Watermarking Public-Key Cryptographic Primitives

Rishab Goyal[1], Sam Kim[2], Nathan Manohar[3], Brent Waters[1,4], and David J. Wu[5]

[1] UT Austin, Austin, TX
[2] Stanford University, Stanford, CA
[3] UCLA, Los Angeles, CA
[4] NTT Research, East Palo Alto, CA
[5] University of Virginia, Charlottesville, VA

Abstract. A software watermarking scheme enables users to embed a message or mark within a program while preserving its functionality. Moreover, it is difficult for an adversary to remove a watermark from a marked program without corrupting its behavior. Existing constructions of software watermarking from standard assumptions have focused exclusively on watermarking pseudorandom functions (PRFs).

In this work, we study watermarking public-key primitives such as the signing key of a digital signature scheme or the decryption key of a public-key (predicate) encryption scheme. While watermarking public-key primitives might intuitively seem more challenging than watermarking PRFs, our constructions only rely on simple assumptions. Our watermarkable signature scheme can be built from the minimal assumption of one-way functions while our watermarkable public-key encryption scheme can be built from most standard algebraic assumptions that imply public-key encryption (e.g., factoring, discrete log, or lattice assumptions). Our schemes also satisfy a number of appealing properties: public marking, public mark-extraction, and collusion resistance. Our schemes are the first to simultaneously achieve all of these properties.

The key enabler of our new constructions is a relaxed notion of functionality-preserving. While traditionally, we require that a marked program (approximately) preserve the *input/output* behavior of the original program, in the public-key setting, preserving the "functionality" does not necessarily require preserving the *exact* input/output behavior. For instance, if we want to mark a signing algorithm, it suffices that the marked algorithm still output valid signatures (even if those signatures might be different from the ones output by the unmarked algorithm). Similarly, if we want to mark a decryption algorithm, it suffices that the marked algorithm correctly decrypt all valid ciphertexts (but may behave differently from the unmarked algorithm on invalid or malformed ciphertexts). Our relaxed notion of functionality-preserving captures the essence of watermarking and still supports the traditional applications, but provides additional flexibility to enable new and simple realizations of this powerful cryptographic notion.

1 Introduction

Watermarking is a way to embed special information called a "mark" into digital objects such as images, videos, audio, or software so that the marked object has the same appearance or behavior of the original object. Moreover, it should be difficult for an adversary to remove the mark without damaging the object itself. Watermarking is a useful tool both for protecting ownership and for preventing unauthorized distribution of digital media.

Software watermarking. In this work, we focus on software watermarking for cryptographic functionalities. Barak et al. [8, 9] and Hopper et al. [35] provided the first rigorous mathematical framework for software watermarking. Very briefly, a software watermarking scheme consists of two main algorithms. First, there is a *marking* algorithm that takes as input a program, modeled as a Boolean circuit C, and outputs a new marked circuit C' with the property that C and C' agree almost everywhere. Second, there is an *extraction* algorithm that takes as input a circuit C and outputs a bit indicating whether the program is marked or not. In the case of message-embedding watermarking, the marking algorithm additionally takes a message τ as input, and the extraction algorithm will either output the mark τ or a special symbol $\perp$ to indicate an unmarked program. The primary security requirement is unremovability, which says that given a marked circuit C' with an embedded message τ, no efficient adversary can construct a new circuit $\tilde{C}'$ that has roughly the same behavior as C', and yet the extraction algorithm on $\tilde{C}'$ fails to output τ. Notably, there are no restrictions on the circuit the adversary can output (other than the requirement that the adversary be efficient). This notion of security is often referred to as security against *arbitrary removal strategies* and captures the intuitive notion of watermarking where an adversary cannot replicate a program's functionality without also preserving the watermark.

Realizing the strong security requirements put forth in the early works on cryptographic watermarking [8, 35, 9] has proven challenging. In fact, Barak et al. showed an impossibility result (under indistinguishability obfuscation) on the existence of watermarking schemes that are *perfectly functionality-preserving* (i.e., schemes where the input/output behavior of the marked function is identical to that of the original function). In light of this lower bound, early works [44, 60, 48] provided partial results for watermarking specific classes of cryptographic functionalities by imposing limitations on the adversary's ability to modify the program and remove the watermark.

The first positive result on constructing watermarking schemes with security against *arbitrary* adversarial strategies was due to Cohen et al. [27] who showed that if we relax the perfect functionality-preserving requirement to only require *statistical* functionality-preserving (i.e., the marked function only has to implement the original function almost everywhere), then watermarking is possible. Moreover, Cohen et al. showed how to watermark several classes of cryptographic primitives, including pseudorandom functions (PRFs) and public-key encryption, with strong security from indistinguishability obfuscation. Since the seminal work

of Cohen et al., a number of works have studied how to build watermarkable families of PRFs from weaker assumptions such as lattice-based assumptions [37, 38] or CCA-secure encryption [52].

Watermarking public-key primitives. Existing constructions of software watermarking from standard cryptographic assumptions all focus on watermarking symmetric primitives, notably, PRFs [37, 52, 38]. The one exception is the work of Baldimtsi et al. [6], who showed how to watermark public-key cryptographic primitives, but in a stateful setting, and under a modified security model where a trusted watermarking authority generates *both* unmarked and marked keys.[6] Our focus in this work is constructing software watermarking schemes for two classes of public-key cryptographic primitives: digital signatures and (CPA-secure) public-key encryption (and more generally, public-key predicate encryption [19, 55, 36]).

1.1 Our Contributions

In this work, we show how to construct a watermarkable signature scheme, where the signing functionality can be marked, as well as a watermarkable public-key (predicate) encryption scheme, where the decryption functionality can be marked. Moreover, all of our constructions are based on very weak assumptions: namely, our watermarkable signature scheme can be constructed based on any vanilla signature scheme (say, from one-way functions [31]), and our watermarkable public-key predicate encryption scheme can be based on any public-key encryption scheme together with a low-complexity pseudorandom generator (implied by most standard intractability assumptions [45, 46, 47, 7]). One caveat is that our watermarkable predicate encryption scheme is only bounded-collusion secure.

Relaxing functionality-preserving. In spite of the recent progress in realizing new constructions of cryptographic watermarking from standard assumptions, watermarking remains a challenging notion to realize. Existing constructions of watermarking [37, 52, 38] from standard assumptions do not support properties like collusion resistance (where the watermark remains unremovable even if a user sees multiple marked versions of the program) or public verifiability (where anyone is able to tell if a program is marked). Moreover, these constructions rely on heavy cryptographic machinery, such as fully homomorphic encryption, even to watermark a PRF.

Our starting point, in this work, is to take a step back and revisit some of the definitions underlying software watermarking. Much like Cohen et al. [27] started by relaxing perfect functionality-preserving to statistical functionality-preserving and used that as the basis for obtaining the first positive results on watermarking, we also start by identifying another meaningful relaxation of the functionality-preserving requirement. As discussed above, functionality-preserving is typically synonymous with preserving a program's input/output behavior: namely the

[6] In the standard watermarking model, anyone can generate keys (without going through or trusting the watermarking authority), and at a later time, decide if they want to mark the keys or not.

input/output behavior of a marked circuit C' should be almost identical to that of the original circuit C. In many settings, such as when C implements a PRF, this is indeed the most natural notion of functionality-preserving. However, when considering the signing functionality of a signature scheme or the decryption functionality of an encryption scheme, there is additional flexibility:

- Suppose the circuit C implements the signing algorithm for a signature scheme. The functionality we care about is that on input a message m, $C(m)$ outputs a valid signature (with respect to the verification key vk). In this case, we can preserve this functionality *without* preserving the exact input/output behavior. Namely, we can allow the marked circuit C' to behave differently from C, as long as $C'(m)$ still outputs a valid signature (under vk) on the message m. In particular, the marked circuit is just as good as the original signing circuit even if they do not have identical input/output behavior. For instance, if we are watermarking the signing key used in a signature-based challenge-response authentication scheme, it suffices that the marked key still produces valid signatures, even if those signatures are not exactly the same as the ones output by an unmarked key.
- Suppose the circuit C implements the decryption algorithm for a public-key encryption scheme. In this case, the functionality we care about is that on input a valid ciphertext ct (i.e., one output by the encryption algorithm), $C(\mathsf{ct})$ outputs the underlying message m. In this case, the set of valid ciphertexts (i.e., those in the support of the honest encryption algorithm) might form a sparse subset of a larger space. In this case, we can define our functionality-preserving requirement to just require the marked circuit C' to correctly decrypt the set of valid ciphertexts. If we invoke C' on an invalid or malformed ciphertext, then C' and C are allowed to disagree. Analogous to the case with the signature scheme, the marked circuit C' is just as useful as a decryption circuit. Since the behavior of the decryption algorithm on a malformed ciphertext is usually unspecified, preserving this behavior seems non-essential for most applications. For example, if we are watermarking the decryption key (e.g., for a Blu-Ray player), it suffices that the marked key correctly decrypts *valid* ciphertexts.

To summarize, in the public-key setting, we can capture the spirit of "functionality-preserving" watermarking without requiring that the marked circuit and the unmarked circuit have identical input/output behaviors. It turns out that this added degree of freedom enables new constructions of software watermarking from simple assumptions (e.g., one-way functions or public-key encryption) that also satisfy a number of desirable properties that have eluded all existing watermarking constructions: collusion resistance, public marking, and public extraction. We discuss these properties in greater detail below and then comment more broadly on their implications.

Our results. By working with our relaxed notion of functionality-preserving, we construct watermarkable signatures and watermarkable public-key predicate encryption schemes that *simultaneously* achieve *all* of the following properties:

– **Collusion resistance:** Existing constructions of watermarking [27, 14, 37, 52, 38] only provide unremovability against adversaries that see a *single* marked key. While this is the natural notion in the setting where programs are either marked or unmarked, this is not true in the message-embedding case. In fact, in all the aforementioned constructions, an adversary that obtains two copies of a key marked with different messages $\tau \neq \tau'$ can efficiently construct a new program that is functionally-similar to the marked program, and yet does not contain the watermark. Such watermarking schemes are not *collusion resistant*. We say that a watermarking scheme is collusion resistant if an adversary who sees marked versions of a circuit C with marks $\tau_1, \ldots, \tau_n$ cannot construct a new circuit C' that is functionally-close to C and yet, on input C', the extraction algorithm fails to produce one of $\tau_1, \ldots, \tau_n$.[7] In applications where keys are watermarked with different identities (for instance, when the decryption key embedded in a Blu-Ray player is marked with the owner's name), collusion resistant unremovability is a critical property. In this work, we construct a watermarkable signature scheme that is fully collusion resistant (i.e., collusion resistant against an adversary who can see an arbitrary polynomial of marked circuits) and a watermarkable public-key predicate encryption scheme that provides bounded-collusion resistance.
– **Public marking:** A watermarking scheme supports public marking if anyone is able to run the marking algorithm. Conversely, a scheme supports secret marking if only the holder of a secret watermarking key is able to watermark programs. Public marking is a desirable feature because users are able to watermark their secret keys without having to share them with a watermarking authority. Several previous watermarking schemes for PRFs [52, 38] provided public marking, but at the expense of giving the watermarking authority a trapdoor that allows it to break security of all of the keys in the system (including *unmarked* keys). Our schemes naturally support public marking without this drawback (and, in fact, our schemes do not require the existence of a central watermarking authority at all).
– **Public extraction:** A watermarking scheme supports public mark-extraction if anyone can run the extraction algorithm and obtain the watermark within a piece of software. This is useful if users want to directly prove software ownership (or authenticity) without going through a trusted watermarking authority. Obtaining watermarkable PRFs with public extraction from standard assumptions remains a major open problem, and existing watermarking schemes with this property [27] all rely on indistinguishability obfuscation. In this work, all of our schemes support public mark-extraction.
– **Security against a malicious watermarking authority:** A watermarking scheme that supports public marking and public mark-extraction is very appealing because users do *not* need to trust a central watermarking authority for marking or extraction. Our schemes give the first watermarking scheme

[7] This is conceptually very similar to the closely-related cryptographic primitive of traitor tracing, and we discuss the similarities and differences in greater detail later in this section and in Section 1.2.

that supports public marking and public extraction. This resolves a key open question in the work of Cohen et al. [27], although under a relaxed (but still meaningful) notion of functionality-preserving. In fact, our schemes remain secure even if the public parameters of the watermarking scheme are chosen maliciously.

