Watermarking Cryptographic Programs Against Arbitrary Removal Strategies *

Ryo Nishimaki† Daniel Wichs‡

December 6, 2015

Abstract

A watermarking scheme for programs embeds some information called a mark into a program while preserving its functionality. No adversary can remove the mark without damaging the functionality of the program. In this work, we study the problem of watermarking various cryptographic programs such as pseudorandom function (PRF) evaluation, decryption, and signing. For example, given a PRF key K, we create a marked program $\widetilde{C}$ that evaluates the PRF $F(K, \cdot)$. An adversary that gets $\widetilde{C}$ cannot come up with *any* program C^* in which the mark is removed but which still evaluates the PRF correctly on even a small fraction of the inputs.

The work of Barak et al. (CRYPTO'01 and J.ACM, 59(2)) shows that, assuming indistinguishability obfuscation (iO), such watermarking is *impossible* if the marked program $\widetilde{C}$ evaluates the original program with perfect correctness. In this work we show that, assuming iO, such watermarking is *possible* if the marked program $\widetilde{C}$ is allowed to err with even a negligible probability, which would be undetectable to the user. We construct such a watermarking scheme with a secret-marking key used to embed marks in programs, and a public-detection key that allows anyone to detect marks in programs. For our security definition, we assume that the adversary can get oracle access to the marking functionality.

We emphasize that our security notion of watermark non-removability considers arbitrary adversarial strategies to modify the marked program – for example, an adversary could obfuscate the marked program and this should not remove the mark. This is in contrast to the prior works, such as that of Nishimaki (EUROCRYPT '13), which only consider restricted removal strategies that preserve the original structure of the marked program (e.g., as a vector of group elements), but do not provide security against arbitrary strategies.

1 Introduction

Digital watermarking enables us to embed some special information called a *mark* into digital objects such as images, movies, music files, or software. We often call such objects *marked*. There are two basic requirements for watermarking. The first is that a marked object should not be significantly different from the original object. The second is that malicious entities should not be able to remove embedded marks without somehow "destroying" the object (e.g., modify an image beyond recognition).

There are many works on watermarking perceptual objects such as images, movies, music files, etc. Most of them do not give a rigorous theoretical treatment and their constructions are heuristic and ad-hoc. The work of [HMW07] proposes general and rigorous definitions for such schemes, and gives an in-depth exploration of the connections between such definitions, but does not provide any actual constructions.

*The results of this paper were subsumed by [CHN+15].

†NTT Secure Platform Laboratories, nishimaki.ryo@lab.ntt.co.jp. This work was done while the author is visiting Northeastern University.

‡Northeastern University, wichs@ccs.neu.edu. Research supported by NSF grants CNS-1347350, CNS-1314722, CNS-1413964.

Watermarking Cryptographic Programs. In this work, we focus on watermarking *programs*, and specifically ones that are cryptographic in nature, such as programs evaluating a pseudorandom function (PRF) or implementing a signing or a decryption procedure. For concreteness, let us focus on the case of PRFs.

We define a watermarking scheme for a PRF family $\{F(K,\cdot)\}$ to consist of procedures $\mathsf{Mark}(mk,\cdot)$ and $\mathsf{Detect}(dk,\cdot)$ with a secret-marking key mk and a public-detection key dk. Given a PRF key K, the marking procedure $\widetilde{C} \leftarrow \mathsf{Mark}(mk,K)$ creates a marked PRF circuit $\widetilde{C}$ that evaluates $F(K,\cdot)$. Although we will see that we cannot achieve *perfect correctness* where $\widetilde{C}(x) = F(K,x)$ for all inputs x, we will be able to achieve *statistical correctness* where we allow a negligible error probability. The detection procedure $\mathsf{Detect}(dk,C')$ either outputs that a circuit is marked or $\mathsf{unmarked}$.

For security, we consider a game where we choose a random PRF key K and give the adversary the marked circuit $\widetilde{C} \leftarrow \mathsf{Mark}(mk,K)$. Intuitively, we require that the adversary cannot come up with *any* program that correctly evaluates the PRF and removes the mark. More precisely, the adversary produces a circuit C^* and we insist that either: (1) the circuit C^* does not even approximately compute $F(K,\cdot)$, meaning that $C^*(x) = F(K,x)$ on at most an ε-fraction of the inputs x, or (2) the detection procedure correctly detects that the circuit is marked by outputting $\mathsf{marked} \leftarrow \mathsf{Detect}(dk,C^*)$. The parameter ε, called the "approximation factor" and we can set it to some small constant such as 1% or even to any $1/\mathrm{poly}$ fraction. During the attack, the adversary is also given the public-detection key dk and access to the marking oracle $\mathsf{Mark}(mk,\cdot)$ that he can query on arbitrary PRF keys of his choice.

Variants. We call the above type of watermarking "mark-embedding" to denote that it only distinguishes between marked and unmarked program. We also consider "message-embedding" watermarking where the marking procedure can be used to embed an arbitrary message into the program and the detection procedure should recover the message.

We can also consider a relaxation of the above definition to the symmetric-key setting, where there is a single secret watermarking key wk used by the marking and detection procedures, and weaker security definitions depending on whether the adversary has access to the marking and/or detection oracles (a detection oracle $\mathsf{Detect}(wk,\cdot)$ allows an adversary to test whether a circuit is marked or unmarked – in the public-key setting this is subsumed by giving the adversary the public-detection key dk).

Motivation. To highlight the usefulness of watermarking cryptographic functions, we describe two applications as examples. However, we emphasize that the concept should have broader applicability beyond these examples.

Consider an automobile manufacturer that wants to put electronic locks on its cars; the car contains a PRF key K and can only be opened by running an identification protocol where it chooses a random input x and the user must respond with $F(K,x)$. When a car is sold to a new owner, the owner is given a software key (e.g., a smart-phone application) consisting of marked program $\widetilde{C}$ that evaluates the PRF $F(K,\cdot)$ and is used to open the car. The mark can embed some identifying information such as the owner's name and address. Even if the software key is stolen, the thief cannot create some new software that would open the car while removing information about the original owner.

As another example, we can create a traitor-tracing scheme that preserves anonymity without having a central registry of users. In this scenario there is a content provider that streams encrypted copyrighted content to users. When users register, they provide some identifying information and they get a marked decryption key with this information embedded, but the identifying information does not need to be stored in any central registry in the long term.[1] The content provider only stores an anonymous encryption key for

[1]Alternately, we can implement this using a two-party computation protocol, where the user has her identifying information as an input (perhaps signed by a third party) and gets the marked key as an output without the content provider learning anything about the user at any point in time.

each user. If any user posts a valid decryption key in public, the identifying information about that user can be recovered from the key.[2]

Impossibility? The work of Barak et al. [BGI+01, BGI+12] initiated the first theoretical study of program watermarking. They propose a game-based definition which is similar to the weakest definition we consider in this work (in the symmetric-key setting with no marking/detection oracles given to the adversary) but require perfect correctness. Unfortunately, they show that this definition is unachievable assuming the existence of indistinguishability obfuscation.

The main intuition behind the negative result is to consider an attacker that takes a marked program and applies indistinguishability obfuscation (iO) to it. If the marked program implements the original program with *perfect correctness* then the result of applying iO to it should be indistinguishable from that of applying iO to the original program. Since the latter is unlikely to be marked, the same should apply to the former. Therefore, this presents a valid attack against watermarking in general.

Barak et al. note that the above attack crucially relies on the *perfect* (rather than merely *statistical*) correctness of the marked program, meaning that it correctly evaluates the original program on every input. They mention that otherwise "it seems that obfuscators would be useful in constructing watermarking schemes, because a watermark could be embedded by changing the value of the function at a random input, after which an obfuscator is used to hide this change." This idea was not explored further in [BGI+01, BGI+12] and it is far from clear if a restricted notion of obfuscation such as iO would be sufficient and what type of watermarking/security can be achieved with this approach. Indeed, this idea serves as the starting point of our work.

