_{\mathsf{nc}}^{O_{\mathcal{B},b}(\mathsf{msk}_{i^*})}(1^\lambda) = b : b \xleftarrow{\text{R}} \{0,1\}]$. By construction, for any oracle O, $D_{\mathsf{nc}}^{O}(1^\lambda)$ outputs $D^{O'}(1^\lambda)$, where D_{nc} simulates oracles queries to O' by issuing a query to O to obtain (x, y), computing $y' \leftarrow y \oplus \big(\bigoplus_{j \neq i^*} \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_j, x)\big)$, and replying with (x, y'). We claim that if $O \equiv O_{\mathcal{B},b}(\mathsf{msk}_{i^*})$, then the oracle O' that D_{nc} simulates for D is precisely $O' \equiv O_{\mathcal{A},b}(\mathsf{msk})$.

– Suppose $O \equiv O_{\mathcal{B},0}(\mathsf{msk}_{i^*})$. The output of O is a pair (x, y) where $x \xleftarrow{\text{R}} \mathcal{X}$ and $y \leftarrow \mathsf{Eval}(\mathsf{msk}_{i^*}, x)$. By construction of D_{nc}, the output of O' is then a pair (x, y') where $x \xleftarrow{\text{R}} \mathcal{X}$ and

$$y' = \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_{i^*}, x) \oplus \left(\bigoplus_{i \neq i^*} \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_i, x) \right) = \mathsf{Eval}(\mathsf{msk}, x).$$

This is precisely the output distribution of $O_{\mathcal{A},0}(\mathsf{msk})$.

– Suppose $O \equiv O_{\mathcal{B},1}(\mathsf{msk}_{i^*})$. The output of O is a pair (x, y) where $x \xleftarrow{\text{R}} \mathcal{X}$ and $y \xleftarrow{\text{R}} \{0,1\}^\rho$. In this case, the distribution of $y' = y \oplus \big(\bigoplus_{i \neq i^*} \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_i, x)\big)$ is uniform over $\{0,1\}^\rho$ since y is sampled independently of msk_i and x for all i. As such, the output distribution of O' precisely coincides with the output distribution of $O_{\mathcal{A},1}(\mathsf{msk})$.

By the above analysis,

$$\Pr[D_{\mathsf{nc}}^{O_{\mathcal{B},b}(\mathsf{msk}_{i^*})}(1^\lambda) = b : b \xleftarrow{\text{R}} \{0,1\}] = \Pr[D^{O_{\mathcal{A},b}(\mathsf{msk})}(1^\lambda) = b : b \xleftarrow{\text{R}} \{0,1\}] \geq 1/2 + 1/(4z).$$

Thus, the event $\mathsf{GoodDis}_{\mathcal{B},1/(4z)}$ holds.

- Next, let $T \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace}^{D_{i^*}}(\mathsf{msk}_{i^*}, 1^{4z})$ where D_{i^*} is constructed from D as specified in Construction 3.9. In the reduction, algorithm $\mathcal{B}$ constructs D_{nc} from D in exactly the same way. By assumption, we have that $T \neq \{b^*\}$. This means that the output of $\mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Trace}^{D_{\mathsf{nc}}}(\mathsf{msk}_{i^*}, 1^{4z})$ is also not $\{b^*\}$. However, since $\mathcal{B}$ makes a single key-generation query to its challenger on identity b^*, this means that event $\neg\mathsf{CorrectTr}_{\mathcal{B},1/(4z)}$ occurs.

We conclude that $\Pr[\mathsf{GoodDis}_{\mathcal{B},1/(4z)} \wedge \neg\mathsf{CorrectTr}_{\mathcal{B},1/(4z)}] \geq \varepsilon'/(2\ell) - \mathsf{negl}(\lambda)$, which is non-negligible. □

Lemma 3.15. *Let D be the distinguisher the adversary outputs at the end of the experiment. We define* $\mathsf{ProbGoodDis}$ *to be the event where* $\mathsf{CheckDis}^D(\mathsf{msk}, 1^{1/\varepsilon}) = 1$*. If* FC *is fully collusion resistant in the presence of $Q+1$ tracing queries, then for all adversaries $\mathcal{A}$ in* Hyb_2*,* $\Pr[\mathsf{ProbGoodDis} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] \leq \mathsf{negl}(\lambda)$*.*

Proof. Suppose there is an adversary $\mathcal{A}$ in Hyb_2 where $\Pr[\mathsf{Hyb}_2[\mathsf{ProbGoodDis} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] = 1] = \varepsilon'$ for some non-negligible ε'. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the fingerprinting code security game:

1. Algorithm $\mathcal{B}$ samples $\mathsf{msk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Setup}(1^\lambda)$ for each $i \in [\ell]$. It also initializes an initially empty table T.