Our relaxed notion of functionality-preserving is certainly much weaker than the more stringent requirement of preserving input/output behavior. But, as we discussed above, our relaxed notion still seems to capture the essence of the requirement in the context of watermarking signatures and encryption schemes. By relaxing this functionality-preserving requirement, we are able to achieve stronger security notions from weaker cryptographic assumptions. At a philosophical level, our work highlights the need to further explore and identify the "right" set of definitions for software watermarking that enable useful and meaningful constructions from simple assumptions while still supporting the standard applications of software watermarking.

Watermarking digital signatures. Our watermarkable digital signature scheme relies on constrained signatures (also known as policy-based signatures) [10, 59]. In a constrained signature scheme over a message space $\mathcal{M}$, the signing key sk can be used to derive a constrained signing key sk_f for a particular predicate $f\colon \mathcal{M} \to \{0,1\}$ with the property that the constrained key sk_f can be used to sign all messages m where $f(m) = 1$. The security property is that an adversary who is given constrained keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ for functions $f_1, \ldots, f_n$ cannot produce a valid signature on any message m where $f_i(m) = 0$ for all $i \in [n]$. It is straightforward to construct constrained signatures from any standard signature scheme using certificates [10], and we briefly recall this basic construction in Section 3.3.

A constrained signature scheme that supports the class of "prefix-based" constraints immediately gives rise to a watermarkable signature scheme. In more detail, if we want to construct a watermarkable signature with message space $\mathcal{M}$ and mark space $\mathcal{T}$, we use a prefix-constrained signature scheme with message space $\mathcal{T} \times \mathcal{M}$. Signing and verification keys for the watermarkable signature directly correspond to signing and verification keys for the underlying prefix-constrained signature scheme. A signature on a message m consists of a tuple $\sigma_m = (\bot, \sigma')$ where σ' is a signature on $(\bot, m)$. To verify a signature $\sigma = (\tau, \sigma')$ on a message m, the verification algorithm checks that σ' is a valid signature on the pair (τ, m). Now, to mark a signing key with mark $\tau^* \in \mathcal{T}$, the user constrains the signing key sk to the prefix-based constraint $f_{\tau^*}\colon \mathcal{T} \times \mathcal{M} \to \{0,1\}$ where $f_{\tau^*}(\tau, x) = 1$ if $\tau^* = \tau$ and 0 otherwise. The marked circuit C_{τ^*} is a circuit that takes as input a message m and outputs (τ^*, σ'), where σ' is a signature on (τ^*, m) using the constrained key sk_{τ^*}. To extract a watermark from a candidate circuit C', simply sample a random message $m \xleftarrow{R} \mathcal{M}$,[8] compute $(\tau, \sigma') \leftarrow C'(m)$, and output τ if σ' is a valid signature on (τ, m). Note that if C' only succeeds

[8] More generally, we can consider a stronger notion of unremovability where we replace the uniform distribution over $\mathcal{M}$ with *any* (adversarially-chosen) efficiently-sampleable distribution over $\mathcal{M}$ where the circuit succeeds in generating valid signatures with

in producing valid signatures with ε probability (for non-negligible ε), then this procedure can be repeated λ/ε times. If no marks are extracted after λ/ε iterations, then output $\perp$ (to indicate an unmarked circuit).

By correctness of the underlying constrained signature scheme, the marked circuit C_{τ^*} outputs valid signatures on all messages $m \in \mathcal{M}$, so the marked circuit is functionality-preserving (even though the signatures output by C are noticeably different than the signatures output by the original signing algorithm). Unremovability follows from security of the underlying constrained signature. Namely, an adversary who only has signing circuits marked with $\tau_1, \ldots, \tau_n$ should only be able to compute signatures on tuples of the form (τ_i, m) for $i \in [n]$. Thus, if the extraction algorithm outputs some $\tau' \neq \tau_i$ for all $i \in [n]$, then the adversary's circuit must have forged a valid signature on (τ', m) for some message $m \in \mathcal{M}$, which breaks security of the underlying constrained signature scheme. In addition, if the underlying constrained signature scheme is collusion resistant (i.e., security holds against adversaries that obtain an a priori unbounded polynomial number of constrained keys), then the resulting watermarkable signature scheme is also collusion resistant. We describe this construction and its security analysis in greater detail in Section 3.

Watermarking public-key encryption and traitor tracing. Turning now to (CPA-secure) public-key encryption, we first describe a correspondence between watermarkable public-key encryption and traitor tracing [24]. In traitor tracing, there is a set of n honest users, each associated with a numeric identity $i \in [n]$. In addition, there is a central authority that generates a public key pk for the scheme as well as secret decryption keys sk_i for each user $i \in [n]$. Anyone can encrypt a message under the public key pk, and each legitimate user is able to decrypt the resulting ciphertext using their individual secret key sk_i. The tracing property says that there is an efficient tracing algorithm that, given black-box access to any valid decryption circuit, is able to recover at least one of the secret decryption keys sk_i that went into constructing the private decoder. As noted by Nishimaki et al. [49], a collusion resistant watermarkable public-key encryption scheme can be used to build a traitor tracing scheme: namely, the secret decryption keys for each user would correspond to watermarked decryption keys, where the watermark is the user's index.

With a few syntactic changes, the converse also holds; namely, any traitor tracing scheme that supports *public* tracing also implies a watermarkable public-key encryption scheme under our relaxed notion of functionality-preserving. Typically, in a traitor tracing scheme, there is a central authority that generates the public key and *all* of the decryption keys at the same time. In watermarking, however, anyone should be able to sample a public/private key-pair and, later on, have the ability to watermark their decryption key. However, this distinction is superficial, as we can always take the master secret key of the traitor tracing scheme to be the setup randomness and let that be the secret key in the watermarkable public-key encryption scheme. To mark the secret key with an identity $i \in [n]$, the

non-negligible probability. Notably, the support of this distribution may have negligible density in $\mathcal{M}$. We provide more details in the full version of this paper.

marking algorithm would run the setup algorithm of the traitor tracing scheme and output decryption key sk_i. Unremovability of the scheme follows directly from the traceability of the underlying traitor tracing scheme.[9]

Watermarking advanced public-key functionalities. Having established a correspondence between traitor tracing schemes with public tracing and watermarkable public-key encryption schemes, we ask whether we can watermark more complex public-key functionalities like identity-based encryption [54, 13, 26], attribute-based encryption [53, 34], or predicate encryption [19, 55, 36]. In the following description, we focus on predicate encryption, the most general notion among these primitives. In a (key-policy) predicate encryption scheme, ciphertexts are associated with an attribute x as well as a message m, while secret keys are associated with functions or predicates f. A secret key sk_f for a predicate f can decrypt all ciphertexts encrypted with respect to an attribute x where $f(x) = 1$. The security property is that an adversary who has keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ for predicates $f_1, \ldots, f_n$ cannot learn anything about ciphertexts encrypted to an attribute x where $f_i(x) = 0$ for all $i \in [n]$. Moreover, in a predicate encryption scheme, the ciphertexts *hide* the attribute x (whereas in the similar setting of attribute-based encryption, the attribute is public).

The question we ask is whether we can watermark the decryption keys sk_f in a predicate encryption scheme. As an example application, imagine an organization that uses a predicate encryption scheme for enforcing access control (e.g., ciphertexts are tagged with different classification levels), and it wants to issue decryption keys to different clients, each marked with the client's identity. Then, if a client uses their key to construct an unauthorized decryption device, it is possible to identify the identity of the client (by extracting the watermark).

Constructing a watermarkable predicate encryption scheme. In this work, we show that a generalization of the traitor tracing scheme by Nishimaki et al. [49] in combination with a hierarchical functional encryption scheme [22] gives a watermarkable (bounded-collusion) predicate encryption scheme based only on public-key encryption and the existence of pseudorandom generators (PRGs) computable in NC^1 (which follow from standard intractability assumptions such as factoring, discrete log, or lattice-based assumptions [45, 46, 47, 7]). This notion is conceptually similar to the notion of attribute-based traitor tracing, and we compare and contrast the two notions in Section 1.1. In contrast to the setting of watermarking public-key encryption, watermarking predicate encryption does not appear to follow from attribute-based traitor tracing (although the converse does follow).

[9] Traditionally, in a traitor tracing scheme, the tracing algorithm requires a secret tracing key output by the tracing algorithm [16, 30, 33, 23]. A traitor tracing scheme supports public tracing [18, 2, 49] if the tracing algorithm does not depend on any secret information. In this simple construction of watermarkable public-key encryption from traitor tracing, the extraction algorithm would not have access to the tracing key, so instantiating this basic blueprint will require a traitor tracing scheme that supports public tracing.

The starting point of our construction is the classic approach for constructing traitor tracing via a *private linear broadcast encryption* (PLBE) introduced by Boneh et al. [16]. In a PLBE scheme with n users, each associated with an index $i \in [n]$, it is possible to construct a ciphertext that can only be decrypted by users whose index $i < T$ is smaller than some threshold T. Moreover, ciphertexts encrypted to two different thresholds $T < T'$ are only distinguishable if a user possesses a secret key for an index $i \in \{T, T+1, \ldots, T'-1\}$. A PLBE scheme that supports n users implies a traitor tracing scheme with identity space $[n]$: namely, to trace a circuit C, the tracing algorithm encrypts (random) messages to indices $i = 0, 1, \ldots, n$, and tests whether C correctly decrypts the ciphertext or not. When $i = 0$, decryption always fails, while at $i = n$, decryption should succeed with noticeable probability. Thus, there must be some index i where there is a "big jump" in the decryption success probability, which corresponds to the user possessing the decryption key for index i. When n is polynomial, the tracing algorithm can simply do a linear scan over the entire identity space to identify the big jumps. Nishimaki et al. [49] show how to generalize this approach to the setting where the identity space is exponential (which allows embedding arbitrary information in the decryption keys).

As described above, a traitor tracing scheme that supports public tracing directly implies a watermarkable public-key encryption scheme. To extend this to the setting of watermarking a predicate encryption scheme, we use a hierarchical functional encryption scheme. In a standard functional encryption (FE) scheme [17, 50], encryption keys are associated with functions f, and decrypting a ciphertext encrypting a value x with a function key sk_f for f yields the value $f(x)$. It is not difficult to see that a FE scheme can be used to build a predicate encryption scheme as well as a PLBE scheme (this is the approach taken by Nishimaki et al. [49] in their traitor-tracing construction). In a hierarchical FE scheme [22], there is an additional delegation function that allows one to take a function key sk_f for a function f and delegate it to a key $\mathsf{sk}_{g \circ f}$ for the function $g \circ f$. At a high-level, our construction of a watermarkable predicate encryption scheme relies on a two-level hierarchical FE scheme, where the ordinary function keys are used to implement a predicate encryption scheme, while the marked keys consist of a delegated key that embeds a PLBE functionality (used to embed the watermark).