Other Related Work. We mention that there are several other works [NSS99, YF11, Nis13] that propose concrete schemes for watermarking cryptographic functions, under several different definitions and assumptions. For example, the work of Nishimaki [Nis13] gives formal definitions and provably secure constructions for watermarking cryptographic functions (such as trapdoor functions). The main aspect that sets our work apart from these works is that they only consider restricted attacks which attempt to remove a watermark by outputting a new program which has some specific format (rather than an arbitrary program). For example, in the work of Nishimaki [Nis13], cryptographic keys (both marked and unmarked ones) are represented as a vector of group elements, and a valid attack is required to output a key of the same type. In particular, for all of the schemes proposed in these works, the watermark can be removed via the attack described in [BGI+01, BGI+12] where an adversary uses iO to obfuscate the marked program so as to preserve its functionality but completely change its structure.

1.1 Our Results and Techniques

We start by giving new formal definitions of program watermarking, along the lines of what we described earlier. To avoid the [BGI+01, BGI+12] impossibility results, our definition allows for statistical rather than perfect correctness.

As our main technical contribution, we show how to watermark a family of PRFs. Our scheme has a public-key detection procedure and achieves security in the presence of a marking oracle. We get a mark-embedding scheme (no message) that allows for any $\varepsilon = 1/\mathrm{poly}$ approximation factor and a message-embedding scheme that allows for approximation factors $\varepsilon = 1/\sqrt{2} + 1/\mathrm{poly}$. In the case of message-

[2]Note that, in this scenario, we assume that the content is encrypted separately under each user's key, which corresponds to a naive traitor-tracing scheme. Unfortunately, our watermarking definitions/constructions do not guarantee any security if the same key is marked with different messages and therefore each user would need to get a different key. The novelty here is that the identifying information about users is embedded in their keys rather than stored in a central registry. It would be interesting to combine this idea with more advanced traitor-tracing schemes where content doesn't need to be encrypted separately.

embedding constructions, we show that there is an inherent lower bound of $\varepsilon = 1/2$. Both schemes rely on (sub-exponentially secure) indistinguishability obfuscation (iO).

For intuition, we also first construct a simple "basic scheme" which is a mark-embedding scheme (no message) in the symmetric-key setting (both the marking and detection keys are secret) and where the adversary does not get access to the marking or detection oracles. This scheme highlights the high-level ideas but removes much of the complexity needed for our full construction.

Once we have shown how to watermark a family of PRFs, we can easily extend this to watermarking other cryptographic primitives such as the decryption procedure of a public-key encryption or the signing procedure of a signature scheme. To do so, we rely on recent (obfuscation-based) constructions of public-key encryption and signatures where the decryption/signing procedures are simply PRF evaluation [SW14].

Our techniques. On a high level, to watermark a PRF $F(K, \cdot)$, we create a circuit C that evaluates the PRF correctly on almost all inputs x, except for some special set of "marked-points" (of negligible density) on which the program outputs incorrect values that can be detected. We obfuscate the circuit C to create a marked circuit $\widetilde{C}$ which "hides" the marked-points. The detection function can test an arbitrary circuit C' by evaluating it on some subset of the marked-points to determine if it is marked on unmarked.

For our basic scheme, we can implement the above idea by choosing a polynomial-sized set of ℓ marked-points $x_1, \ldots, x_\ell$ and corresponding outputs $y_1, \ldots, y_\ell$ uniformly at random. These values form a watermarking key wk which is needed to both embed marked and detect marks. The detection function tests if on *at least one* of the ℓ points x_i the program evaluates to y_i, and if so it determines that the program is "marked" else "unmarked". We show that this implements "mark-embedding" watermarking (no message) with security in the symmetric-key setting against an attacker that doesn't get access to the marking or detection oracles. We require that the PRF is a *puncturable PRF* [BW13, BGI14, KPTZ13, SW14] and the obfuscation scheme satisfies iO security. The proof follows in two steps. Firstly, using a careful series of hybrids, we show that an adversary that gets the marked program cannot differentiate between the marked-points x_i and uniformly random points in the domain. This relies on the puncturing proof technique by Sahai and Waters [SW14]. Secondly, we show that this implies that the probability of an adversary being able to come up with a program which is correct on even an ε-fraction of inputs, but is not detected (differs from the marked program on all of the ℓ marked-points) is negligible as long as $\ell = O(\lambda/\varepsilon)$ where λ is the security parameter. Therefore, we can set the approximation factor to any $\varepsilon = 1/\mathrm{poly}(\lambda)$ by choosing a correspondingly larger ℓ.

Unfortunately, the above scheme already becomes insecure if the adversary has access to a marking oracle *or* a detection oracle, let alone if we make the key wk public. Fixing this in our full scheme is a significant challenge and requires us to introduce a more complex variant of the above ideas where: (1) The set of marked-points is different for each marked circuit and depends on the key K being marked, while still allowing the detection procedure to find marked-points without knowing K, (2) The number of marked-points is super-polynomial and the detection procedure tests the circuit on a small random subset of ℓ of them; furthermore we obfuscate the main components of the detection procedure and release them as a public-detection key. The proof of security of the full scheme scheme relies on a delicate sequence of super-polynomially many hybrids, which requires us to rely on complexity leveraging, and therefore assume the sub-exponential security of iO and puncturable PRFs.

To embed a message in the marked program, we follow the above approach but ensure that the outputs of the marked circuit on the marked-points encode some information about the message which can be recovered by the detection procedure.

2 Preliminaries

2.1 Notations

For any $n \in \mathbb{N} \setminus \{0\}$, let $[n]$ be the set $\{1, \ldots, n\}$. A bold face lower-case letter denotes a vector like $\boldsymbol{x} = (x_1, \ldots, x_n)$. For two strings x_1 and x_2, $x_1 \| x_2$ denotes a concatenation of x_1 and x_2. For program (or circuit) C, $C[a, b, c, \ldots]$ denotes that C contains the values $a, b, c, \ldots$ "hardwired" in its description. When D is a random variable or distribution, $y \leftarrow D$ denote that y is randomly selected from D according to its distribution. If S is a set, then $x \leftarrow S$ denotes that x is uniformly selected from S. The expression $y := z$ denotes that y is set, defined or substituted by z. We say that function $f : \mathbb{N} \to \mathbb{R}$ is negligible in $\lambda \in \mathbb{N}$ if $f(\lambda) = \lambda^{-\omega(1)}$. Hereafter, we use $f \leq \mathsf{negl}(\lambda)$ to mean that f is negligible in λ.

Let $\mathcal{X} = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ denote two ensembles of random variables indexed by $\lambda \in \mathbb{N}$. We define a distinguish probability between $\mathcal{X}$ and $\mathcal{Y}$ to be $\mu(\lambda)$ if for all PPT algorithm D,

$$|\Pr[D(X_\lambda) = 1] - \Pr[D(Y_\lambda) = 1]| \leq \mu(\lambda)$$

We write $\mathcal{X} \overset{\mathsf{c}}{\approx}_\mu \mathcal{Y}$ to denote this. If μ is negligible, we just write $\mathcal{X} \overset{\mathsf{c}}{\approx} \mathcal{Y}$.

2.2 Definitions

In this section, we review basic notions and definitions used in this paper.

Obfuscation. The notion of indistinguishability obfuscation (iO) was proposed by Barak et. al. [BGI+01, BGI+12] and the first candidate construction was proposed by Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH+13].