2. Algorithm $\mathcal{B}$ starts running $\mathcal{A}$. Whenever $\mathcal{A}$ makes an oracle query, algorithm $\mathcal{B}$ does the following:

   - **Evaluation queries:** On input $x \in \mathcal{X}$, $\mathcal{B}$ computes $y_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{Eval}(\mathsf{msk}_i, x)$ and replies to $\mathcal{A}$ with $y \leftarrow \bigoplus_{i \in [\ell]} y_i$.
   - **Key-generation queries:** On input $\mathsf{id} \in [n]$, $\mathcal{B}$ makes an encode query to its oracle to obtain a codeword $\bar{w}^{(\mathsf{id})} \in \{0,1\}^\ell$. Then, for each $i \in [\ell]$, algorithm $\mathcal{B}$ checks if there exists a mapping $(i, \bar{w}_i^{(\mathsf{id})}) \mapsto r_{i,\bar{w}_i^{(\mathsf{id})}}$ in T. If not, it samples a random $r_{i,\bar{w}_i^{(\mathsf{id})}} \xleftarrow{\text{R}} \mathcal{R}$ and adds $(i, \bar{w}_i^{(\mathsf{id})}) \mapsto r_{i,\bar{w}_i^{(\mathsf{id})}}$ to T. Algorithm $\mathcal{B}$ then computes $\mathsf{sk}_i \leftarrow \mathsf{TPRF}_{\mathsf{nc}}.\mathsf{KeyGen}(\mathsf{msk}_i, \bar{w}_i^{(\mathsf{id})}; r_{i,\bar{w}_i^{(\mathsf{id})}})$ for each i and replies to $\mathcal{A}$ with the tuple $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell, \bot, \bot, \bot)$

- **Tracing queries:** On input a distinguisher D and a tracing parameter 1^z, algorithm $\mathcal{B}$ first runs $\mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$ where $\mathsf{msk} = (\mathsf{msk}_1, \dots, \mathsf{msk}_\ell, \perp, \perp, \perp)$. If $\mathsf{CheckDis}$ outputs 0, output $\varnothing$. Otherwise, algorithm $\mathcal{B}$ computes the bits $\bar{w}_i^*$ using the same procedure as in Trace^D for each $i \in [\ell]$. It then submits a tracing query $\bar{w}^* \in \{0,1\}^\ell$ to its challenger. If the challenger replies with $\perp$, then $\mathcal{B}$ halts and outputs $\perp$. Otherwise, if the challenger replies with a set $S \subseteq [n]$, algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with S.

3. After $\mathcal{A}$ finishes making its queries, algorithm $\mathcal{A}$ outputs a distinguisher D. Algorithm $\mathcal{B}$ again computes $\bar{w}_i^*$ using the same procedure as in Trace^D for each $i \in [\ell]$. It submits a tracing query $\bar{w}^* \in \{0,1\}^\ell$ to its challenger.

We claim that $\mathcal{B}$ either *perfectly* simulates an execution of Hyb_2 for $\mathcal{A}$ or the experiment $\mathsf{ExptFC}_{\mathcal{B}}^{\mathsf{FC}}$ outputs 1. First, algorithm $\mathcal{B}$ perfectly simulates the evaluation and key-generation queries. We consider the tracing queries. Let D be the distinguisher and 1^z be the tracing parameter that $\mathcal{A}$ submits to the tracing oracle.