In more detail, to encrypt a message m with attribute x, we construct an FE encryption of the triple $(x, m, 1)$, where the last component is a special flag (used for mark extraction). A predicate encryption key for the predicate f consists of an FE key for the associated function g_f where

$$g_f(x, m, b) = \begin{cases} (x, m) & b = 0 \\ (0^\ell, 0^n) & b = 1 \text{ and } f(x) = 0 \\ (1^\ell, m) & b = 1 \text{ and } f(x) = 1. \end{cases}$$

By construction, decrypting an honestly-generated ciphertext with attribute x and message m with a key f where $f(x) = 1$ will always yield the pair $(1^\ell, m)$, from which the message can be recovered. Now, to mark a key sk_f with a mark

$\tau \in \{0,1\}^\ell \setminus \{1^\ell\}$, we take sk_f and use the delegation mechanism to issue a function key for the function $h_\tau \circ g_f$ where

$$h_\tau(x, m) = \begin{cases} (0^\ell, 0^n) & x \leq \tau \\ (1^\ell, m) & x > \tau, \end{cases}$$

where we interpret $x, \tau \in \{0,1\}^\ell$ as the binary representation of an ℓ-bit integer. We make two observations. First, the marked key can still be used to decrypt all honestly-generated ciphertexts (namely, ciphertexts where the flag is set to 1). However, notice that when the flag $b = 0$, the marked key can only decrypt ciphertexts where the encrypted attribute x exceeds the threshold τ associated with the marked key. This precisely coincides with the semantics of a PLBE scheme. We can now apply the techniques developed by Nishimaki et al. [49] to extract the associated identity τ, thereby recovering the watermark. We provide the full description of this scheme and its analysis in Section 4. Overall, we show that we can obtain a bounded-collusion resistant watermarkable family of predicate encryption schemes from public-key encryption and low-complexity PRGs; both of these assumptions can be instantiated by most assumptions that imply public-key encryption (e.g., factoring, discrete log, or lattice-based assumptions [45, 46, 47, 7]).

The parameter sizes of the resulting bounded-collusion watermarkable predicate encryption scheme are directly inherited from those of the underlying bounded-collusion hierarchical functional encryption scheme, which can in turn be built from a standard bounded-collusion functional encryption scheme [22]. For instance, instantiating the underlying functional encryption scheme with [32] yields a watermarkable predicate encryption scheme where the ciphertext size scales with $O(Q^4)$, where Q is the collusion bound. Alternatively, with the FE scheme from [3], the ciphertext size scales with $O(Q^2)$ and with the scheme from [5], the ciphertext size scales with $O(Q)$.

1.2 Additional Related Work

In this section, we survey some additional related work as well as compare our new watermarking notions to related notions studied in prior work.

Constrained signatures. Numerous works [43, 21, 10, 59] have studied constructing constrained signatures (and variants thereof) together with properties like privacy, anonymity, succinct keys, or succinct signatures.

Traitor tracing. Since the work of Chor et al. [24], there have been a vast number of constructions of fully collusion resistant traitor tracing from combinatorial constructions [15, 56], pairing-based assumptions [16, 18, 30, 29, 41, 42], lattice-based assumptions [33, 23], and indistinguishability obfuscation [20, 49]. With the exception of [49], the existing constructions only support efficient tracing over a polynomial-size identity space (this is referred to as "flexible" traitor tracing [49]). There are also numerous constructions that provide security in the bounded-collusion setting [24, 58, 39, 12, 25, 57, 51, 1, 28, 11, 40, 2].

Attribute-based traitor tracing. Directly relevant to our notion of watermarkable predicate encryption is the notion of attribute-based traitor tracing [1, 41, 42, 23], which is a hybrid of attribute-based encryption and traitor tracing. The main difference between these two notions is that in the traitor-tracing setting, the marking and key-generation algorithms are combined (namely, the key-generation algorithm takes as input the function together with the mark). In watermarking, we have the additional flexibility that we can embed the watermark *after* issuing the key as well as support watermarking *adversarially-chosen* keys. When considering the simpler notion of watermarkable public-key encryption and traitor tracing, we can equate these two notions with a suitable redefinition of the traitor tracing schema (assuming that the traitor tracing scheme supports a *public* tracing algorithm). However, this equivalence does not seem to extend to the setting of attribute-based encryption or predicate encryption. Another key difference is that existing constructions of attribute-based traitor tracing from standard assumptions only support tracing over a polynomial-size identity space, while in the standard notions of message-embedding watermarking, the identity space is exponential. Note that since a watermarkable predicate encryption scheme implies an attribute-based traitor tracing scheme, our results give a bounded-collusion attribute-based traitor tracing scheme that supports an exponential number of possible identities.

2 Preliminaries

We begin by introducing the notation that we use in this work. We use λ (often implicitly) to denote the security parameter. We write $\mathsf{poly}(\lambda)$ to denote a quantity that is bounded by a fixed polynomial in λ and $\mathsf{negl}(\lambda)$ to denote a function that is $o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We say that an event occurs with overwhelming probability if its complement occurs with negligible probability. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. For two families of distributions $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_2 = \{\mathcal{D}_{2,\lambda}\}_{\lambda\in\mathbb{N}}$, we write $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$ if the two distributions are computationally indistinguishable (i.e., no efficient algorithm can distinguish distribution $\mathcal{D}_1$ from $\mathcal{D}_2$ except with negligible probability), and $\mathcal{D}_1 \overset{s}{\approx} \mathcal{D}_2$ if the two distributions are statistically indistinguishable (i.e., the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is $\mathsf{negl}(\lambda)$).

For an integer $n \geq 1$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. For integers $n \geq m \geq 1$, we write $[m, n]$ to denote the set of integers $\{m, m+1, \ldots, n\}$, and $[m, n]_{\mathbb{R}}$ to denote the closed interval between m and n (inclusive) over the real numbers. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is drawn from $\mathcal{D}$. For a finite set S, we write $x \xleftarrow{R} S$ to denote that x is drawn uniformly at random from S. For 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 $\mathcal{Y}$. In the the full version of this paper, we also recall the definition of a digital signature scheme and a public-key predicate encryption scheme.

3 Watermarking Digital Signatures

In this section, we show how to watermark a digital signature scheme. We begin by formally introducing the notion of a watermarkable signature scheme. Our definitions are based on adaptations of existing definitions of watermarking PRFs [27, 14, 37, 52, 38] as well as the candidate definitions for watermarking public-key functionalities put forward in the work of Cohen et al. [27]. We present our construction in the fully public-key setting (namely, both marking and extraction are public operations, and there is no watermarking secret key).

Definition 3.1 (Watermarkable Signature). *A watermarkable digital signature scheme with message space $\mathcal{M}$ and mark space $\mathcal{T}$ is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Mark}, \mathsf{Extract})$ *with the following properties:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{wpp}$*: On input the security parameter λ, the setup algorithm outputs a set of watermarking public parameters* wpp.
- *The public parameters* wpp *induce a digital signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *with message space $\mathcal{M}$, verification key space $\mathcal{VK}$, signing key space $\mathcal{SK}$, and signature space $\mathcal{SIG}$. Note that we implicitly allow* KeyGen, Sign, *and* Verify *to take* wpp *as input.*
- $\mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}, \tau) \to C$*: On input the watermarking parameters* wpp*, a signing key* $\mathsf{sk} \in \mathcal{SK}$*, and a mark $\tau \in \mathcal{T}$, the marking algorithm outputs a circuit $C \colon \mathcal{M} \to \mathcal{SIG}$.*
- $\mathsf{Extract}(\mathsf{wpp}, \mathsf{vk}, C) \to \tau/\bot$*: On input the watermarking parameters* wpp*, a verification key* $\mathsf{vk} \in \mathcal{VK}$*, and a circuit $C \colon \mathcal{M} \to \mathcal{SIG}$, the extraction algorithm either outputs a mark $\tau \in \mathcal{T}$ or a special symbol $\bot$.*

Remark 3.2 (Comparison with Cohen et al. [27]). There are several differences between our watermarking schema and that introduced by Cohen et al. [27]. We summarize these below:

- **Extraction semantics:** Our extraction algorithm $\mathsf{Extract}$ additionally takes the verification key associated with the signing circuit as an additional input. This does not seem like a substantial limitation to the usefulness of the scheme since in most applications, the verification key associated with a signature scheme is assumed to be publicly known.
- **Independent key-generation and marking algorithms:** In addition, we have *independent* key-generation and marking algorithms. The schema from Cohen et al. for public-key primitives introduced an additional restriction that the watermark is generated at the same time as the signing key, while in our scheme, the signing key can be generated independently, and later on, a user can decide to mark the key. Thus, our schema provides additional flexibility in how keys are generated and marked. In particular, our definition allows a user to take the same signing key and mark it with different messages (for instance, to give to different users). This definition is more similar to existing definitions for watermarking secret-key primitives, which consider independent key-generation and marking algorithms.

We additionally note that if we allow the verification key to depend on the mark (i.e., as in the Cohen et al. construction), there is a simple way to satisfy their definition.[10] In particular, we simply include the mark τ as part of the signing key and verification key. A signature on a message m is just the pair (τ, σ), where σ is a vanilla signature on m. Verification first affirms that the first component of the signature is the mark τ and then checks σ as usual. If the adversary constructs a circuit that outputs valid signatures with probability better than $\varepsilon > 1/2 + 1/\mathsf{poly}(\lambda)$,[11] then the output of the circuit contains the mark τ on a majority of inputs. In this case, the extraction algorithm can evaluate the circuit on $\mathsf{poly}(\lambda)$ random inputs and output the majority tag. This basic construction does not apply in our setting because we require that the signing/verification keys be generated *independently* of the mark.

– **Collusion resistance:** Since our definition separates the key-generation and marking algorithms, the *same* signing key can be marked with different messages. Correspondingly, we can define a notion of *collusion resistant* watermarking, where unremovability holds even if the adversary sees the same signing key marked with *distinct* messages. This is a critical property for any realistic application of watermarking where a single key might be marked with multiple identities.

Correctness. Next, we introduce the correctness requirements of a watermarking scheme. There are three main properties we care about. The first is the usual notion of extraction correctness, which says that the extraction algorithm should successfully recover the watermark from an honestly-marked key. The second property is a "meaningfulness" or "non-triviality" property, which says that most circuits should not be marked. Finally, the third property is functionality-preserving. As noted in Section 1.1, one of the main differences between this work and previous works on software watermarking is we consider a relaxed notion of functionality-preserving, where we require that a marked signing key can sign arbitrary messages (that verify with respect to the *same* verification key), but we allow the resulting signatures to be different from the signatures output by the original signing key. In other words, the marked key implements a perfectly valid signing algorithm, but it does not have to preserve the exact input/output behavior of the unmarked signing key. We give the precise definition below:

Definition 3.3 (Correctness). *Let* $\Pi_{\mathsf{WM}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Mark}, \mathsf{Extract})$ *be a watermarkable signature scheme with message space* $\mathcal{M}$*, signature space* $\mathcal{SIG}$*, and verification key space* $\mathcal{VK}$*. Then,* Π_{WM} *is correct if for all*

[10] This basic construction also directly extends to their notion of watermarkable public-key encryption, which considers the analogous restriction where the encryption/decryption keys are sampled jointly with the watermark.

[11] As noted in Remark 3.9, the unremovability definition in Cohen et al. [27] (for message-embedding watermarking) is satisfiable only when the adversary is restricted to constructing circuits that agree with the marked circuit on strictly more than half of the inputs. This coincides with the setting where our simple construction applies.