Definition 2.1 (Indistinguishability Obfuscation [BGI+12, GGH+13]) A PPT algorithm $i\mathcal{O}$ is an indistinguishability obfuscator (iO) if it satisfies the following two conditions.

Functionality: For all security parameter $\lambda \in \mathbb{N}$, for all circuit C for all input x, it holds that

$$\Pr[C'(x) = C(x) \mid C' \leftarrow i\mathcal{O}(1^\lambda, C)] = 1.$$

Indistinguishability: For all PPT distinguisher $\mathcal{D}$ and all circuit ensembles $C_0 = \{C_\lambda^{(0)}\}_{\lambda \in \mathbb{N}}$ and $C_1 = \{C_\lambda^{(1)}\}_{\lambda \in \mathbb{N}}$ such that $\forall \lambda, x : C_\lambda^{(0)}(x) = C_\lambda^{(1)}(x)$ and $|C_\lambda^{(0)}| = |C_\lambda^{(1)}|$ we have:

$$\left|\Pr[\mathcal{D}(i\mathcal{O}(1^\lambda, C_\lambda^{(0)})) = 1] - \Pr[\mathcal{D}(i\mathcal{O}(1^\lambda, C_\lambda^{(1)})) = 1]\right| \leq \mu(\lambda)$$

where $\mu(\lambda)$ is negligible. If the distinguishing probability $\mu(\lambda)$ is bounded by $2^{-\lambda^\varepsilon}$ for some constant ε, then we say $i\mathcal{O}$ is sub-exponentially secure iO.

For simplicity, we write $i\mathcal{O}(C)$ instead of $i\mathcal{O}(1^\lambda, C)$ when the security parameter λ is clear from context.

Pseudorandom Functions. We review several variants of pseudorandom functions (PRFs).

Definition 2.2 (Pseudorandom Functions) A pseudorandom function $\mathcal{F}$ consists of two PPT algorithms $\mathcal{F} = (\mathsf{Key}, F)$ and a pair of poly-time computable functions $n(\cdot)$ and $m(\cdot)$ that satisfy the following condition. For all PPT adversary $\mathcal{A}$ and $K \leftarrow \mathsf{Key}(1^\lambda)$, it holds

$$\left|\Pr[\mathcal{A}^{F(K,\cdot)} = 1] - \Pr[\mathcal{A}^{\mathcal{R}(\cdot)} = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $F(K,\cdot) : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$ is a deterministic function and $\mathcal{R}$ is chosen uniformly at random from the set of all functions with the same domain/range.

The notion of puncturable pseudorandom function (pPRF) was proposed by Sahai and Waters [SW14].

Definition 2.3 (Puncturable Pseudorandom Functions) A puncturable pseudorandom function (pPRF) $\mathcal{F}$ consists of three (probabilistic) algorithms $\mathcal{F} = (\mathsf{Key}, \mathsf{Puncture}, F)$ and a pair of computable functions $n(\cdot)$ and $m(\cdot)$ that satisfy the following two conditions.

Functionality preserving under puncturing: For all polynomial size set $S \subseteq \{0,1\}^{n(\lambda)}$ and for all $x \in \{0,1\}^{n(\lambda)} \setminus S$, it holds that

$$\Pr[F(K,x) = F(K\{S\},x) \mid K \leftarrow \mathsf{Key}(1^\lambda), K\{S\} := \mathsf{Puncture}(K,S)] = 1.$$

Pseudorandom at punctured points: For all polynomial size set $S = \{x_1,\ldots,x_{k(\lambda)}\} \subseteq \{0,1\}^{n(\lambda)}$ it holds that for all PPT adversary $\mathcal{A}$,

$$\mu(\lambda) := \left|\Pr[\mathcal{A}(K\{S\}, \{F(K,x_i)\}_{i\in[k]}) = 1] - \Pr[\mathcal{A}(K\{S\}, U_{m(\lambda)\cdot|S|}) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $K \leftarrow \mathsf{Key}(1^\lambda)$, $K\{S\} := \mathsf{Puncture}(K,S)$ and U_ℓ denotes the uniform distribution over ℓ bits.

If the distinguish probability $\mu(\lambda)$ is bounded by $2^{-\lambda^\varepsilon}$ for some constant ε, then we say $\mathcal{F}$ is sub-exponentially secure pPRF.

Theorem 2.4 ([GGM86, BW13, BGI14, KPTZ13]) *If one-way functions exits, then for all efficiently computable $n(\cdot)$ and $m(\cdot)$, there exists a pPRF family whose input is an $n(\cdot)$ bit string and output is an $m()$ bit string.*

Prefix Puncturable PRF. We introduce the notion of prefix puncturable PRFs, which is an extension of pPRF. Prefix pPRFs allow us to puncture PRF keys at points that have a fixed prefix such as $x*$ where $*$ means any (fixed length) string.

Definition 2.5 (Prefix Puncturable Pseudorandom Functions) A prefix pPRF $\mathcal{F}$ consists of three (probabilistic) algorithms $\mathcal{F} = (\mathsf{Key}, \mathsf{Puncture}, F)$ and poly-time computable functions $n(\cdot)$, $m(\cdot)$ that satisfy the following two conditions.

Functionality preserving under prefix puncturing: For all strings s and for all $x \in \{0,1\}^{n(\lambda)}$ such that x does not contains s as a prefix it holds that

$$\Pr[F(K,x) = F(K\{s\|*\},x) \mid K \leftarrow \mathsf{Key}(1^\lambda), K\{s\|*\} := \mathsf{Puncture}(K,s\|*)] = 1.$$

Pseudorandom at prefix punctured points: For all PPT adversary $\mathcal{A}$, and for all values s and all polynomial sets of points $S = \{x_1,\ldots,x_{k(\lambda)}\} \subseteq \{0,1\}^{n(\lambda)}$ such that each $x_i \in S$ contains s as a prefix, it holds that

$$\mu(\lambda) := \left|\Pr[\mathcal{A}(K\{s\|*\}, \{F(K,x_i)\}_{i\in[k(\lambda)]}) = 1] - \Pr[\mathcal{A}(K\{s\|*\}, U_{m(\lambda)|S|}) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $K \leftarrow \mathsf{Key}(1^\lambda)$, $K\{s\|*\} := \mathsf{Puncture}(K,s\|*)$, and U_ℓ denotes the uniform distribution over ℓ bits.

If the distinguish probability $\mu(\lambda)$ is bounded by $2^{-\lambda^\varepsilon}$ for some constant ε, then we say $\mathcal{F}$ is sub-exponentially secure prefix pPRF.

We can easily construct prefix pPRFs from the GGM tree-based construction in the same way as puncturable PRFs.

Injective PRFs. We need pPRFs that satisfy the following injectivity condition.

Definition 2.6 (Injective property of pPRF)

$$\Pr_{K \leftarrow \mathsf{Key}(1^\lambda)}[\exists \alpha, K' \neq K \ : \ F(K,\alpha) = F(K',\alpha)] \leq \mathsf{negl}(\lambda)$$

In other words this says that w.o.p. over the choice of K, the value $F(K,\alpha)$ uniquely determines K. The GGM tree-based pPRF from a special *injective* PRG satisfies this requirement. A special PRG is $\mathsf{G}(x) = \mathsf{G}_0(x) \| \mathsf{G}_1(x)$ where $\mathsf{G}_0(x)$ (resp, $\mathsf{G}_1(x)$) is the first (resp, last) half of the bits of $\mathsf{G}(x)$ and G_0 and G_1 are each *injective*, meaning that $\Pr_x[\exists x' \neq x \ : \ \mathsf{G}(x) = \mathsf{G}(x')] \leq \mathsf{negl}(\lambda)$. Moreover each of $|\mathsf{G}_0(x)|, |\mathsf{G}_1(x)|$ is of length at most $|x| + p(\lambda)$ where $p(\lambda)$ is some fixed polynomial. This ensures that, in the GGM tree-based pPRF construction, the size of the output can grow by some additive amount in each level but the overall growth in a level d tree is bounded by $d\mathrm{poly}(\lambda)$. Such PRGs can be constructed from standard assumptions such as DDH or LWE.