- If $\mathsf{CheckDis}^D(\mathsf{msk}, 1^{1/\varepsilon})$ outputs 0, then the output in Hyb_2 is $\varnothing$, which matches the behavior of $\mathcal{B}$.
- Alternatively, if $\mathsf{CheckDis}^D(\mathsf{msk}, 1^z)$ outputs 1, then the word $\bar{w}^* \in \{0,1\}^\ell$ is computed using the same procedure as in Hyb_2. If $\bar{w}^* \in F(W)$, where $W = \{\bar{w}^{(\mathsf{id}_j)}\}_{j \in [Q]}$, $\mathsf{id}_1, \dots, \mathsf{id}_Q \in [n]$ are the identities $\mathcal{A}$ submitted to the key-generation oracle prior to outputting D, and $\Gamma = \{\bar{w}^{(i)}\}_{i \in [n]}$ is the codebook sampled by the challenger for the fingerprinting code, then either the simulation is correct or experiment $\mathsf{ExptFC}_{\mathcal{B}}^{\mathsf{FC}}$ outputs 1. If $\bar{w}^* \notin F(W)$, then the output in Hyb_2 is $\perp$, which matches the behavior of $\mathcal{B}$.

Thus, either $\mathcal{B}$ perfectly simulates an execution of Hyb_2 for $\mathcal{A}$ or the experiment $\mathsf{ExptFC}_{\mathcal{B}}^{\mathsf{FC}}$ outputs 1.

It suffices to show that in the case where $\mathcal{B}$ perfectly simulates the execution of Hyb_2, the experiment $\mathsf{ExptFC}_{\mathcal{B}}^{\mathsf{FC}}$ also outputs 1 with probability at least ε'. By assumption, in Hyb_2, with probability ε', algorithm $\mathcal{A}$ will output a distinguisher D that satisfies $\mathsf{ProbGoodDis}$ and $\neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}$ (and the experiment does not abort). Since $\mathsf{ProbGoodDis}$ occurs (and the experiment does not abort), $\mathsf{Trace}^D(\mathsf{msk}, 1^{1/\varepsilon})$ computes a word $\bar{w}^* \in \{0,1\}^\ell$ where $\bar{w}^* \in F(W)$, where $W = \{\bar{w}^{(\mathsf{id}_j)}\}_{j \in [Q]}$ and $\mathsf{id}_1, \dots, \mathsf{id}_Q \in [n]$ are the identities algorithm $\mathcal{B}$ makes to the encode oracle (when responding to $\mathcal{A}$'s key-generation queries). Since the output of $\mathsf{Trace}^D(\mathsf{msk}, 1^{1/\varepsilon})$ in this case is $\mathsf{FC.Trace}(\mathsf{tk}_{\mathsf{FC}}, \bar{w}^*)$, and $\neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}$ occurs, this means that $\mathsf{FC.Trace}(\mathsf{tk}_{\mathsf{FC}}, \bar{w}^*)$ outputs a non-empty set S that contains an identity id' where $\mathsf{id}' \notin \{\mathsf{id}_1, \dots, \mathsf{id}_Q\}$. In this case, experiment $\mathsf{ExptFC}_{\mathcal{B}}^{\mathsf{FC}}$ outputs 1 and the claim holds. □

Lemma 3.16. *If* FC *is a fully collusion resistant in the presence of* $Q+1$ *tracing queries, then for all adversaries* $\mathcal{A}$ *in* Hyb_2*,* $\Pr[\mathsf{BadTr}_{\mathcal{A},\varepsilon}] \le \mathsf{negl}(\lambda)$.

Proof. Take an adversary $\mathcal{A}$ in Hyb_2, and let D be the distinguisher that $\mathcal{A}$ outputs at the end of Hyb_2. Let $S_{\mathsf{id}} \subseteq [n]$ be the set of identities that $\mathcal{A}$ submits to the key-generation oracle in Hyb_2. Consider the output of $\mathsf{Trace}^D(\mathsf{msk}, 1^{1/\varepsilon})$. We consider two possibilities:

- Suppose $\mathsf{CheckDis}^D(\mathsf{msk}, 1^{1/\varepsilon})$ outputs 0. Then the Trace algorithm outputs $\varnothing$ and $\mathsf{BadTr}_{\mathcal{A},\varepsilon}$ does not occur.
- Suppose $\mathsf{CheckDis}^D(\mathsf{msk}, 1^{1/\varepsilon})$ outputs 1. For $\mathsf{BadTr}_{\mathcal{A},\varepsilon}$ to occur in this case, the Trace algorithm must output a set T where $\varnothing \ne T \not\subseteq S_{\mathsf{id}}$. But this means $\neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}$ occurs in addition to the event $\mathsf{ProbGoodDis}$ (from Lemma 3.15). By Lemma 3.15, this event occurs with negligible probability. □