$\mathsf{wpp} \leftarrow \mathsf{Setup}(1^\lambda)$, *the induced signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is correct, and the following properties also hold:*

- ***Extraction correctness:*** *For all marks* $\tau \in \mathcal{T}$*,*

  $$\Pr[(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{wpp}) : \mathsf{Extract}(\mathsf{wpp}, \mathsf{vk}, \mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}, \tau)) \neq \tau] = \mathsf{negl}(\lambda).$$

- ***Meaningfulness:*** *For all fixed circuits* $C \colon \mathcal{M} \to \mathcal{SIG}$ *(independent of the public parameters* wpp*) and all verification keys* $\mathsf{vk} \in \mathcal{VK}$*,*

  $$\Pr[\mathsf{Extract}(\mathsf{wpp}, \mathsf{vk}, C) \neq \bot] = \mathsf{negl}(\lambda),$$

  and for $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{wpp})$*,*

  $$\Pr[\mathsf{Extract}(\mathsf{wpp}, \mathsf{vk}, \mathsf{Sign}(\mathsf{wpp}, \mathsf{sk}, \cdot)) \neq \bot] = \mathsf{negl}(\lambda).$$

- ***Functionality-preserving:*** *For all marks* $\tau \in \mathcal{T}$ *and all messages* $m \in \mathcal{M}$*, if we take* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{wpp})$ *and* $C \leftarrow \mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}, \tau)$*,*

  $$\Pr[\mathsf{Verify}(\mathsf{wpp}, \mathsf{vk}, m, C(m)) \neq 1] = \mathsf{negl}(\lambda).$$

Remark 3.4 (Unique Signature Schemes and Functionality-Preserving). We note that if we have a *unique* signature scheme (i.e., a signature scheme where for every message $m \in \mathcal{M}$, there is a unique signature σ that verifies with respect to the verification key), then our notion of functionality-preserving precisely coincides with preserving the input/output behavior of the original signing circuit. We do not know how to watermark a unique signature scheme and leave this as an intriguing open problem.

Security of the underlying signature scheme. The first security requirement is that the underlying signature scheme associated with a watermarkable signature scheme satisfies the usual notion of unforgeability. In fact, we would like the stronger property that even if the watermarking parameters wpp are chosen in a malicious manner, the resulting signature scheme remains secure (i.e., provides unforgeability). Recent constructions of watermarking for PRFs [52, 38] have the drawback that even a semi-honest watermarking authority is able to break security of the *unmarked* keys in the system (and previous constructions from standard assumptions [37] become insecure if the watermarking authority generates the parameters maliciously). Our security notion ensures that a malicious party cannot generate the parameters in such a way as to embed a "trapdoor" into the signature scheme. In fact, since our watermarking scheme supports both public marking and public verification, this property means that users can use the scheme without needing to trust any central authority; this is an appealing property that is not satisfied by *any* existing watermarking scheme.

Definition 3.5 (Signature Unforgeability). *Let* $\Pi_{\mathsf{WM}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Mark}, \mathsf{Extract})$ *be a watermarkable signature scheme. We say that* Π_{WM} *satisfies signature unforgeability if the induced signature scheme* $(\mathsf{KeyGen},$

$\mathsf{Sign}, \mathsf{Verify}$) *satisfies unforgeability. We say that* Π_{WM} *satisfies* signature unforgeability in the presence of a malicious authority *if the induced signature scheme satisfies unforgeability even if the adversary can choose the public parameters* wpp *for* Π_{WM}.

Unremovability. The main security requirement we require from a watermarking scheme is unremovability: namely, an adversary that obtains one or more marked keys cannot produce a new key that preserves the same functionality as the original key and, yet, does not contain the watermark. Our definition is the direct generalization of the corresponding notion of unremovability in the setting of watermarking PRFs [27, 14], with the following differences:

- First, we use the same relaxation of functionality-preserving discussed above: namely, the adversary is allowed to construct any circuit that outputs valid signatures with noticeable probability (that verify under the signature scheme's verification key); it does not have to preserve the input/output behavior of the marked circuits it is given. This gives the adversary *more* power, but is consistent with our relaxed view of what it means to be "functionality-preserving" in the public-key setting.
- Second, we allow the adversary to make multiple marking queries: namely, the adversary can see the same signing key marked with different and adversarially-chosen identities, and, even then, we require that the adversary cannot produce a new circuit whose watermark is not one of the ones corresponding to a signing circuit already given to the adversary. Namely, if an adversary sees a signing key marked with identities τ_1 and τ_2, it cannot create a new functional signing circuit where the watermark is not one of τ_1 or τ_2. We discuss this notion of *collusion resistance* in greater detail in Remark 3.8.
- Finally, because of our relaxed notion of functionality-preserving, signatures output by the unmarked key can look different from signatures output by the marked key, so we additionally give the adversary access to the signing oracle with the unmarked key.

We give our formal definition below:

Definition 3.6 (Watermarking Signatures Security Experiment). *Let* $\Pi_{\mathsf{WM}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Mark}, \mathsf{Extract})$ *be a watermarkable signature scheme with message space* $\mathcal{M}$*, mark space* $\mathcal{T}$*, and signature space* $\mathcal{SIG}$*. Then, for an adversary* $\mathcal{A}$*, we define the watermarking signatures security experiment as follows:*

1. *The challenger begins by sampling* $\mathsf{wpp} \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{wpp})$. *The challenger gives* $(\mathsf{wpp}, \mathsf{vk})$ *to the adversary.*
2. *The adversary is now given access to the following oracles:*
   - ***Marking oracle:*** *On input a mark* $\tau \in \mathcal{T}$*, the challenger replies with* $C_\tau \leftarrow \mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}, \tau)$.
   - ***Signing oracle:*** *On input a message* $m \in \mathcal{M}$*, the challenger replies with* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$.

3. The adversary outputs a circuit $C^* : \mathcal{M} \to \mathcal{SIG}$.

The output of the experiment, denoted $\mathsf{ExptWM}_{\mathsf{Sig}}[\lambda, \mathcal{A}]$ *is* $\mathsf{Extract}(\mathsf{wpp}, \mathsf{vk}, C^*)$.

Definition 3.7 (ε-Unremovability). *Let Π_{WM} be a watermarkable signature scheme with message space $\mathcal{M}$ and signature space $\mathcal{SIG}$. We say an adversary $\mathcal{A}$ is ε-unremovable admissible if the adversary's circuit $C^* : \mathcal{M} \to \mathcal{SIG}$ in the watermarking signatures security experiment satisfies*

$$\Pr[m \xleftarrow{R} \mathcal{M} : \mathsf{Verify}(\mathsf{wpp}, \mathsf{vk}, m, C^*(m)) = 1] \geq \varepsilon.$$

In the watermarking signatures security experiment, let $\mathcal{Q} \subseteq \mathcal{T}$ be the set of marks the adversary submitted to the marking oracle. Then, Π_{WM} satisfies ε-unremovability if for all efficient and ε-unremovability admissible adversaries $\mathcal{A}$, $\Pr[\mathsf{ExptWM}_{\mathsf{Sig}}[\lambda, \mathcal{A}] \notin \mathcal{Q}] = \mathsf{negl}(\lambda)$.

Remark 3.8 (Collusion Resistance). We say that a watermarking scheme is *fully* collusion resistant if unremovability holds against all efficient adversaries that can make any a priori unbounded $\mathsf{poly}(\lambda)$ queries to the marking oracle. We say that it is q-bounded collusion resistant if unremovability hold only against efficient adversaries that make at most q marking queries. Existing watermarking schemes for cryptographic functionalities from standard assumptions are only 1-collusion resistant [37, 52, 38].

Remark 3.9 (Small Unremovability Thresholds). Previously, Cohen et al. [27] showed that message-embedding watermarking schemes satisfying ε-unremovability are possible only when $\varepsilon \geq 1/2 + 1/\mathsf{poly}(\lambda)$. This lower bound does not apply to our notion of ε-unremovability (Definition 3.7). In fact, our constructions satisfy ε-unremovability for *any* non-negligible $\varepsilon = 1/\mathsf{poly}(\lambda)$. The reason is that our mark-extraction algorithm takes in the verification key as input, while the Cohen et al. definition does not (i.e., their extraction algorithm only takes the circuit as input).

To provide some additional detail, we first recall the attack from Cohen et al. when $\varepsilon = 1/2$. Let C be the challenge circuit marked with a message m in the unremovability security game, and let C' be an arbitrary circuit (for a different function) marked with a message $m' \neq m$. In both the secret and public marking setting, the adversary can generate C' by either using the marking oracle (secret-marking setting) or using the public marking algorithm (public-marking setting). To carry out the attack, the adversary constructs a challenge circuit C^* that agrees with C on half of the points (chosen randomly) and agrees with C' on the other half. By symmetry, the extraction algorithm on C^* outputs m and m' with equal probability, where the probability is taken over the coins of the $\mathsf{Extract}$ algorithm and the adversary's randomness used to determine m, m', and C^*. This means that the probability that the extraction algorithm outputs m is at most $1/2$, and so the adversary succeeds with probability at least $1/2$.

The above attack critically relies on the fact that the adversary is able to obtain a marked circuit C' where the extraction algorithm on C' outputs $m' \neq m$.

In order to mount the same attack in our setting, the adversary needs to be able to obtain a circuit C' such that $\mathsf{Extract}(\mathsf{wpp}, \mathsf{vk}, C') = m'$, where vk is the verification key chosen by the challenger. In the security game, there is no mechanism for the adversary to obtain a marked circuit with respect to vk other than by making a marking query on m'. If the adversary makes a marking query on m', then as long as the extraction algorithm recovers *either* m or m', the adversary does not break unremovability. Observe that if, on the contrary, vk is not provided as input to $\mathsf{Extract}$, then the adversary can easily construct a circuit with an embedded mark m' (by marking an arbitrary key of its choosing) and mount the Cohen et al. attack. This distinction, where the extraction algorithm is defined with respect to a *specific* verification key, enables us to circumvent the lower bound for ε-unremovability when $\varepsilon \leq 1/2$.