Injective pPRF from LWE. For example, using the learning with errors (LWE) assumption, we define $\mathsf{G}_{\boldsymbol{A}} : \mathbb{Z}_q^n \to \mathbb{Z}_p^m$ as $\mathsf{G}_{\boldsymbol{A}}(\boldsymbol{x}) := \left\lfloor \boldsymbol{A}^\mathsf{T} \cdot \boldsymbol{x} \right\rceil_p$ where operator $\lfloor \cdot \rceil_p$ returns the nearest integer (for each coordinate) modulo p. We can set $q := p^2 = 2^{2k}$ for some $k = O(\lambda)$ and $m := 4n + O(\lambda)$. Let $\boldsymbol{A} = \boldsymbol{A}_0 \| \boldsymbol{A}_1$ where $\boldsymbol{A}_0, \boldsymbol{A}_1 \in \mathbb{Z}_q^{n \times m/2}$, then $\mathsf{G}_b(x) = \left\lfloor \boldsymbol{A}_b^\mathsf{T} \boldsymbol{x} \right\rceil$. In this case, each $\mathsf{G}_b(x)$ is injective w.o.p. over the choice of $\boldsymbol{A}$ and it maps $2nk$ bits to $2nk + O(k\lambda)$ bits. See [BPR12] for details about the LWE assumption and proof of security of the above construction.

Injective pPRF from DDH. Alternatively, it may seem that using DDH, we can set $\mathsf{G}_{g_1,g_2}(x) = g_1^x, g_2^x$ where g_1, g_2 are generators of some group $\mathbb{G}$ of primer order p. Unfortunately, the outputs cannot be directly used as PRG inputs in the next level of the tree since they are group elements rather than exponents and we do not know how to extract out two uniform values in $\mathbb{Z}_p$ from them. Nevertheless, this approach can be made to work by defining $\mathsf{G}_{g_1,g_2,g_3,h_0,h_1}(x) = h_0(g_1^x, g_2^x, g_3^x), h_1(g_1^x, g_2^x, g_3^x)$ where h_0, h_1 are universal hash functions that map $\mathbb{G}^3 \to \mathbb{Z}_{p'}$ for some p' such that $\log(p') = \log(p) + O(\lambda)$ and $\log(p') \leq (3/2)\log(p) - \Omega(\lambda)$. This ensures injectivity (we are hashing p balls into p' bins and therefore for any fixed ball there is unlikely to be another ball colliding with it). It also ensures pseudorandomness security by thinking of h_0, h_1 as extractors via the leftover-hash lemma. In the context of the GGM construction we need a hierarchy of DDH groups of order $p_1, p_2, \ldots$ (one for each level) where $\log(p_{i+1}) = \log(p_i) + O(\lambda)$. Therefore the output does not get "too large".

3 Definition of Watermarking

We begin by defining the notion of program watermarking. Our definition is similar to the game-based definition of Barak et al. [BGI$^+$01, BGI$^+$12] with the main difference that: (1) we allow "statistical" rather than perfect correctness, (2) the challenge circuit to be marked is chosen uniformly at random from the circuit family (for example, in the case of PRFs, this corresponds to marking a random PRF key), (2) we strengthen the definition to the public-key detection setting and give the attacker access to the marking oracle.

We start by defining two types of schemes which we will refer to as "mark-embedding" and "message-embedding" watermarking. In the former programs are simply either marked on unmarked while in the latter they can be marked with a (long) message. For conciseness, we define the notions simultaneously by defining the more general notion of "message-embedding" watermarking and thinking of "mark-embedding" watermarking as a special case where the message space $\mathcal{M}$ consists of a single value $\mathcal{M} = \{\mathsf{marked}\}$.

Definition 3.1 (Watermarking Syntax) A *message-embedding watermarking* scheme for a circuit class $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ and a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}$ consists of three algorithms $(\mathsf{Gen}, \mathsf{Mark}, \mathsf{Detect})$.

Key Generation: $(dk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$ takes as input the security parameter and outputs a pair of keys (dk, mk), called detection key and mark key.

Mark: $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})$ takes as input a mark key, a circuit $C \in \mathcal{C}_\lambda$ and a message $\mathsf{msg} \in \mathcal{M}_\lambda$ and outputs a marked circuit $\widetilde{C}$.

Detect: $\mathsf{msg}' \leftarrow \mathsf{Detect}(dk, C')$ takes as input a detection key and and an arbitrary circuit C' (which may not be in the family $\mathcal{C}_\lambda$) and outputs $\mathsf{msg}' \leftarrow \mathsf{Detect}(dk, C')$ where $\mathsf{msg}' \in \mathcal{M} \cup \{\mathsf{unmarked}\}$. We say that detection is *black-box* if it only uses oracle-access to C' to evaluate it on various inputs, but ignores its description otherwise.

We also define *mark-embedding watermarking* which is the same as above but the message space $\mathcal{M}$ consists of a single value $\mathcal{M} = \{\mathsf{marked}\}$. In this case we omit the input msg from the syntax of the marking algorithm and simply write $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C)$. We also define a variant with *symmetric-key detection* where there is a single secret watermarking key $wk = mk = dk$.

Before we define the correctness and security of watermarking, we define the following notion of *approximating* a function.

Definition 3.2 (ε-Approximating a Function) A circuit C is said to ε-approximate a function $f : \{0,1\}^n \to \{0,1\}^*$, denoted by $C \cong_\varepsilon f$, if $\Pr_{x\leftarrow\{0,1\}^n}[C(x) = f(x)] \geq \varepsilon$.

We are now ready to define program watermarking.

Definition 3.3 (Watermarking Security) A watermarking scheme $(\mathsf{Gen}, \mathsf{Mark}, \mathsf{Detect})$ for circuit family $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ and with message space $\mathcal{M} = \{\mathcal{M}_\lambda\}$ is required to satisfy the following properties.

Statistical Correctness: There is a negligible function $\nu(\lambda)$ such that for any circuit $C \in \mathcal{C}_\lambda$, any message $\mathsf{msg} \in \mathcal{M}_\lambda$ and any input x in the domain of C, it holds that

$$\Pr\left[\widetilde{C}(x) = C(x) \;:\; \begin{array}{c}(dk, mk) \leftarrow \mathsf{Gen}(1^\lambda)\\ \widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})\end{array}\right] \geq 1 - \nu(\lambda).$$

Detection Correctness: For every $C \in \mathcal{C}_\lambda$, $\mathsf{msg} \in \mathcal{M}_\lambda$ and $(dk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$:

$$\mathsf{Detect}(dk, \mathsf{Mark}(mk, C, \mathsf{msg})) = \mathsf{msg}.$$

Meaningfulness: For every circuit C (not necessarily in $\mathcal{C}_\lambda$), it holds that

$$\Pr_{(dk,mk)\leftarrow\mathsf{Gen}(1^\lambda)}[\mathsf{Detect}(dk, C) \neq \mathsf{unmarked}] \leq \mathsf{negl}(\lambda).$$

Non-Removability: For every PPT $\mathcal{A}$ we have

$$\mathsf{Adv}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon) := \Pr[\mathsf{Exp}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon) = 1] \leq \mathsf{negl}(\lambda)$$

where ε is a parameter of the scheme called the *approximation factor* and $\mathsf{Exp}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon)$ is the game defined next.

Definition 3.4 (Non-Removability Security Game) The game $\mathsf{Exp}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon)$ is defined as follows.