Combining Lemmas 3.13 to 3.16, we have that in $\mathsf{Hyb}_0 \equiv \mathsf{ExptTPRF}_{\mathcal{A},\varepsilon}^{\mathsf{TPRF}}$, for all efficient adversaries $\mathcal{A}$,

$$\Pr[\mathsf{CheckDis}^D(\mathsf{msk}, 1^{1/\varepsilon}) = 1 \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] \le \mathsf{negl}(\lambda) \text{ and } \Pr[\mathsf{BadTr}_{\mathcal{A},\varepsilon}] \le \mathsf{negl}(\lambda),$$

where D is the distinguisher $\mathcal{A}$ outputs at the end of the experiment. To complete the proof, we compute the probability of the event $\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}$. By Lemma 3.2, if $\mathsf{GoodDis}_{\mathcal{A},\varepsilon}$ holds, then $\mathsf{CheckDis}^D(\mathsf{msk}, 1^{1/\varepsilon}) = 1$ with probability $1 - \mathsf{negl}(\lambda)$. Correspondingly,

$$\Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon} \wedge \neg\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] \le \mathsf{negl}(\lambda),$$

and the claim follows. □

4 An Application: Traitor Tracing with Active Security

In this section, we introduce stronger security notions for traitor tracing in the secret-key setting, and then show that our new traceable PRFs directly yields constructions of these notions. We strengthen existing definitions along two main axis:

- **CCA-security:** We require that the underlying encryption scheme itself is secure against chosen-ciphertext attacks (i.e., "CCA-secure") [NY90, RS91, DDN00]. CCA-security (and in the symmetric setting, authenticated encryption), is essential for guaranteeing security against active adversaries. Previous definitions of traitor tracing only required that the underlying encryption scheme be secure against chosen plaintext attacks (i.e., "CPA-secure"), which is inadequate in the presence of active adversaries.
- **Secure tracing against active adversaries:** Like many recent works [GKW18, CVW$^+$18, Zha20], our construction only supports secret tracing. Analogous to the setting of traceable PRFs, we can consider a stronger tracing requirement where secure tracing holds even if the adversary has access to a tracing oracle. This models active adversaries that can make multiple attempts to try and evade the traitor tracing algorithm (and can observe the behavior of the tracing authority in response to each of those attempts).

We first recall the definition of traitor tracing, specialized to the secret-key setting. Our definitions are adapted from those of Goyal et al. [GKW18] and Zhandry [Zha20].

Definition 4.1 (Traitor Tracing [GKW18, Zha20, adapted])**.** Let λ be a security parameter. A secret-key traitor tracing scheme with message space $\mathcal{M}$ and identity space $[n]$ where $n = n(\lambda)$ is a tuple of five algorithms $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ with the following properties:[7]

- $\mathsf{Setup}(1^\lambda) \to \mathsf{msk}$: The setup algorithm takes as input the security parameter λ, and outputs a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$: The key-generation algorithm takes the master secret key msk and an identity $\mathsf{id} \in [n]$ and outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.
- $\mathsf{Enc}(\mathsf{msk}, m) \to \mathsf{ct}$. The encryption algorithm takes the master secret key msk and a message $m \in \mathcal{M}$ and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$. The decryption algorithm takes as input a secret key sk (which could be the master key msk) and a ciphertext ct and outputs a message $m \in \mathcal{M} \cup \{\perp\}$.
- $\mathsf{Trace}^D(\mathsf{msk}, 1^z, m_0, m_1) \to T$. The tracing algorithm has oracle access to a program D, and takes as input the master secret key msk, a parameter z, and two messages $m_0, m_1 \in \mathcal{M}$. It outputs a set $T \subseteq [n]$.

Moreover, the traitor tracing scheme should satisfy the following correctness property:

- **Correctness:** For all polynomials $n = n(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all identities $\mathsf{id} \in [n]$ and all messages $m \in \mathcal{M}$,

  $$\Pr[\mathsf{Dec}(\mathsf{msk}, \mathsf{Enc}(\mathsf{msk}, m)) \neq m : \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)] \leq \mathsf{negl}(\lambda)$$

  and

  $$\Pr[\mathsf{Dec}(\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}), \mathsf{Enc}(\mathsf{msk}, m)) \neq m : \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)] \leq \mathsf{negl}(\lambda).$$

[7] More generally, the message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ can also be parameterized by the security parameter λ. For simplicity of notation, we omit this parameterization here.