3.1 Building Block: Constrained Signatures

As discussed in Section 1.1, the main building block we use to construct a watermarkable signature scheme is a prefix-constrained signature (which can be built generically from any signature scheme, or more generally, any one-way function). We recall the formal definition below:

Definition 3.10 (Constrained Signatures [10, 59]). *A constrained signature scheme with message space $\mathcal{M}$ and constraint family $\mathcal{F} \subseteq \mathsf{Funs}[\mathcal{M}, \{0,1\}]$ is a tuple of algorithm $\Pi_{\mathsf{CSig}} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Constrain}, \mathsf{ConstrainSign})$ with the following properties:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{msk})$*: On input the security parameter λ, the setup algorithm outputs the verification key and the master secret key* msk*.*
- $\mathsf{Sign}(\mathsf{msk}, m) \to \sigma$*: On input the master signing key* msk *and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ.*
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to b$*: On input the verification key* vk*, a message $m \in \mathcal{M}$, and a signature σ, the verification algorithm outputs a bit $b \in \{0,1\}$.*
- $\mathsf{Constrain}(\mathsf{msk}, f) \to \mathsf{sk}_f$*: On input the master signing key* msk *and a function $f \in \mathcal{F}$, the constrain algorithm outputs a constrained key* sk_f*.*
- $\mathsf{ConstrainSign}(\mathsf{sk}_f, m) \to \sigma$*: On input a constrained key* sk_f *and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ.*

Definition 3.11 (Correctness). *A constrained signature scheme $\Pi_{\mathsf{CSig}} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Constrain}, \mathsf{ConstrainSign})$ with message space $\mathcal{M}$ and constraint family $\mathcal{F}$ is correct if for all messages $m \in \mathcal{M}$ and taking $(\mathsf{vk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$,*

$$\Pr[\mathsf{Verify}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{msk}, m)) = 1] = 1.$$

In addition, for all constraints $f \in \mathcal{F}$ where $f(m) = 1$, if we compute $\mathsf{sk}_f \leftarrow \mathsf{Constrain}(\mathsf{msk}, f)$,

$$\Pr[\mathsf{Verify}(\mathsf{vk}, m, \mathsf{ConstrainSign}(\mathsf{sk}_f, m)) = 1] = 1.$$

Definition 3.12 (Constrained Unforgeability). *Let* $\Pi_{\mathsf{CSig}} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Constrain}, \mathsf{ConstrainSign})$ *be a constrained signature scheme with message space* $\mathcal{M}$ *and constraint family* $\mathcal{F}$*. We define the constrained unforgeability experiment between an adversary* $\mathcal{A}$ *and a challenger as follows:*

1. *At the beginning of the experiment, the challenger samples* $(\mathsf{vk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ *and gives* vk *to the adversary.*
2. *The adversary is then given access to the following oracles:*
   - ***Constrain oracle:*** *On input a function* $f \in \mathcal{F}$*, the challenger replies with* $\mathsf{sk}_f \leftarrow \mathsf{Constrain}(\mathsf{msk}, f)$*.*
   - ***Signing oracle:*** *On input a message* $m \in \mathcal{M}$*, the challenger replies with a signature* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msk}, m)$*.*
   - *At the end of the game, the adversary outputs a message-signature pair* (m^*, σ^*)*.*

The output of the experiment, denoted $\mathsf{ExptCSig}[\mathcal{A}, \lambda]$ *is* 1 *if the following conditions hold:*

- *The adversary did not make a signing query on message* m^**.*
- *The adversary did not make a constrain query on any function* $f \in \mathcal{F}$ *where* $f(m^*) = 1$*.*
- $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$*.*

We say that Π_{CSig} *is secure if for all efficient adversaries* $\mathcal{A}$*,* $\Pr[\mathsf{ExptCSig}[\mathcal{A}, \lambda] = 1] = \mathsf{negl}(\lambda)$*.*

3.2 Watermarking Signatures from Constrained Signatures

In this section, we show how to construct a fully collusion-resistant watermarking scheme for digital signatures from prefix-constrained signatures.

Construction 3.13 (Watermarkable Signatures from Prefix-Constrained Signatures) *Fix a message space* $\mathcal{M}$ *and a mark space* $\mathcal{T}$*. Let* $\varepsilon = 1/\mathsf{poly}(\lambda)$ *be an unremovability parameter. We define the following primitives:*

- *Let* $\mathcal{Z}$ *be a tag space, and let* $\mathcal{T}' = \mathcal{T} \cup \{\perp\}$*.*
- *For a mark* $\tau^* \in \mathcal{T}$*, let* $f_{\tau^*} \colon \mathcal{T}' \times \mathcal{M} \to \{0, 1\}$ *be the function where* $f_{\tau^*}(\tau, m) = 1$ *if* $\tau = \tau^*$ *and* 0 *otherwise.*
- *Let* $\Pi_{\mathsf{CSig}} = (\mathsf{CSig.Setup}, \mathsf{CSig.Sign}, \mathsf{CSig.Verify}, \mathsf{CSig.Constrain}, \mathsf{CSig.ConstrainSign})$ *be a constrained signature scheme with message space* $\mathcal{M}' = \mathcal{T}' \times \mathcal{M}$ *and function class* $\mathcal{F} = \{\tau^* \in \mathcal{T} \colon f_{\tau^*}\}$*. Let* $\mathcal{SIG}'$ *be the signature space of* Π_{CSig}*.*

We construct a watermarkable signature scheme $\Pi_{\mathsf{WM}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Mark}, \mathsf{Extract})$ *with signature space* $\mathcal{SIG} = \mathcal{Z} \times \mathcal{T}' \times \mathcal{SIG}'$ *as follows:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{wpp}$*: On input the security parameter* λ*, sample* $z \xleftarrow{\mathrm{R}} \mathcal{Z}$*, and output* $\mathsf{wpp} = z$*.*

- $\mathsf{KeyGen}(1^\lambda, \mathsf{wpp}) \to (\mathsf{vk}, \mathsf{sk})$*: On input the security parameter* λ *and public parameters* $\mathsf{wpp} = z$*, the key-generation algorithm outputs a signing/verification key-pair* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{CSig.Setup}(1^\lambda)$.
- $\mathsf{Sign}(\mathsf{wpp}, \mathsf{sk}, m) \to \sigma$*: On input the public parameters* $\mathsf{wpp} = z$*, a signing key* sk*, and a message* $m \in \mathcal{M}$*, the signing algorithm signs* $\sigma' \leftarrow \mathsf{CSig.Sign}(\mathsf{sk}, (\bot, m))$*, and outputs the signature* $\sigma = (z, \bot, \sigma')$.
- $\mathsf{Verify}(\mathsf{wpp}, \mathsf{vk}, m, \sigma) \to b$*: On input the public parameters* $\mathsf{wpp} = z$*, a verification key* vk*, a message* $m \in \mathcal{M}$*, and a signature* $\sigma = (z', \tau', \sigma')$*, the verification algorithm outputs* 0 *if* $z' \neq z$*, and, otherwise, it outputs the bit* $b \leftarrow \mathsf{CSig.Verify}(\mathsf{vk}, (\tau', m), \sigma')$.
- $\mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}, \tau) \to C$*: On input the public parameters* $\mathsf{wpp} = z$*, a signing key* sk*, and a mark* $\tau \in \mathcal{T}$*, the marking algorithm computes* $\mathsf{sk}_\tau \leftarrow \mathsf{CSig.Constrain}(\mathsf{sk}, f_\tau)$ *and outputs a circuit* $C_\tau \colon \mathcal{M} \to \mathcal{SIG}$ *where* $C_\tau(\cdot) := (z, \tau, \mathsf{CSig.ConstrainSign}(\mathsf{sk}_\tau, (\tau, \cdot)))$.
- $\mathsf{Extract}(\mathsf{wpp}, \mathsf{vk}, C) \to \tau/\bot$*: On input the public parameters* $\mathsf{wpp} = z$*, a verification key* vk*, and a circuit* $C \colon \mathcal{M} \to \mathcal{SIG}$*, the extraction algorithm performs the following procedure* $T = \lambda/\varepsilon = \mathsf{poly}(\lambda)$ *times:*
  - *For* $i \in [T]$*, sample* $m_i \xleftarrow{R} \mathcal{M}$ *and compute* $(z'_i, \tau'_i, \sigma'_i) \leftarrow C(m)$*. If* $z'_i = z$ *and* $\mathsf{CSig.Verify}(\mathsf{vk}, (\tau'_i, m_i), \sigma'_i) = 1$*, abort and output* τ'_i.

  If the above procedure does not abort with some output τ*, then output* $\bot$.

Correctness and security analysis. We now state our correctness and security theorems, but defer their formal analysis to the full version of this paper.

Theorem 3.14 (Correctness). *Suppose* $1/|\mathcal{Z}| = \mathsf{negl}(\lambda)$ *and* Π_{CSig} *is correct. Then, the watermarkable signature scheme* Π_{WM} *from Construction 3.13 is correct (Definition 3.3).*

Theorem 3.15 (Signature Unforgeability). *If* Π_{CSig} *satisfies constrained unforgeability (Definition 3.12), then the watermarkable signature scheme* Π_{WM} *from Construction 3.13 satisfies signature unforgeability in the presence of a malicious watermarking authority (Definition 3.5).*

Theorem 3.16 (Unremovability). *Take any* $\varepsilon = 1/\mathsf{poly}(\lambda)$*. If* $1/|\mathcal{M}| = \mathsf{negl}(\lambda)$ *and* Π_{CSig} *satisfies constrained unforgeability (Definition 3.12), then the watermarkable signature scheme* Π_{WM} *from Construction 3.13 is* ε*-unremovable.*

3.3 Instantiations and Extensions

As noted by Bellare and Fuchsbauer [10], fully collusion resistant constrained signatures (for arbitrary circuit constraints) satisfying unforgeability follow immediately from any standard signature scheme (which can in turn be based on one-way functions [31]). We briefly recall the "certificate-based" construction here. The public parameters for the constrained signature scheme is a verification key vk for a standard signature scheme, and the master secret key is the associated signing key sk. To issue a constrained key for a function f, the authority generates

a fresh pair of signing and verification keys $(\mathsf{vk}', \mathsf{sk}')$, and constructs a signature ("certificate") σ on (vk', f) with the master signing key sk. The constrained key is the tuple $(\mathsf{vk}', \mathsf{sk}', f, \sigma)$. A signature on m using the constrained key consists of a signature σ' on m using sk' together with the tuple $(\mathsf{vk}', f, \sigma)$. To verify, one checks that σ is a valid signature on (vk', f) with respect to the master verification key vk, that $f(m) = 1$, and that σ' is a valid signature on m with respect to vk'. From this construction, we obtain the following corollary:

Corollary 3.17 (Watermarkable Signatures from One-Way Functions). *Take any $\varepsilon = 1/\mathsf{poly}(\lambda)$ and mark space $\mathcal{T} = \{0,1\}^\ell$, where $\ell = \mathsf{poly}(\lambda)$. Assuming one-way functions exist, there exists a fully collusion resistant watermarkable family of signatures with mark space $\mathcal{T}$ that satisfies ε-unremovability (Definition 3.7) and where the underlying signature scheme is unforgeable even against a malicious authority (Definition 3.5).*

In the full version of this paper, we describe a variant of our watermarkable signature scheme that achieves mark-unforgeability in the secret-marking setting (i.e., no efficient adversary is able to come up with a marked circuit of its own).

4 Watermarking Public-Key Predicate Encryption

In this section, we show how to watermark a public-key predicate encryption scheme. In particular, this notion implies watermarking for simpler classes of public-key primitives like public-key encryption,[12] identity-based encryption, and attribute-based encryption. We begin by formally introducing the notion of watermarking public-key predicate encryption schemes. Our definitions have a very similar flavor to our corresponding definitions for watermarking digital signature schemes from Section 3 and the previous definitions of Cohen et al. [27].

Definition 4.1 (Watermarkable Public-Key Predicate Encryption). *A watermarkable public-key predicate encryption scheme with message space $\mathcal{M}$, attribute space $\mathcal{X}$, function space $\mathcal{F} \subseteq \mathsf{Funs}[\mathcal{X}, \{0,1\}]$, and mark space $\mathcal{T}$ is a tuple of algorithms* $(\mathsf{WMSetup}, \mathsf{PESetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Mark}, \mathsf{Extract})$ *with the following properties:*

- $\mathsf{WMSetup}(1^\lambda) \to \mathsf{wpp}$*: On input the security parameter λ, the watermarking setup algorithm outputs a set of watermarking public parameters* wpp*.*
- *The watermarking parameters* wpp *induces a public-key predicate-encryption scheme* $(\mathsf{PESetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *with message space $\mathcal{M}$, attribute space $\mathcal{X}$, and function space $\mathcal{F}$. We implicitly allow* $\mathsf{PESetup}$*,* KeyGen*,* $\mathsf{Encrypt}$*, and* $\mathsf{Decrypt}$ *to take the watermarking parameters* wpp *as input. Let $\mathcal{PK}$ denote the space of master public keys, $\mathcal{SK}$ denote the space of function keys, and $\mathcal{CT}$ denote the ciphertext space of the induced predicate encryption scheme.*

[12] As noted in Section 1.1, a traitor-tracing scheme that supports public tracing (e.g., [18, 49, 2]) directly gives a watermarkable public-key encryption scheme.