1. The challenger generates $(dk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$ and gives dk to the adversary $\mathcal{A}$.

2. The adversary has oracle access to the mark oracle $\mathcal{MO}$. If $\mathcal{MO}$ is queried a circuit C_i and message msg_i, then it answers $\mathsf{Mark}(mk, C_i, \mathsf{msg}_i)$.

3. At some point, the adversary sends a challenge message $\mathsf{msg} \in \mathcal{M}_\lambda$ to the challenger. The challenger samples a circuit $C \leftarrow \mathcal{C}_\lambda$ uniformly at random and generates $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})$, and sends $\widetilde{C}$ to the adversary.

4. The adversary can make more queries to $\mathcal{MO}$ and finally outputs a circuit C^*. If it holds $C^* \cong_\varepsilon \widetilde{C} \wedge \mathsf{Detect}(dk, C^*) \neq \mathsf{msg}$ then the experiment outputs 1, otherwise 0. [3]

We can also weaken the above definition in several ways that we mention below.

Variant: Selective-Message Security and Mark-Embedding. Firstly, in the case of message-embedding watermarking, we could consider a *selective-message security* variant where the adversary has to declare the message msg embedded in the challenge circuit at the very beginning of the game, before seeing the detection key or making queries to the marking oracle. In this case the security game consists of the following steps:

1. The adversary sends a challenge message $\mathsf{msg} \in \mathcal{M}_\lambda$ to the challenger.

2. The challenger generates $(dk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$, samples a circuit $C \leftarrow \mathcal{C}_\lambda$ uniformly at random and generates $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})$. It gives $dk, \widetilde{C}$ to the adversary $\mathcal{A}$.

3. The adversary $\mathcal{A}$ gets oracle access to the mark oracle $\mathcal{MO}$ and at some point outputs a circuit C^*. If it holds $C^* \cong_\varepsilon \widetilde{C} \wedge \mathsf{Detect}(dk, C^*) \neq \mathsf{msg}$ then the experiment outputs 1, otherwise 0.

Note that selective-message security implies full security via a guessing argument – the reduction loses a factor of $|\mathcal{M}_\lambda|$ in the advantage and therefore this relies on complexity leveraging when $|\mathcal{M}_\lambda|$ is super-polynomial.

In the case of *mark-embedding* watermarking, there is no challenge message (we always have $\mathsf{msg} = \mathsf{marked}$) and therefore the game becomes as above with step 1 omitted.

Variant: Symmetric-Key Schemes and No-Oracle Security. We can also consider a variant in the symmetric-key setting where there is a single watermarking key $wk = mk = dk$ and the attacker is not given dk in step 1 of the security game. In this case we can consider several variants of the definition depending on whether the adversary has access to the marking and/or detection oracles. The simplest setting we consider is a mark-embedding scheme in the symmetric-key setting with no-oracle security, where the attacker does not get access to either the marking or the detection oracles. In this case, the security game just consists of the following two steps:

1. The challenger chooses $wk \leftarrow \mathsf{Gen}(1^\lambda)$, $C \leftarrow \mathcal{C}_\lambda$ and gives $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C)$ to the adversary.

2. The adversary outputs a circuit C^*. If $C^* \cong_\varepsilon \widetilde{C} \wedge \mathsf{Detect}(wk, C^*) \neq \mathsf{marked}$ then the experiment outputs 1, otherwise 0.

[3] The definition would be equivalent if we had required $C^* \cong_\varepsilon C$ instead of $C^* \cong_\varepsilon \widetilde{C}$, up to a negligible difference in ε, since by statistical correctness we have $C \cong_\delta \widetilde{C}$ for some $\delta = 1 - \mathsf{negl}(\lambda)$.

Note on Statistical Correctness. We mention that, by an averaging argument, the statistical correctness requirement implies that for *any* distribution $\mathcal{D}$ over inputs x, with overwhelming probability over the choice of the marked circuit $\widetilde{C}$, we have $\Pr_{x \leftarrow \mathcal{D}}[\widetilde{C}(x) = C(x)] \geq 1 - \mathsf{negl}(\lambda)$. Therefore, this requirement is more meaningful than simply insisting that $\widetilde{C} \cong_\delta C$ for some $\delta = 1 - \mathsf{negl}(\lambda)$.

Note on the Circuit Family. We also note that the marking procedure expects to get a circuit $C \in \mathcal{C}_\lambda$ from the designated family. For example, in the case of watermarking a PRF family $\{F(K, \cdot)\}$, we will assume that the family $\mathcal{C}_\lambda$ consists of some "canonical representation" of the circuits computing the function $F(K, \cdot)$, and we can simply equate circuits $C \in \mathcal{C}_\lambda$ with PRF keys K; we will write $\mathsf{Mark}(mk, K, \mathsf{msg})$ in place of $\mathsf{Mark}(mk, C, \mathsf{msg})$ where the C is the canonical circuit computing $F(K, \cdot)$. In the non-removability security game, the fact that the challenge circuit $C \leftarrow \mathcal{C}_\lambda$ is chosen randomly corresponds to marking a random PRF key. However, we emphasize that the detection procedure $\mathsf{Detect}(dk, C')$ must work on arbitrary circuits C', which may not be from the designated family, and the adversary wins if he can create *any* circuit C' which is correct and on which detection fails.

Impossibility of Message-Embedding for $\varepsilon \leq \frac{1}{2}$. Unfortunately, message-embedding watermarking is impossible in a setting where the adversary can make a single non-adaptive query to the *marking oracle* (even without a detection oracle) when $\varepsilon \leq \frac{1}{2}$. This holds even if the message space is just $\mathcal{M} = \{0, 1\}$ and even in the symmetric-key setting when the adversary has no access to a detection oracle. The adversary chooses a random bit $b \leftarrow \{0, 1\}$ at the beginning of the game. It makes a single non-adaptive call to the marking oracle on a random circuit $C \leftarrow \mathcal{C}_\lambda$ with the message bit $1 - b$ and gets back some marked circuit $\widetilde{C}_{1-b}$. It then chooses the challenge message b and gets back some challenge marked circuit $\widetilde{C}_b$. It creates a circuit C^* such that, on inputs x that start with the bit b, $C^*(x) = \widetilde{C}_b(x)$ and on inputs x that start with the bit $1 - b$, $C^*(x) = \widetilde{C}_{1-b}(x)$. This ensures that $C^* \cong_\varepsilon \widetilde{C}_b$ for $\varepsilon = \frac{1}{2}$ meaning that C^* is an ε-approximation of the challenge marked circuit. But since the way that C^* is created is symmetric with respect to b and $1 - b$, we know that $\Pr[\mathsf{Detect}(dk, C^*) = b] \leq \frac{1}{2}$.

4 Watermarking PRFs

In this section, we construct schemes for watermarking any puncturable PRF family. To build up intuition, we start with a simple "basic construction" which is a mark-embedding symmetric-key watermarking scheme with "no-oracle security" (the attacker does not get access to either the marking or the detection oracles). We then explain the challenges that we need to overcome to get full security in the public-key detection setting, and give our full construction which achieves this.

For all of the schemes, let $\mathcal{F} = (\mathsf{Key}, \mathsf{Puncture}, F)$ be some puncturable PRF (pPRF) family where, for $K \leftarrow \mathsf{Key}(1^\lambda)$ we have $F(K, \cdot) : \mathsf{D}_\lambda \to \mathsf{R}_\lambda$ with $\mathsf{D}_\lambda = \{0, 1\}^{n(\lambda)}$, and $\mathsf{R}_\lambda = \{0, 1\}^{m(\lambda)}$ for some super-logarithmic $n(\lambda), m(\lambda) = \omega(\log \lambda)$. We often drop λ from D_λ and R_λ. We construct a watermarking scheme for *PRF evaluation* of $\mathcal{F}$. We identify the PRF evaluation circuits computing the function $F(K, \cdot)$ with their key K and assume (without loss of generality) that the marking procedure just takes K as an input.