Security. There are two main security requirements on a traitor tracing scheme. The first is that the underlying encryption scheme is semantically secure and the second is tracing security. As noted above, most existing definitions of traitor tracing only consider these notions in a "passive setting" (i.e., CPA security and tracing security where the adversary does not have access to the tracing oracle). In this work, we consider *active* notions of both security notions. Namely, we require that the underlying encryption scheme satisfy CCA-security and that tracing security holds even if the adversary has access to the tracing oracle. We define these notions formally below:

Definition 4.2 (CCA Security)**.** A secret-key traitor tracing scheme $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ is CCA-secure if for every efficient and admissible algorithm $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in N$,

$$\Pr\left[\mathcal{A}^{O_b(\mathsf{msk},\cdot,\cdot),\mathsf{Dec}(\mathsf{msk},\cdot)}(1^\lambda) = b : \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda), b \xleftarrow{\text{R}} \{0,1\}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathcal{O}_b(\mathsf{msk}, m_0, m_1)$ outputs $\mathsf{Enc}(\mathsf{msk}, m_b)$. We say that $\mathcal{A}$ is admissible if for all queries ct that algorithm $\mathcal{A}$ submits to the decryption oracle $\mathsf{Dec}(\mathsf{msk}, \cdot)$, it is the case that ct was *not* previously output by the encryption oracle $\mathcal{O}_b$.

Definition 4.3 (Tracing Security against Active Adversaries)**.** Let $\mathsf{TT} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Trace})$ be a secret-key traitor tracing scheme with message space $\mathcal{M}$ and identity space $[n]$ where $n = n(\lambda)$. For a function $\varepsilon = \varepsilon(\lambda)$ and adversary $\mathcal{A}$, we define the tracing experiment $\mathsf{ExptTT}^{\mathsf{TT}}_{\mathcal{A},\varepsilon}(\lambda)$ as follows:

Experiment $\mathsf{ExptTT}^{\mathsf{TT}}_{\mathcal{A},\varepsilon}(\lambda)$

- $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$.
- $(D, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{Enc}(\mathsf{msk},\cdot),\mathsf{Dec}(\mathsf{msk},\cdot)}$
- $T \leftarrow \mathsf{Trace}^D(\mathsf{msk}, 1^{1/\varepsilon(\lambda)}, m_0, m_1)$.

Let S_{id} be the set of identities algorithm $\mathcal{A}$ submits to the key-generation oracle. Based on the output of $\mathsf{ExptTT}^{\mathsf{TT}}_{\mathcal{A},\varepsilon}$ above experiment, we define the following set of (probabilistic) events and the corresponding probabilities (which are functions of λ and parameterized by $\mathcal{A}, \varepsilon$):

- $\mathsf{GoodDis}_{\mathcal{A},\varepsilon}$: This is the event where $\Pr[D^{O_b}(1^\lambda) = b : b \xleftarrow{\text{R}} 0, 1] \geq 1/2 + \varepsilon(\lambda)$, where oracle O_b is the semantic security challenge oracle (with msk hard-wired) that outputs $\mathsf{Enc}(\mathsf{msk}, m_b)$. This property says that the distinguisher D output by $\mathcal{A}$ can successfully distinguish between encryptions of m_0 and m_1.[8]
- $\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}$: This is the event where $T \neq \emptyset \wedge T \subseteq S$. This event corresponds to the tracing algorithm successfully outputting one or more of the keys the adversary possesses.
- $\mathsf{BadTr}_{\mathcal{A},\varepsilon}$: $T \not\subseteq S$. This event corresponds to the tracing algorithm outputting a key that the adversary did not request.