- $\mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}, \tau) \to C_\tau$*: On input the watermarking parameters* wpp*, a secret key* $\mathsf{sk} \in \mathcal{SK}$*, and a mark* $\tau \in \mathcal{T}$*, the marking algorithm outputs a circuit* $C_\tau \colon \mathcal{CT} \to \mathcal{M} \cup \{\bot\}$*.*
- $\mathsf{Extract}(\mathsf{wpp}, \mathsf{mpk}, C) \to \tau/\bot$*: On input the watermarking parameters* wpp*, a master public key* $\mathsf{mpk} \in \mathcal{PK}$ *and a circuit* $C \colon \mathcal{CT} \to \mathcal{M} \cup \{\bot\}$*, the extraction algorithm either outputs a mark* $\tau \in \mathcal{T}$ *or a special symbol* $\bot$*.*

Definition 4.2 (Correctness). *Let* $\Pi_{\mathsf{WM}} = (\mathsf{WMSetup}, \mathsf{PESetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Mark}, \mathsf{Extract})$ *be a watermarkable predicate encryption scheme for function family* $\mathcal{F}$*. Then,* Π_{WM} *is correct if for* $\mathsf{wpp} \leftarrow \mathsf{Setup}(1^\lambda)$*, the induced public-key predicate encryption scheme* $(\mathsf{PESetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is correct and the following properties also hold:*

- ***Extraction correctness:*** *For all marks* $\tau \in \mathcal{T}$ *and all functions* $f \in \mathcal{F}$*, if we take* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PESetup}(1^\lambda, \mathsf{wpp})$ *and* $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{wpp}, \mathsf{msk}, f)$*, then*
$$\Pr[\mathsf{Extract}(\mathsf{wpp}, \mathsf{mpk}, \mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}_f, \tau)) \neq \tau] = \mathsf{negl}(\lambda).$$
- ***Meaningfulness:*** *For all fixed circuits* $C \colon \mathcal{CT} \to \mathcal{M} \cup \{\bot\}$ *(independent of the public parameters* wpp*) and all master public keys* $\mathsf{mpk} \in \mathcal{PK}$*,*
$$\Pr[\mathsf{Extract}(\mathsf{wpp}, \mathsf{mpk}, C) \neq \bot] = \mathsf{negl}(\lambda),$$
and for all functions $f \in \mathcal{F}$*,* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PESetup}(1^\lambda, \mathsf{wpp})$*, and* $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{wpp}, \mathsf{msk}, f)$*,*
$$\Pr[\mathsf{Extract}(\mathsf{wpp}, \mathsf{mpk}, \mathsf{Decrypt}(\mathsf{wpp}, \mathsf{sk}_f, \cdot)) \neq \bot] = \mathsf{negl}(\lambda).$$
- ***Functionality-preserving:*** *For all marks* $\tau \in \mathcal{T}$*, all messages* $m \in \mathcal{M}$*, all attributes* $x \in \mathcal{X}$*, and all functions* $f \in \mathcal{F}$ *where* $f(x) = 1$*, if we take* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PESetup}(1^\lambda, \mathsf{wpp})$*,* $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{wpp}, \mathsf{mpk}, f)$ *and* $C \leftarrow \mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}_f, \tau)$*, we have that*
$$\Pr[C(\mathsf{Encrypt}(\mathsf{wpp}, \mathsf{mpk}, x, m)) \neq m] = \mathsf{negl}(\lambda).$$

Definition 4.3 (Security). *Let* $\Pi_{\mathsf{WM}} = (\mathsf{WMSetup}, \mathsf{PESetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Mark}, \mathsf{Extract})$ *be a watermarkable predicate encryption scheme. Then,* Π_{WM} *is secure if the induced predicate encryption scheme* $(\mathsf{PESetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is secure. We say that* Π_{WM} *satisfies* security in the presence of a malicious authority *if the induced predicate encryption scheme is secure even if the adversary is allowed to choose the watermarking parameters* wpp*.*

Definition 4.4 (Watermarking PE Security Experiment). *Let* $\Pi_{\mathsf{WM}} = (\mathsf{WMSetup}, \mathsf{PESetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Mark}, \mathsf{Extract})$ *be a watermarkable predicate encryption scheme with message space* $\mathcal{M}$*, attribute space* $\mathcal{X}$*, function space* $\mathcal{F} \subseteq \mathsf{Funs}[\mathcal{X}, \{0,1\}]$*, and mark space* $\mathcal{T}$*. Let* $\mathcal{CT}$ *denote the ciphertext space for* Π_{WM}*. For an adversary* $\mathcal{A}$*, we define the watermarking PE security experiment as follows:*

1. *The challenger begins by sampling* $\mathsf{wpp} \leftarrow \mathsf{WMSetup}(1^\lambda)$ *and* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{wpp})$. *It gives* $(\mathsf{wpp}, \mathsf{mpk})$ *to the adversary* $\mathcal{A}$.
2. *The adversary specifies a function* $f \in \mathcal{F}$ *it would like to target. The challenger computes a secret key* $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{wpp}, \mathsf{msk}, f)$, *but does* not *give* sk_f *to the adversary.*
3. *The adversary can then make marking oracle queries. On each marking query, the adversary specifies a mark* $\tau \in \mathcal{T}$, *and the challenger replies with the circuit* $C_\tau \leftarrow \mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}_f, \tau)$.
4. *At the end of the experiment, the adversary outputs a circuit* $C^* : \mathcal{CT} \to \mathcal{M} \cup \{\perp\}$ *and an attribute* $x \in \mathcal{X}$.

The output of the experiment, denoted $\mathsf{ExptWM}_{\mathsf{PE}}(\lambda, \mathcal{A})$, *is* $\mathsf{Extract}(\mathsf{wpp}, \mathsf{mpk}, C')$.

Definition 4.5 (ε-Unremovability). *Let* Π_{WM} *be a watermarkable public-key encryption scheme with message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{CT}$. *We say an adversary* $\mathcal{A}$ *is* ε*-unremovable admissible if the adversary in the watermarking security game outputs an attribute* $x \in \mathcal{X}$ *and a circuit* $C^* : \mathcal{CT} \to \mathcal{M} \cup \{\perp\}$ *where*

$$\Pr[m \xleftarrow{R} \mathcal{M} : C^*(\mathsf{Encrypt}(\mathsf{wpp}, \mathsf{mpk}, x, m)) = m] \geq \varepsilon.$$

In the watermarking PE security experiment, let $\mathcal{Q} \subseteq \mathcal{T}$ *be the set of marks the adversary submitted to the marking oracle. Then,* Π_{WM} *satisfies* ε*-unremovability if for all efficient and* ε*-unremovability admissible adversaries* $\mathcal{A}$, $\Pr[\mathsf{ExptWM}_{\mathsf{PE}}[\lambda, \mathcal{A}] \notin \mathcal{Q}] = \mathsf{negl}(\lambda)$.

Remark 4.6 (Collusion Resistance). We say that a watermarkable predicate encryption scheme Π_{WM} is $(q_{\mathsf{key}}, q_{\mathsf{mark}})$-collusion resistant if the induced predicate encryption scheme is q_{key}-bounded collusion resistant and the watermarking adversary in the unremovability game can make at most q_{mark} marking queries. When q_{key} and q_{mark} can be arbitrary and a priori unbounded polynomials, we say Π_{WM} is *fully* collusion resistant.

Remark 4.7 (Stronger Notions of Unremovability). Our definitions of unremovability (Definitions 4.4 and 4.5) only allows the adversary to request (multiple) marked version of a *single* predicate encryption key sk_f. A stronger notion would allow the adversary to specify both a decryption function f as well as a mark τ on each marking oracle query. Such a scheme would then be secure even if an adversary could obtain marked versions of different decryption keys. Our construction does not achieve this stronger notion and we leave this as an open problem.

Remark 4.8 (Watermarking Functional Encryption). A further generalization of watermarking predicate encryption is to watermark the decryption keys in a *functional encryption* scheme. One challenge here is characterizing the set of decryption keys that can be marked. For example, it is not possible to watermark a decryption key for a constant-valued function, since the adversary can implement the decryption functionality with a circuit that just computes the constant function (which, of course, removes the watermark). It seems plausible that

we can watermark decryption keys corresponding to functions with "high min-entropy:" namely, functions $f\colon \mathcal{X} \to \mathcal{Y}$ where for any $y \in \mathcal{Y}$, $\Pr[x \xleftarrow{R} \mathcal{X} : f(x) = y] = \mathsf{negl}(\lambda)$. While it is straightforward to modify our construction of watermarkable predicate encryption to support marking function keys of this form, in the resulting construction, we would additionally have to provide the $\mathsf{Extract}$ algorithm a description of the function f associated with a particular decryption circuit. Whether this a reasonable modeling assumption will depend on the particular application. It is an interesting question to both develop a better understanding of the types of function families that can be watermarked as well as identify a suitable schema for watermarking general functional encryption schemes.

4.1 Building Blocks: Functional Encryption and Traitor Tracing

In this section, we review the main building blocks we use to construct our scheme for watermarking predicate encryption.

Hierarchical functional encryption. Our main building block for constructing a watermarkable predicate encryption scheme is a general-purpose hierarchical functional encryption scheme. Below, we recall the formal definition from [4, 22]:

Definition 4.9 (Hierarchical Functional Encryption [4, 22]). *A hierarchical functional encryption (FE) scheme with domain $\mathcal{X}$, range $\mathcal{Y}$, and function space $\mathcal{F}$ is a tuple of algorithms $\Pi_{\mathsf{HFE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Delegate})$ with the following properties:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$*: On input the security parameter λ, the setup algorithm outputs the master public key mpk and the master secret key msk.*
- $\mathsf{KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$*: On input the master secret key msk and a function $f \in \mathcal{F}$, the key-generation algorithm outputs a secret key sk_f.*
- $\mathsf{Encrypt}(\mathsf{mpk}, x) \to \mathsf{ct}_x$*: On input the master public key mpk and an input $x \in \mathcal{X}$, the encryption algorithm outputs a ciphertext ct_x.*
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to y/\bot$*: On input a secret key sk and a ciphertext ct, the decryption algorithm either outputs a value $y \in \mathcal{Y}$ or a special symbol $\bot$.*
- $\mathsf{Delegate}(\mathsf{sk}_f, g) \to \mathsf{sk}_{g \circ f}$*: On input a secret key sk_f and a function $g \in \mathcal{F}$, the delegate algorithm outputs a secret key $\mathsf{sk}_{g \circ f}$.*

A hierarchical functional encryption scheme should satisfy the following properties:

- ***Correctness:*** *For all $x \in \mathcal{X}$ and functions $f \in \mathcal{F}$, if we sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$, and $\mathsf{ct}_x \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, x)$, then*

$$\Pr[\mathsf{Decrypt}(\mathsf{sk}_f, \mathsf{ct}_x) = f(x)] = 1.$$

- ***Delegation correctness:*** *For all $x \in \mathcal{X}$ and functions $f, g \in \mathcal{F}$ where $g \circ f \in \mathcal{F}$, if we sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$, $\mathsf{sk}_{g \circ f} \leftarrow \mathsf{Delegate}(\mathsf{sk}_f, g)$, and $\mathsf{ct}_x \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, x)$, then*

$$\Pr[\mathsf{Decrypt}(\mathsf{sk}_{g \circ f}, \mathsf{ct}_x) = g(f(x))] = 1.$$

Note that this definition only considers correctness for single-hop *delegation. We can define a corresponding notion of multi-hop delegation correctness. However, single-hop delegation already suffices for our construction.*
- ***Security:*** *Due to space limitations, we defer the security definition of hierarchical functional encryption to the full version of this paper.*

Remark 4.10 (Collusion Resistance). We say that a (hierarchical) functional encryption scheme Π_{HFE} is q-bounded collusion resistant if the security property holds against all efficient adversaries that make at most q key-generation queries and that it is fully collusion resistant if security holds against all adversaries that can make an a priori unbounded polynomial number of key-generation queries.