4.1 Basic Construction: Symmetric-Key Watermarking, No Oracles

In this section, we start by proposing the most basic of our watermarking schemes, which is a mark-embedding construction in the symmetric-key setting and achieves security against an adversary with no oracle access (weakest security). We call this "the basic construction" throughout the paper, and the intuition for it was given in the introduction.

Construction. Our watermarking scheme with "approximation factor" $\varepsilon = \varepsilon(\lambda)$ is defined as follows.

$\mathsf{Gen}(1^\lambda)$**:** Set $\ell := \lambda/\varepsilon$. Choose an $\boldsymbol{x} = (x_1, \ldots, x_\ell) \leftarrow \mathsf{D}^\ell$ and a vector $\boldsymbol{y} = (y_1, \ldots, y_\ell) \leftarrow \mathsf{R}^\ell$ uniformly at random and set $wk := (\boldsymbol{x}, \boldsymbol{y})$. We can assume that $x_1, \ldots, x_\ell$ are distinct.

$\mathsf{Mark}(wk, K)$**:** Output $i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}])$. The program $C[K, \boldsymbol{x}, \boldsymbol{y}]$ is described in Figure 1. The size of the program is appropriately padded to be the maximum size of all modified programs, which will appear in the security proof.

$\mathsf{Detect}(wk, C')$**:** If there exists $i \in [\ell]$ such that $y_i = C'(x_i)$, then output marked, else unmarked.

Constants: $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in \mathsf{D}^\ell$, $\boldsymbol{y} = (y_1, \ldots, y_\ell) \in \mathsf{R}^\ell$, pPRF key K.

Inputs: $x \in \mathsf{D}$

Code: If there exists i such that $x = x_i$, then output y_i. Else, output $F(K, x)$.

Figure 1: Program $C[K, \boldsymbol{x}, \boldsymbol{y}]$

We call the value $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ the *marked-points* since marked programs outputs special values from $\boldsymbol{y}$ for these inputs. We can easily verify that the basic construction satisfies statistical correctness, detection correctness, and meaningfulness properties in Definition 3.3.

Security Proofs for the Basic Construction. We prove the security, non-removability, of the basic construction.

Theorem 4.1 *The basic construction satisfies symmetric-key non-removable security with no oracles (where the adversary in the security game of non-removability has oracle access to neither the mark nor detect oracle) if $\mathcal{F}$ is a pPRF and $i\mathcal{O}$ is an indistinguishability obfuscator.*

We prove Theorem 4.1 by using the following two lemmas.

Lemma 4.2 *If $\mathcal{F}$ is a pPRF and $i\mathcal{O}$ is an indistinguishability obfuscator, then the following holds. For $K \leftarrow \mathsf{Key}(1^\lambda)$ and uniformly random and independent $\boldsymbol{x}, \boldsymbol{x}' \leftarrow \mathsf{D}^\ell$ and $\boldsymbol{y} \leftarrow \mathsf{R}^\ell$, it holds that*

$$(\boldsymbol{x}, i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}])) \stackrel{\mathsf{c}}{\approx} (\boldsymbol{x}', i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}])).$$

Lemma 4.3 *If a PPT adversary $\mathcal{A}$ that has no oracle access breaks the non-removability security with probability $\delta(\lambda)$, then we can construct a PPT distinguisher $\mathcal{D}$ that can distinguish $(\boldsymbol{x}, i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}]))$ and $(\boldsymbol{x}', i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}]))$ with probability $\delta(\lambda) - \mathsf{negl}(\lambda)$.*

Lemma 4.2 and 4.3 immediately imply Theorem 4.1.

Proof of Lemma 4.3. Given an input $(\boldsymbol{x}^* = (x_1^*, \ldots, x_\ell^*), \widetilde{C} = i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}]))$, we construct a distinguisher $\mathcal{D}$ that uses the adversary $\mathcal{A}$ to distinguish between $\boldsymbol{x}^* = \boldsymbol{x}$ and $\boldsymbol{x}^* = \boldsymbol{x}'$ for uniformly random and independent $\boldsymbol{x}'$. The distinguisher $\mathcal{D}$ gives $\widetilde{C}$ to the adversary $\mathcal{A}$ as a challenge marked program. Eventually, $\mathcal{A}$ outputs a program C^* and $\mathcal{D}$ verifies whether there exits i such that $C^*(x_i^*) = \widetilde{C}(x_i^*)$ or not. Formally, $\mathcal{D}$ proceeds as follows.

1. $\mathcal{D}$ gets $(\boldsymbol{x}^*, \widetilde{C})$ and gives $\widetilde{C}$ to $\mathcal{A}$.

2. $\mathcal{D}$ receives C^* from $\mathcal{A}$ and runs $\mathsf{Test}(C^*, \widetilde{C})$ described in Figure 2 to test if C^* approximates $\widetilde{C}$. If Test outputs 0, then $\mathcal{D}$ outputs 0. Otherwise, goes to the next step.

3. If for all $i \in [\ell]$, $\widetilde{C}(x_i^*) \neq C^*(x_i^*)$, then $\mathcal{D}$ outputs 1, otherwise 0.

Inputs: Two programs C_0 and C_1.

Parameters: $0 < \varepsilon < 1$.

Set $\mathsf{cnt} := 0$ and $R := 8\lambda(1/\varepsilon)^2$. For $i = 1, \ldots, R$, do

1. Choose $z_i \leftarrow \mathsf{D}$
2. If $C_0(z_i) = C_1(z_i)$ then set $\mathsf{cnt} := \mathsf{cnt} + 1$

Let $\widehat{\varepsilon} := \mathsf{cnt}/R \in [0, 1]$ be the fraction of trials in which $C_0(z_i) = C_1(z_i)$.

If $\widehat{\varepsilon} < \frac{3}{4}\varepsilon$, then output 0, else 1.

Figure 2: Test algorithm Test for approximation of programs

Analysis of Test. Let Z_i be a random variable such that $Z_i = 1$ if and only if $C_0(z_i) = C_1(z_i)$ for $z_i \leftarrow \mathsf{D}$. Note that Z_i are independent and identically distributed.

If $C_0 \cong_\varepsilon C_1$ then $\Pr[Z_i = 1] \geq \varepsilon$ and the Chernoff bound implies $\Pr[\widehat{\varepsilon} < \frac{3}{4}\varepsilon] \leq \exp(-\frac{1}{8}R\varepsilon^2) \leq \exp(-\lambda)$. This means that the test outputs 1 with overwhelming probability.

If it is *not* the case that $C_0 \cong_{\varepsilon/2} C_1$ then $\Pr[Z_i = 1] < \varepsilon/2$ and the Chernoff bound implies $\Pr[\widehat{\varepsilon} \geq \frac{3}{4}\varepsilon] \leq \exp(-\frac{1}{8}R\varepsilon^2) \leq \exp(-\lambda)$. This means that the test output 0 with overwhelming probability.

Analysis of $\mathcal{D}$. Next, we analyze how $\mathcal{D}$ works.

If the given distribution is the real one $(\boldsymbol{x}, i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}]))$ (i.e., $\boldsymbol{x}^* = \boldsymbol{x}$), then $\mathcal{A}$ wins with the non-removability game with probability $\delta(\lambda)$, meaning that it outputs an un-marked program C^* that ε-approximates $\widetilde{C}$. If this happens, then Test passes (outputs 1) with overwhelming probability $1 - \mathsf{negl}(\lambda)$ and, for all $i \in [\ell]$, $C^*(x_i^*) \neq \widetilde{C}(x_i^*)$. Therefore, in this case, $\mathcal{D}$ outputs 1 with probability at least $\delta(\lambda) - \mathsf{negl}(\lambda)$.