A traitor tracing scheme $\mathcal{T}$ satisfies secure tracing if for every $\lambda \in \mathbb{N}$, every efficient adversary $\mathcal{A}$, and every inverse polynomial function $\varepsilon(\lambda) = 1/\mathsf{poly}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{BadTr}_{\mathcal{A},\varepsilon}] \leq \mathsf{negl}(\lambda) \text{ and } \Pr[\mathsf{CorrectTr}_{\mathcal{A},\varepsilon}] \geq \Pr[\mathsf{GoodDis}_{\mathcal{A},\varepsilon}] - \mathsf{negl}(\lambda). \tag{4.1}$$

[8] In the public-key setting considering in previous works, it is unnecessary to give D oracle access to the encryption algorithm, since the distinguisher can simulate encryption queries itself using the public key. In the secret-key setting, we provide the adversary oracle access to the encryption algorithm.

Similar to the case with traceable PRFs (Definition 2.14), we say that TT satisfies secure tracing against active adversaries if Eq. (4.1) holds even if the adversary $\mathcal{A}$ in $\mathsf{ExptTT}^{\mathsf{TT}}_{\mathcal{A},\varepsilon}$ has oracle access to a tracing oracle $\mathcal{O}(\mathsf{msk}, \cdot, \cdot, \cdot, \cdot)$ that takes as input the description of a distinguisher D, the tracing parameter 1^z, and two messages m_0 and m_1, and outputs $\mathsf{Trace}^D(\mathsf{msk}, 1^z, m_0, m_1)$. We say TT satisfies secure tracing against Q-bounded active adversaries if secure tracing holds against all efficient adversaries that makes at most Q queries to the tracing oracle $\mathcal{O}(\mathsf{msk}, \cdot, \cdot, \cdot, \cdot)$.

4.1 Traceable PRFs to Traitor Tracing

It is well known that PRFs can be used to construct authenticated encryption schemes (e.g., via the "encrypt-then-MAC" paradigm [BN00]). Not surprising, instantiating the encryption scheme with a traceable PRF and composing with an arbitrary MAC (*without* any tracing guarantees) directly yields a traitor tracing scheme where the underlying encryption scheme is an authenticated encryption (and hence, trivially satisfies CCA-security). Moreover, if the underlying traceable PRF is secure against (Q-bounded) active adversaries, then the resulting traitor tracing scheme is also secure against (Q-bounded) active adversaries. We state the construction below:

Construction 4.4 (Secret-Key Traitor Tracing with Active Security)**.** Let $\lambda \in \mathbb{N}$ be a security parameter. Let $\mathsf{TPRF} = (\mathsf{TPRF.Setup}, \mathsf{TPRF.KeyGen}, \mathsf{TPRF.Eval}, \mathsf{TPRF.Trace})$ be a traceable PRF with domain $\mathcal{X}$, range $\{0,1\}^\rho$, and identity space $[n]$. Let $\mathsf{PRF}\colon \mathcal{K} \times (\mathcal{X} \times \{0,1\}^\rho) \to \{0,1\}^\lambda$ be a secure PRF. We construct a secret-key traitor tracing scheme with message space $\{0,1\}^\rho$ as follows:

- $\mathsf{Setup}(1^\lambda)$: Run $\mathsf{TPRF.msk} \leftarrow \mathsf{TPRF.Setup}(1^\lambda)$ and $k \xleftarrow{\text{R}} \mathcal{K}$. Output $\mathsf{msk} = (\mathsf{TPRF.msk}, k)$.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$: On input $\mathsf{msk} = (\mathsf{TPRF.msk}, k)$, output $\mathsf{sk}_{\mathsf{id}} = (\mathsf{TPRF.sk}_{\mathsf{id}}, k)$ where $\mathsf{TPRF.sk}_{\mathsf{id}} \leftarrow \mathsf{TPRF.KeyGen}(\mathsf{TPRF.msk}, \mathsf{id})$.
- $\mathsf{Enc}(\mathsf{msk}, m)$: On input $\mathsf{msk} = (\mathsf{TPRF.msk}, k)$, sample $x \xleftarrow{\text{R}} \mathcal{X}$, and compute $y \leftarrow \mathsf{TPRF.Eval}(\mathsf{TPRF.msk}, x) \oplus m$, $\tau \leftarrow \mathsf{PRF}(k, (x, y))$. Output the ciphertext $\mathsf{ct} \leftarrow (x, y, \tau)$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: On input $\mathsf{sk} = (\mathsf{TPRF.sk}, k)$ and $\mathsf{ct} = (x, y, \tau)$, check if $\mathsf{PRF}(k, (x, y)) = \tau$. If the check fails, output $\perp$. Otherwise, output $\mathsf{TPRF.Eval}(\mathsf{TPRF.sk}, x) \oplus y$.
- $\mathsf{Trace}^D(\mathsf{msk}, 1^z, m_0, m_1)$: On input $\mathsf{msk} = (\mathsf{TPRF.msk}, k)$, the parameter z, messages $m_0, m_1 \in \{0,1\}^\rho$, and a distinguisher D, define the oracle-aided traceable PRF distinguisher $\hat{D}^{\hat{O}}$ that operates as follows:
  - Sample a bit $\beta \xleftarrow{\text{R}} \{0,1\}$.
  - Run the distinguisher D. Whenever D makes a query to its encryption oracle, algorithm $\hat{D}$ makes a query to its oracle $\hat{O}$ to obtain a value (x, y).
  - Compute $z \leftarrow y \oplus m_\beta$ and $\tau \leftarrow \mathsf{PRF}(k, (x, z))$. Algorithm $\hat{D}$ replies to D's query with the ciphertext $\mathsf{ct} = (x, z, \tau)$.
  - Eventually, algorithm D outputs a bit $\beta' \in \{0,1\}$. Algorithm $\hat{D}$ outputs 0 if $\beta = \beta'$ and 1 otherwise.