The jump-finding problem. We recall the jump-finding problem introduced in the work of Nishimaki et al. [49] for constructing flexible traitor tracing schemes (i.e., traitor tracing schemes where the space of identities that can be traced is exponential). We rely on similar techniques to watermark the decryption keys in a predicate encryption scheme.

Definition 4.11 (Noisy Jump Finding Problem [49, Definition 3.6]). *The $(N, q, \delta, \varepsilon)$-jump-finding problem is defined as follows. An adversary chooses a set $C \subseteq [N]$ of q unknown points. Then, the adversary provides an oracle $P\colon [0, N] \to [0, 1]_{\mathbb{R}}$ with the following properties:*

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

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

Theorem 4.12 (Noisy Jump Finding Algorithm [49, Theorem 3.7]). *There is an efficient algorithm $\mathsf{QTrace}^Q(\lambda, N, q, \delta, \varepsilon)$ that runs in time $t = \mathsf{poly}(\lambda, \log N, q, 1/\delta)$ and makes at most t queries to Q that solves the $(N, q, \delta, \varepsilon)$-noisy-jump-finding problem whenever $\varepsilon > \delta(5 + 2(\lceil \log N - 1 \rceil)q)$. In particular, $\mathsf{QTrace}^Q(\lambda, N, q, \delta, \varepsilon)$ will output at least one element in C with probability $1 - \mathsf{negl}(\lambda)$ and will never output an element outside C. Moreover, any element x output by $\mathsf{QTrace}^Q(\lambda, N, q, \delta, \varepsilon)$ has the property that $P(x) - P(x-1) > \delta$, where $P(x) = \Pr[Q(x) = 1]$.*

4.2 Watermarking Predicate Encryption from Hierarchical FE

In this section, we show how to construct a (bounded) collusion resistant watermarkable predicate encryption scheme for general predicates from any (bounded) collusion resistant hierarchical functional encryption scheme for general circuits.

Construction 4.13 (Watermarkable PE from Hierarchical FE) *Let $\mathcal{M} = \{0,1\}^n$ be a message space, $\mathcal{X} = \{0,1\}^\ell \setminus \{1^\ell\}$ be an attribute space, $\mathcal{F} \subseteq \mathsf{Funs}[\{0,1\}^\ell, \{0,1\}]$ be a class of predicates, and $\mathcal{T} \subseteq \mathcal{X} = \{0,1\}^\ell \setminus \{1^\ell\}$ be a mark space. Let $\varepsilon = 1/\mathsf{poly}(\lambda)$ be an unremovability parameter. We rely on the following ingredients:*

- *Let $\mathcal{Z} = \mathcal{M} = \{0,1\}^n$ be a tag space.*
- *Let $q_{\mathsf{mark}} = \mathsf{poly}(\lambda)$ be a bound on the number of marking oracle queries the watermarking adversary is allowed to make. In Remark 4.17, we describe a simple adaptation of the extraction algorithm that achieves full collusion resistance (assuming a fully collusion resistant hierarchical FE scheme)*
- *For a function $f \in \mathcal{F}$, let $g_f \colon \{0,1\}^{\ell+n+1} \to \{0,1\}^{\ell+n}$ be the function defined as follows:*

$$g_f(x,m,b) = \begin{cases} (x,m) & b = 0 \\ (0^\ell, 0^n) & b = 1 \text{ and } f(x) = 0 \\ (1^\ell, m) & b = 1 \text{ and } f(x) = 1, \end{cases} \tag{4.1}$$

  where $x \in \{0,1\}^\ell$, $m \in \{0,1\}^n$, and $b \in \{0,1\}$. Define the function class $\mathcal{G} = \{f \in \mathcal{F} : g_f\}$.
- *For a mark $\tau \in \{0,1\}^\ell$, define the function $h_\tau \colon \{0,1\}^{\ell+n} \to \{0,1\}^{\ell+n}$ as follows:*

$$h_\tau(x,m) = \begin{cases} (0^\ell, 0^n) & x \le \tau \\ (1^\ell, m) & x > \tau, \end{cases} \tag{4.2}$$

  where $x \in \{0,1\}^\ell$ and $m \in \{0,1\}^n$, and are interpreted as values in $[0, 2^\ell - 1]$ and $[0, 2^n - 1]$, respectively.
- *Let $\Pi_{\mathsf{HFE}} = (\mathsf{HFE.Setup}, \mathsf{HFE.KeyGen}, \mathsf{HFE.Encrypt}, \mathsf{HFE.Decrypt}, \mathsf{HFE.Delegate})$ be a hierarchical FE scheme with domain $\{0,1\}^{\ell+n+1}$, range $\{0,1\}^{\ell+n}$, and function class $\mathcal{G}$. Let $\mathcal{CT}$ be the space of ciphertexts for Π_{HFE}.*

We construct a watermarkable predicate encryption scheme $\Pi_{\mathsf{WM}} = (\mathsf{WMSetup}, \mathsf{PESetup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Mark}, \mathsf{Extract})$ as follows:

- $\mathsf{WMSetup}(1^\lambda) \to \mathsf{wpp}$*: On input the security parameter λ, sample $z \xleftarrow{R} \{0,1\}^n$ and output $\mathsf{wpp} = z$.*
- $\mathsf{PESetup}(1^\lambda, \mathsf{wpp}) \to (\mathsf{mpk}, \mathsf{msk})$*: On input the security parameter λ and the watermarking parameters $\mathsf{wpp} = z$, the key-generation algorithm outputs a key-pair $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{HFE.Setup}(1^\lambda)$.*
- $\mathsf{KeyGen}(\mathsf{wpp}, \mathsf{msk}, f) \to \mathsf{sk}_f$*: On input the watermarking parameters $\mathsf{wpp} = z$, a master secret key msk, and a function $f \in \mathcal{F}$, the key-generation algorithm outputs a secret key $\mathsf{sk}_f \leftarrow \mathsf{HFE.KeyGen}(\mathsf{msk}, g_f)$, where g_f is defined in Eq. (4.1).*
- $\mathsf{Encrypt}(\mathsf{wpp}, \mathsf{mpk}, x, m) \to \mathsf{ct}_{x,m}$*: On input the watermarking parameters $\mathsf{wpp} = z$, a master public key mpk, an attribute $x \in \{0,1\}^\ell$, and a message $m \in \{0,1\}^n$, the encryption algorithm outputs a ciphertext $\mathsf{ct}_{x,m} \leftarrow \mathsf{HFE.Encrypt}(\mathsf{mpk}, (x, m \oplus z, 1)) \in \mathcal{CT}$.*

– $\mathsf{Decrypt}(\mathsf{wpp}, \mathsf{sk}, \mathsf{ct}) \to m/\perp$: *On input the watermarking parameter* $\mathsf{wpp} = z$*, a secret key* sk*, and a ciphertext* ct*, the decryption algorithm computes* $y \leftarrow \mathsf{HFE.Decrypt}(\mathsf{sk}, \mathsf{ct})$*. If* $y = \perp$*, then output* $\perp$*. Otherwise, it parses* $y = (x, m')$ *where* $x \in \{0,1\}^\ell$ *and* $m' \in \{0,1\}^n$*. It outputs* $m' \oplus z$ *if* $x = 1^\ell$ *and* $\perp$ *otherwise.*

– $\mathsf{Mark}(\mathsf{wpp}, \mathsf{sk}, \tau) \to C_\tau$: *On input the watermarking parameters* $\mathsf{wpp} = z$*, a secret key* sk*, and a mark* $\tau \in \{0,1\}^t$*, the marking algorithm constructs a new key* $\mathsf{sk}_\tau \leftarrow \mathsf{HFE.Delegate}(\mathsf{sk}, h_\tau)$*, where* h_τ *is defined in Eq. (4.2). Finally, it outputs the circuit* $C \colon \mathcal{CT} \to \mathcal{M} \cup \{\perp\}$ *that computes the marked function* $P[z, \mathsf{sk}_\tau]$ *defined as follows:*

> *On input a ciphertext* $\mathsf{ct} \in \mathcal{CT}$*:*
>
> 1. *Compute* $y \leftarrow \mathsf{HFE.Decrypt}(\mathsf{sk}_\tau, \mathsf{ct})$*. If* $y = \perp$*, output* $\perp$*.*
> 2. *Otherwise, parse* $y = (x, m')$*, where* $x \in \{0,1\}^\ell$ *and* $m' \in \{0,1\}^n$*. Output* $m' \oplus z$ *if* $x = 1^\ell$ *and* $\perp$ *otherwise.*

Fig. 1: The marked function $P[z, \mathsf{sk}_\tau]$

– $\mathsf{Extract}(\mathsf{wpp}, \mathsf{mpk}, C) \to \tau/\perp$: *On input the watermarking parameters* $\mathsf{wpp} = z$*, a master public key* mpk*, and a circuit* $C \colon \mathcal{CT} \to \mathcal{M} \cup \{\perp\}$*, the extraction algorithm first performs the following decryption check* $T = \lambda/\varepsilon = \mathsf{poly}(\lambda)$ *times:*

- *For each* $i \in [T]$*, sample* $m_i \xleftarrow{R} \mathcal{M}$ *and compute the ciphertext* $\mathsf{ct}_i \leftarrow \mathsf{HFE.Encrypt}(\mathsf{mpk}, (1^\ell, m_i \oplus z, 0))$ *and* $y_i \leftarrow C(\mathsf{ct}_i)$*.*

If for all $i \in [T]$*,* $y_i \neq m_i$*, then output* $\perp$*. Otherwise, the extraction algorithm constructs the following function* $Q_C \colon \{0,1\}^\ell \to \{0,1\}$*:*

> *On input an input* $x \in \{0,1\}^\ell$ *(interpreted as a value in* $[0, 2^\ell - 1]$*):*
>
> - *Sample a random* $m \xleftarrow{R} \{0,1\}^n$ *and construct the ciphertext* $\mathsf{ct} \leftarrow \mathsf{HFE.Encrypt}(\mathsf{mpk}, (x, m \oplus z, 0))$*.*
> - *Compute* $m' \leftarrow C(\mathsf{ct})$ *and output* 1 *if* $m = m'$ *and* 0 *otherwise.*

Fig. 2: The extraction test function Q_C

Let $\delta = \varepsilon/(5 + 2(\ell - 1)q_{\mathsf{mark}})$ *and compute* $\tau \leftarrow \mathsf{QTrace}^{Q_C}(\lambda, 2^\ell - 1, q_{\mathsf{mark}}, \delta, \varepsilon)$*. If* $\tau = 1^\ell$*, then output* $\perp$*. Otherwise, output* τ*. In Remark 4.17, we describe a variant of the* Extract *algorithm that does not require an explicit bound* q_{mark} *to be provided as input.*

Correctness and security analysis. We now state our correctness and security theorems, but defer their formal analysis to the full version of this paper.