If the given distribution is random one $(\boldsymbol{x}', i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}]))$ (i.e., $\boldsymbol{x}^* = \boldsymbol{x}'$), then $\boldsymbol{x}$ and $\boldsymbol{x}'$ are uniformly random and independently distributed. In this case, we have no guarantees about the program C^* output by $\mathcal{A}$ and we analyze two cases. One case is that C^* does not $\varepsilon/2$-approximate $\widetilde{C}$. In this case, the probability that Test passes (outputs 1) is negligible. The other case is that C^* does $\varepsilon/2$-approximate $\widetilde{C}$. In this case, since $\boldsymbol{x}^* = (x_1^*, \ldots, x_\ell^*)$ are ℓ random values independent of $\widetilde{C}, C^*$, it holds that

$$\Pr[\,\forall i\, \widetilde{C}(x_i^*) \neq C^*(x_i^*)] = \prod_i \Pr[\widetilde{C}(x_i^*) \neq C^*(x_i^*)] \leq (1 - \varepsilon/2)^\ell \leq \mathsf{negl}(\lambda).$$

Therefore, if the distribution is a random the probability that $\mathcal{D}$ outputs 1 is at most $\mathsf{negl}(\lambda)$.

This means that $\mathcal{D}$ has distinguishing advantage $\delta - \mathsf{negl}(\lambda)$. □

Proof of Lemma 4.2. We define a sequence of hybrid games to prove the lemma.

Hyb^0**:** This is the real game. The distribution is as follows. For $K \leftarrow \mathsf{Key}(1^\lambda)$, $\boldsymbol{x} \leftarrow \mathsf{D}^\ell$ and $\boldsymbol{y} \leftarrow \mathsf{R}^\ell$,

$$(\boldsymbol{x}, i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}])).$$

Hyb^1**:** In this game, we replace the pPRF key with a punctured one. The distribution is as follows. For $K \leftarrow \mathsf{Key}(1^\lambda)$, $\boldsymbol{x} \leftarrow \mathsf{D}^\ell$, $\boldsymbol{y} \leftarrow \mathsf{R}^\ell$, $K\{\boldsymbol{x}\} := \mathsf{Puncture}(K, \boldsymbol{x})$, program $C[K\{\boldsymbol{x}\}, \boldsymbol{x}, \boldsymbol{y}]$ in Figure 3,

$$(\boldsymbol{x}, i\mathcal{O}(C[K\{\boldsymbol{x}\}, \boldsymbol{x}, \boldsymbol{y}])).$$

Constants: $\boldsymbol{x} \in \mathsf{D}^\ell$, $\boldsymbol{y} \in \mathsf{R}^\ell$, pPRF key $K\{\boldsymbol{x}\}$.

Inputs: $x \in \mathsf{D}$

Code: If there exists i such that $x = x_i$, then output y_i. Else, output $F(K\{\boldsymbol{x}\}, x)$.

Figure 3: Program $C[K\{\boldsymbol{x}\}, \boldsymbol{x}, \boldsymbol{y}]$

It holds that $\mathsf{Hyb}^0 \stackrel{\mathsf{c}}{\approx} \mathsf{Hyb}^1$ if $i\mathcal{O}$ is an indistinguishability obfuscation. In Hyb^1, the PRF key is punctured, but program $C[K\{\boldsymbol{x}\}, \boldsymbol{x}, \boldsymbol{y}]$ does not compute F at points $\boldsymbol{x}$ due to the if-branch code. Thus, for all input x, the behavior of two programs is exactly the same and this is indistinguishable by $i\mathcal{O}$.

Hyb^2**:** In this game, we replace $\boldsymbol{y}$ with outputs of pPRF. The distribution is as follows. For $K \leftarrow \mathsf{Key}(1^\lambda)$, $\boldsymbol{x} \leftarrow \mathsf{D}^\ell$, $y_i := F(K, x_i)$ for all $i \in [\ell]$, $K\{\boldsymbol{x}\} := \mathsf{Puncture}(K, \boldsymbol{x})$, and program remains $C[K\{\boldsymbol{x}\}, \boldsymbol{x}, \boldsymbol{y}]$ shown in Figure 3.

It holds that $\mathsf{Hyb}^1 \stackrel{\mathsf{c}}{\approx} \mathsf{Hyb}^2$ by the security of the pPRF $\mathcal{F}$. The only difference is the distribution of $\boldsymbol{y}$, but this is indistinguishable due to the pseudorandomness at punctured points of pPRF.

Hyb^3**:** We replace the punctured PRF key with a standard one and erase the "if-branch" in the code of the obfuscated program. The distribution is as follows. For $\boldsymbol{x} \leftarrow \mathsf{D}^\ell$, $K \leftarrow \mathsf{Key}(1^\lambda)$, and program $C[K]$ in Figure 4,

$$(\boldsymbol{x}, i\mathcal{O}(C[K])).$$

It holds that $\mathsf{Hyb}^2 \stackrel{\mathsf{c}}{\approx} \mathsf{Hyb}^3$ if $i\mathcal{O}$ is an indistinguishability obfuscation. In these two hybrid games, the

Constants: pPRF key K.

Inputs: $x \in \mathsf{D}$

Code: Output $F(K, x)$.

Figure 4: Program $C[K]$

output of two programs is $F(K, x)$ for *any* input x. Thus, the behavior of two programs are exactly the same due to the functionality preserving property under puncturing.

Note that in Hyb^3 the vector $\boldsymbol{x}$ does not appear in the obfuscated program and therefore by "renaming" this vector we can write the distribution as $\boldsymbol{x}' \leftarrow \mathsf{D}^\ell$, $K \leftarrow \mathsf{Key}(1^\lambda)$, and program $C[K]$ in Figure 4,

$$(\boldsymbol{x}', i\mathcal{O}(C[K])).$$

Hyb^4**:** We change the PRF key again, that is, we use $K\{\boldsymbol{x}\} := \mathsf{Puncture}(K, \boldsymbol{x})$ and set $y_i := F(K, x_i)$ for all $i \in [\ell]$. The distribution is as follows. For $K \leftarrow \mathsf{Key}(1^\lambda)$, $\boldsymbol{x}, \boldsymbol{x}' \leftarrow \mathsf{D}^\ell$, $y_i := F(K, x_i)$ for all $i \in [\ell]$, $K\{\boldsymbol{x}\} := \mathsf{Puncture}(K, \boldsymbol{x})$, and program $C[K\{\boldsymbol{x}\}, \boldsymbol{x}, \boldsymbol{y}]$ in Figure 3,

$$(\boldsymbol{x}', i\mathcal{O}(C[K\{\boldsymbol{x}\}, \boldsymbol{x}, \boldsymbol{y}]).$$

It holds that $\mathsf{Hyb}^3 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^4$ by the security of iO (same as $\mathsf{Hyb}^2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^3$).

Hyb^5**:** We choose uniformly random $\boldsymbol{y} \leftarrow \mathsf{R}^\ell$ again. For $K \leftarrow \mathsf{Key}^{1^\lambda}$, $\boldsymbol{x}, \boldsymbol{x}' \leftarrow \mathsf{D}^\ell$, $\boldsymbol{y} \leftarrow \mathsf{R}^\ell$, $K\{\boldsymbol{x}\} := \mathsf{Puncture}(K, \boldsymbol{x})$, and the program in Figure 3,

$$(\boldsymbol{x}', i\mathcal{O}(C[K\{\boldsymbol{x}\}, \boldsymbol{x}, \boldsymbol{y}])).$$

It holds $\mathsf{Hyb}^4 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^5$ by the security of the pPRF (same as $\mathsf{Hyb}^1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^2$).