  Output $\mathsf{TPRF.Trace}^{\hat{D}}(\mathsf{TPRF.msk}, 1^{2z})$.

Correctness and security analysis. Correctness and security of Construction 4.4 follows directly from correctness and security of the underlying traceable PRF. We state the formal theorems here.

Theorem 4.5 (Correctness)**.** *If* TPRF *satisfies key similarity, then* TT *from Construction 4.4 is correct.*

Proof. Take any message $m \in \mathcal{M}$. Sample $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$, and let $\mathsf{ct} = (x, y, \tau) \leftarrow \mathsf{Enc}(\mathsf{msk}, m)$. If we write $\mathsf{msk} = (\mathsf{TPRF.msk}, k)$, then $y = \mathsf{TPRF.Eval}(\mathsf{msk}, x) \oplus m$. Clearly, decryption with the master secret key correctly recovers m. For any identity $\mathsf{id} \in [n]$, decryption with $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ succeeds as long as $\mathsf{TPRF.Eval}(\mathsf{TPRF.sk}_{\mathsf{id}}, x) = \mathsf{TPRF.Eval}(\mathsf{TPRF.msk}, x)$ where $\mathsf{TPRF.sk}_{\mathsf{id}} \leftarrow \mathsf{TPRF.KeyGen}(\mathsf{TPRF.msk}, \mathsf{id})$. Since x is uniform over $\mathcal{X}$, this follows by key similarity of TPRF. □

Theorem 4.6 (Authenticated Encryption)**.** *If* TPRF *satisfies weak pseudorandomness and* PRF *is secure, then* TT *from Construction 4.4 is an authenticated encryption scheme, and correspondingly, CCA-secure.*

Proof (Sketch). Since TPRF satisfies weak pseudorandomness, we have that $(x, m \oplus \mathsf{TPRF.Eval}(\mathsf{TPRF.msk}, x))$ is computationally indistinguishable from (x, y) where $x \xleftarrow{\text{R}} \mathcal{X}$ and $y \xleftarrow{\text{R}} \{0,1\}^\rho$. Finally, by PRF security, $\mathsf{PRF}(k, (x, y))$ is a secure MAC on (x, y). Thus, this encryption scheme is an instantiation of "encrypt-then-MAC," which we know provides authenticated encryption [BN00]. □

Theorem 4.7. *If* TPRF *satisfies secure tracing, then* TT *from Construction 4.4 also satisfies secure tracing. If* TPRF *is secure against (Q-bounded) active adversaries, then so is* TT*.*

Proof (Sketch). We show that if

$$\Pr[D^{O_b}(1^\lambda) = b : b \xleftarrow{\text{R}} \{0,1\}] \geq 1/2 + \varepsilon,$$

where O_b is the semantic security challenge oracle, then

$$\Pr[\hat{D}^{\hat{O}_b}(1^\lambda) = b : b \xleftarrow{\text{R}} \{0,1\}] \geq 1/2 + \varepsilon/2,$$

where $\hat{D}$ is the distinguisher constructed by Trace^D and $\hat{O}_b$ is the weak PRF challenge oracle. To see this, we consider two cases:

- Suppose $\hat{D}$ is given oracle access to $\hat{O}_0$. In this case, whenever $\hat{D}$ makes a query to O_0, it receives (x, y) where $y \leftarrow \mathsf{TPRF.Eval}(\mathsf{TPRF.msk}, x))$ for a uniformly random x. In this case, $\hat{D}$ responds to D's queries with $\mathsf{Enc}(\mathsf{msk}, m_\beta)$. By assumption, algorithm D will output β with probability $1/2 + \varepsilon$ in this case. Correspondingly, this means that D outputs 0 with probability $1/2 + \varepsilon$.
- Suppose $\hat{D}$ is given oracle access to $\hat{O}_1$. In this case, whenever $\hat{D}$ makes a query to $\hat{O}_1$, it receives (x, y) where $y \xleftarrow{\text{R}} \{0,1\}^\rho$. This means that $\hat{D}$ responds to D's queries with (essentially) one-time pad encryptions of m_β (the MAC is computed on a message blinded by a one-time pad). As such, the bit β' output by D is independent of β, and so $\beta = \beta'$ with probability exactly $1/2$. In this case, $\hat{D}$ outputs 1 with probability $1/2$.

Finally, since $b \xleftarrow{\text{R}} \{0,1\}$, we have that $\hat{D}^{\hat{O}_b}$ correctly predicts b with probability $1/2 + \varepsilon/2$. The claim now follows from tracing security of TPRF. In particular, the key-generation, encryption, and decryption queries can all be simulated using the key-generation and PRF evaluation queries of TPRF. Similarly, if TPRF is secure against (Q-bounded) active adversaries, the same holds for TT. Here, tracing queries to TT can be simulated by making the analogous query to TPRF. □

Remark 4.8 (Longer Message Space)**.** While Construction 4.4 only suffices to encrypt messages whose length ρ coincides with the output length of the PRF, as long as $\rho = \Omega(\lambda)$, it is easy to extend to arbitrary-length messages using standard key encapsulation techniques. Namely, we would use Construction 4.4 to encrypt a symmetric key k for an authenticated encryption scheme (that supports long messages), and then encrypt the message with the authenticated encryption scheme. As long as the key encapsulation mechanism supports tracing, the same extends to the composed scheme.

Acknowledgments

We thank the anonymous reviewers for helpful feedback on the presentation.

References

[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):6:1–6:48, 2012.

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

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT*, pages 531–545, 2000.

[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.

[BS95] Dan Boneh and James Shaw. Collusion-secure fingerprinting for digital data (extended abstract). In *CRYPTO*, pages 452–465, 1995.

[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.

[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.

[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.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

[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.

[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.

[GKW19] Rishab Goyal, Venkata Koppula, and Brent Waters. New approaches to traitor tracing with embedded identities. In *TCC*, 2019.

[GKWW21] Rishab Goyal, Sam Kim, Brent Waters, and David J. Wu. Beyond software watermarking: Traitor-tracing for pseudorandom functions. In *ASIACRYPT*, 2021.

[GQWW19] Rishab Goyal, Willy Quach, Brent Waters, and Daniel Wichs. Broadcast and trace with n^ϵ 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.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301), 1963.

[KT15] Aggelos Kiayias and Qiang Tang. Traitor deterring schemes: Using bitcoin as collateral for digital content. In *ACM CCS*, pages 231–242, 2015.

[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.

[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.

[Nis20] Ryo Nishimaki. Equipping public-key cryptographic primitives with watermarking (or: A hole is to watermark). In *TCC*, pages 179–209, 2020.

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

[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.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.

[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.

[RS91] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*, pages 433–444, 1991.

[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.

[Tar03] Gábor Tardos. Optimal probabilistic fingerprint codes. In *STOC*, pages 116–125, 2003.

[YAL$^+$18] Rupeng Yang, Man Ho Au, Junzuo Lai, Qiuliang Xu, and Zuoxia Yu. Unforgeable watermarking schemes with public extraction. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, 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.

[Zha20] Mark Zhandry. New techniques for traitor tracing: Size n$^{1/3}$ and more from pairings. In *CRYPTO*, pages 652–682, 2020.