Theorem 4.14 (Correctness). *Suppose* $1/\varepsilon = \mathsf{poly}(\lambda)$, $1/|\mathcal{M}| = \mathsf{negl}(\lambda)$, *and* Π_{HFE} *is correct and secure (Definition 4.9). Then, the watermarkable predicate encryption scheme* Π_{WM} *from Construction 4.13 is correct.*

Theorem 4.15 (Predicate Encryption Security). *If* Π_{HFE} *is secure (Definition 4.9), then Construction 4.13 is secure even in the presence of a malicious authority (Definition 4.3).*

Theorem 4.16 (ε-Unremovability). *Take any* $\varepsilon = 1/\mathsf{poly}(\lambda)$. *If* $1/|\mathcal{M}| = \mathsf{negl}(\lambda)$ *and* Π_{HFE} *is secure (Definition 4.9), then the watermarkable predicate encryption scheme* Π_{WM} *from Construction 4.13 is* ε*-unremovable.*

Remark 4.17 (Extraction Without an A Priori Bound). As described, the $\mathsf{Extract}$ algorithm in Construction 4.13 assumes there is an a priori bound q_{mark} on the number of marked keys the adversary sees (and this bound is provided as an input to the $\mathsf{Extract}$ algorithm). It is straightforward to extend $\mathsf{Extract}$ to operate when no explicit bound is provided. Namely, instead of running QTrace^{Q} with $q = q_{\mathsf{mark}}$, the algorithm instead runs QTrace on successive powers of two $q = 2^0, 2^1, 2^2, \ldots, 2^\ell$ where $\delta = \varepsilon/(5 + 2(\ell - 1)q)$). By Theorem 4.12, if QTrace^{Q} succeeds, it produces a $\tau \in \{0,1\}^\ell$ such that $|\Pr[Q(\tau) = 1] - \Pr[Q(\tau - 1) = 1]| > \delta$. Moreover, we can show that for all efficiently-computable $\tau \notin \mathcal{Q}$, we have that $\Pr[Q(\tau) = 1] - \Pr[Q(\tau - 1) = 1] = \mathsf{negl}(\lambda)$. Thus, if QTrace^{Q} outputs a mark τ, then $\tau \in \mathcal{Q}$, as required. Moreover, once $q > q_{\mathsf{mark}}$, we can appeal to Theorem 4.12 to conclude that with overwhelming probability, the extraction algorithm will output a τ such that this condition holds. This algorithm will terminate after at most $O(\log q_{\mathsf{mark}}) = \mathsf{poly}(\lambda)$ iterations.

4.3 Instantiations and Extensions

In the full version of this paper, we describe two ways to instantiate our watermarkable predicate encryption scheme: one secure against bounded collusions based on the existence of public-key encryption and low-depth pseudorandom generators (PRGs)[13] and one secure against unbounded collusions based on indistinguishability obfuscation (and one-way functions). We also describe a simple variant of our construction that provides watermarking unforgeability in the secret-marking setting. We state the main conclusions below:

Corollary 4.18 (Bounded Collusion-Resistant Watermarkable Predicate Encryption). *Take any* $\varepsilon = 1/\mathsf{poly}(\lambda)$, *any fixed polynomials* $q, q_{\mathsf{key}}, q_{\mathsf{mark}} = \mathsf{poly}(\lambda)$, *and mark space* $\mathcal{T} = \{0,1\}^\ell$, *where* $\ell = \mathsf{poly}(\lambda)$. *Assuming public-key encryption and a PRG computable in* NC^1, *there exists a* $(q_{\mathsf{key}}, q_{\mathsf{mark}})$*-bounded collusion resistant watermarkable family of predicate encryption schemes with mark space* $\mathcal{T}$ *that satisfies* ε*-unremovability (Definition 4.5, Remark 4.6). Moreover, the associated predicate encryption scheme is* q*-bounded collusion resistant*

[13] These are known to follow from most algebraic cryptographic assumptions such as the hardness of factoring, the discrete log assumption, or standard lattice assumptions [45, 46, 47, 7].

and remains secure even in the presence of a malicious watermarking authority (Definition 4.3).

Corollary 4.19 (Fully Collusion-Resistant Watermarkable Predicate Encryption). *Take any $\varepsilon = 1/\mathsf{poly}(\lambda)$ and mark space $\mathcal{T} = \{0,1\}^\ell$. Assuming indistinguishability obfuscation and the existence of one-way functions, there exists a fully collusion resistant watermarkable family of predicate encryption schemes with mark space $\mathcal{T}$ that provides ε-unremovability (Definition 4.5) and where the associated predicate encryption scheme is fully collusion resistant and secure even in the presence of a malicious watermarking authority (Definition 4.3).*

Acknowledgments

We thank Aayush Jain for helpful discussions on this work and the anonymous CRYPTO reviewers for useful feedback on the presentation. R. Goyal was supported by an IBM PhD fellowship. S. Kim was supported by NSF, DARPA, a grant from ONR, and the Simons Foundation. N. Manohar was supported in part by a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1619348, 1228984, 1136174, and 1065276, and BSF grant 2012378. B. Waters was supported by NSF CNS-1908611, CNS-1414082, a DARPA/ARL SAFEWARE award and a Packard Foundation Fellowship. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

References

1. M. Abdalla, A. W. Dent, J. Malone-Lee, G. Neven, D. H. Phan, and N. P. Smart. Identity-based traitor tracing. In *PKC*, 2007.
2. S. Agrawal, S. Bhattacherjee, D. H. Phan, D. Stehlé, and S. Yamada. Efficient public trace and revoke from standard assumptions: Extended abstract. In *ACM CCS*, pages 2277–2293, 2017.
3. S. Agrawal and A. Rosen. Functional encryption for bounded collusions, revisited. In *TCC*, 2017.
4. P. Ananth, D. Boneh, S. Garg, A. Sahai, and M. Zhandry. Differing-inputs obfuscation and applications. *IACR Cryptology ePrint Archive*, 2013, 2013.
5. P. Ananth and V. Vaikuntanathan. Optimal bounded-collusion secure functional encryption. *IACR Cryptology ePrint Archive*, 2019:314, 2019.
6. F. Baldimtsi, A. Kiayias, and K. Samari. Watermarking public-key cryptographic functionalities and implementations. In *ISC*, 2017.
7. A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, 2012.
8. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.
9. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2), 2012.

10. M. Bellare and G. Fuchsbauer. Policy-based signatures. In *PKC*, 2014.
11. O. Billet and D. H. Phan. Efficient traitor tracing from collusion secure codes. In *ICITS*, 2008.
12. D. Boneh and M. K. Franklin. An efficient public key traitor tracing scheme. In *CRYPTO*, 1999.
13. D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, 2001.
14. D. Boneh, K. Lewi, and D. J. Wu. Constraining pseudorandom functions privately. In *PKC*, 2017.
15. D. Boneh and M. Naor. Traitor tracing with constant size ciphertext. In *ACM CCS*, 2008.
16. D. Boneh, A. Sahai, and B. Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, 2006.
17. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC*, 2011.
18. D. Boneh and B. Waters. A fully collusion resistant broadcast, trace, and revoke system. In *ACM CCS*, 2006.
19. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, 2007.
20. D. Boneh and M. Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.
21. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.
22. Z. Brakerski, N. Chandran, V. Goyal, A. Jain, A. Sahai, and G. Segev. Hierarchical functional encryption. In *ITCS*, 2017.
23. Y. Chen, V. Vaikuntanathan, B. Waters, H. Wee, and D. Wichs. Traitor-tracing from LWE made simple and attribute-based. In *TCC*, 2018.
24. B. Chor, A. Fiat, and M. Naor. Tracing traitors. In *CRYPTO*, 1994.
25. B. Chor, A. Fiat, M. Naor, and B. Pinkas. Tracing traitors. *IEEE Trans. Information Theory*, 46(3), 2000.
26. C. C. Cocks. An identity based encryption scheme based on quadratic residues. In *IMA*, 2001.
27. A. Cohen, J. Holmgren, R. Nishimaki, V. Vaikuntanathan, and D. Wichs. Watermarking cryptographic capabilities. In *STOC*, 2016.
28. N. Fazio, A. Nicolosi, and D. H. Phan. Traitor tracing with optimal transmission rate. In *ISC*, 2007.
29. D. M. Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *EUROCRYPT*, 2010.
30. S. Garg, A. Kumarasubramanian, A. Sahai, and B. Waters. Building efficient fully collusion-resilient traitor tracing and revocation schemes. In *ACM CCS*, 2010.
31. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
32. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, 2012.
33. R. Goyal, V. Koppula, and B. Waters. Collusion resistant traitor tracing from learning with errors. In *STOC*, 2018.
34. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, 2006.
35. N. Hopper, D. Molnar, and D. A. Wagner. From weak to strong watermarking. In *TCC*, 2007.

36. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, 2008.
37. S. Kim and D. J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In *CRYPTO*, 2017.
38. S. Kim and D. J. Wu. Watermarking PRFs from lattices: Stronger security via extractable PRFs. In *CRYPTO*, 2019.
39. K. Kurosawa and Y. Desmedt. Optimum traitor tracing and asymmetric schemes. In *EUROCRYPT*, 1998.
40. S. Ling, D. H. Phan, D. Stehlé, and R. Steinfeld. Hardness of k-LWE and applications in traitor tracing. In *CRYPTO*, 2014.
41. Z. Liu, Z. Cao, and D. S. Wong. Blackbox traceable CP-ABE: how to catch people leaking their keys by selling decryption devices on eBay. In *ACM CCS*, 2013.
42. Z. Liu and D. S. Wong. Practical ciphertext-policy attribute-based encryption: Traitor tracing, revocation, and large universe. In *ACNS*, 2015.
43. H. K. Maji, M. Prabhakaran, and M. Rosulek. Attribute-based signatures. In *CT-RSA*, 2011.
44. D. Naccache, A. Shamir, and J. P. Stern. How to copyright a function? In *PKC*, 1999.
45. M. Naor and O. Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *FOCS*, 1995.
46. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, pages 458–467, 1997.
47. M. Naor, O. Reingold, and A. Rosen. Pseudo-random functions and factoring (extended abstract). In *STOC*, pages 11–20, 2000.
48. R. Nishimaki. How to watermark cryptographic functions. In *EUROCRYPT*, 2013.
49. R. Nishimaki, D. Wichs, and M. Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In *EUROCRYPT*, 2016.
50. A. O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010, 2010.
51. D. H. Phan, R. Safavi-Naini, and D. Tonien. Generic construction of hybrid public key traitor tracing with full-public-traceability. In *ICALP*, 2006.
52. W. Quach, D. Wichs, and G. Zirdelis. Watermarking PRFs under standard assumptions: Public marking and security with extraction queries. In *TCC*, 2018.
53. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.
54. A. Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, 1984.
55. E. Shi, J. Bethencourt, H. T. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE S&P*, 2007.
56. T. Sirvent. Traitor tracing scheme with constant ciphertext rate against powerful pirates. *IACR Cryptology ePrint Archive*, 2006, 2006.
57. J. Staddon, D. R. Stinson, and R. Wei. Combinatorial properties of frameproof and traceability codes. *IEEE Trans. Information Theory*, 47(3), 2001.
58. D. R. Stinson and R. Wei. Combinatorial properties and constructions of traceability schemes and frameproof codes. *SIAM J. Discrete Math.*, 11(1), 1998.
59. R. Tsabary. An equivalence between attribute-based signatures and homomorphic signatures, and new constructions for both. In *TCC*, 2017.
60. M. Yoshida and T. Fujiwara. Toward digital watermarking for cryptographic data. *IEICE Transactions*, 94-A(1), 2011.