Hyb^6**:** We replace pPRF key $K\{\boldsymbol{x}\}$ with a normal one. For $K \leftarrow \mathsf{Key}(1^\lambda)$, $\boldsymbol{x}, \boldsymbol{x}' \leftarrow \mathsf{D}^\ell$, $\boldsymbol{y} \leftarrow \mathsf{R}^\ell$, and the program in Figure 1,

$$(\boldsymbol{x}', i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}])).$$

It holds $\mathsf{Hyb}^5 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^6$ by the security of iO (same as $\mathsf{Hyb}^0 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}^1$).

Thus, we obtained $(\boldsymbol{x}, i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}])) \overset{\mathsf{c}}{\approx} (\boldsymbol{x}', i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}]))$. □

4.2 Full Construction: Public-Key Detection and Security with Marking Oracle

We now construct a watermarking scheme with public-key detection and with security in the presence of a marking oracle (see Definition 3.4). We begin with a mark-embedding scheme (no message) that works for any approximation factor $\varepsilon = 1/\mathrm{poly}(\lambda)$ and then show how to convert it into a message-embedding scheme that works for approximation factors $\varepsilon = \frac{1}{2} + 1/\mathrm{poly}(\lambda)$.

Overview and Intuition. There are several challenges that we need to overcome to go form the basic construction to our full construction. Not only does the basic construction crucially rely on the watermarking key wk being secret, it already fails to be secure even if the adversary has access to *either one of* the marking or detection oracles.

Attack using the marking oracle: Assume the adversary makes just one call to the marking oracle with an arbitrary key K' and gets a marked program $\widetilde{C}'$. Then, on the marked-points $x_1, \ldots, x_\ell$ it will happen that $\widetilde{C}'(x_i) \neq F(K', x_i)$, while for all other points equality holds. Therefore, the adversary can easily distinguish the marked-points from random points. In particular this leads to a concrete attack. Once the adversary gets a challenge marked program $\widetilde{C}$ for some random and unknown PRF key K, the adversary can create a program C^* which has $K', \widetilde{C}', \widetilde{C}$ hard-coded and, on input x, if $\widetilde{C}'(x) \neq F(K', x)$ then $C^*(x)$ outputs $\perp$ else it outputs $\widetilde{C}(x)$. The program C^* gives the same output as $\widetilde{C}$ on everything *other than* the marked-points and therefore very closely approximates $\widetilde{C}$ but will not be detected as marked with our detection procedure.

This attack shows that the set of marked-points cannot be static but must depend on the key being marked. But this raises a significant challenge since the detection procedure must be able to find marked-points and test the given program on them without knowing what the underlying key is.

Attack using the detection oracle: If the adversary gets the challenge marked program $\widetilde{C} = i\mathcal{O}(C[K, \boldsymbol{x}, \boldsymbol{y}])$, he can create programs $C'[v](x)$ which evaluate $\widetilde{C}(x)$ unless the first $|v|$ bits of x match the value v in which case $C'[v](x)$ outputs $\perp$. By calling the detection oracle with such program C' (for appropriate choices of v) and seeing if detection succeeds or fails the adversary can use a variant of binary search to learn all the values $\boldsymbol{x}$. Once the adversary learns $\boldsymbol{x}$ he can create a program C^* which evaluates $\widetilde{C}$ on all inputs other than the ones in $\boldsymbol{x}$ and this will remove the mark while maintaining approximate correctness.

We now outline the main ideas for how to solve the above problems.

Security with a Public-Detection Key: Firstly, our main idea to get security in the presence of a detection oracle, is to use *super polynomially many* marked-points. More concretely, the set of marked-points is some super-polynomial sized pseudorandom subset $S \subseteq \mathsf{D}$ of negligible density. The detection procedure only tests the given program on some small (polynomial) number of random points in S. This way, even if the adversary learns the points that were used by polynomially executions of the detection procedure, it does not learn too much about the overall set of marked-points S. Since the set of marked-points is now super-polynomial we cannot simply hard-code the outputs of the marked program at the marked-points. We fix this next while also achieving security against a marking oracle.

Secondly, our main idea to get a scheme with a public-detection key, is to obfuscate the main components of the detection procedure so as to hide the set S of marked-points.

Protection against the marking oracle: Achieving security in the presence of a marking oracle is the most challenging part. To do this, we need to make sure that the set of marked-points x is different for each key K being marked while also ensuring that the detection procedure can find the marked-points without knowing K. We do this by introducing an additional concept that we call "find-points" α which are static and do not depend on K. The set of marked-points x for some key K depends on the values $F(K, \alpha)$ for the find-points α. The detection procedure first tests the program on a find-point α to find the corresponding marked-point x and only then tests it the value of the program on x is some special marked output. By calling the marking oracle on various keys $K^{(1)}, K^{(2)}, \ldots$ the adversary can learn something about the marked-points for each of these keys, but will not learn anything about the set of find-points or the set of marked-points for the challenge key K.

Scheme Outline. To make the above precise, assume we want to mark a PRF family with domain $\mathsf{D} = \{0,1\}^n$ and range $\mathsf{R} = \{0,1\}^m$. We write elements in the domain D as $x = x^{\mathsf{L}} \| x^{\mathsf{R}}$ where $x^{\mathsf{L}} \in \{0,1\}^{n'}$ and $x^{\mathsf{R}} \in \{0,1\}^{n-n'}$ for some prefix length n'. Our construction relies on three auxiliary PRFs $F(K_1, \cdot)$, $F(K_2, \cdot)$, $F(K_3, \cdot)$, with different domains/ranges as will be clear from context, which together make up the marking key $mk = (K_1, K_2, K_3)$.

For each value $x^{\mathsf{L}} \in \{0,1\}^{n'}$ there is a corresponding *find-point* $\alpha = F(K_1, x^{\mathsf{L}})$, which is statically determined by the marking key. There is also a corresponding *marked-point* $x = x^{\mathsf{L}} \| x^{\mathsf{R}}$ which depends on the output of the PRF $F(K, \alpha)$ for the key K being marked and is computed by setting: $\beta := F(K, \alpha), x^{\mathsf{R}} := F(K_2, x^{\mathsf{L}} \| \beta)$. On a marked-point x as above, the marked program outputs a *marked-output* $y = F(K_3, x^{\mathsf{L}} \| \beta)$. Note that there are $\tau = 2^{n'}$ marked-points in total and, by picking n' appropriately, we can ensure that this is a super-polynomial number of marked-points but a negligible fraction of the total domain.

To mark a key K we obfuscate a program that gets some input x, tests if x is a marked point for K, and if so outputs a marked output y, and otherwise computes $F(K, x)$. In the symmetric-key setting, we would use a detection procedure that tests a program C' as follows. It first picks a random value $x^{\mathsf{L}} \leftarrow \{0,1\}^{n_1}$ and computes the corresponding find-point $\alpha := F(K_1, x^{\mathsf{L}})$. Then it computes $\beta = C'(\alpha)$ and uses this to find the corresponding marked-point $x = x^{\mathsf{L}} \| x^{\mathsf{R}}$ where $x^{\mathsf{R}} := F(K_2, x^{\mathsf{L}} \| \beta)$. Finally, it computes $y = C'(x)$ and tests if $y = F(K_3, x^{\mathsf{L}} \| \beta)$ is a marked output (we can repeat this test several time to amplify). In the public-key detection setting, we essentially obfuscate the main components of the detection procedure and release the obfuscation as a public-detection key.

On an intuitive level, the above ensures that even if we are given the marked-points corresponding to some keys K', we do not learn anything about the find-points or about the marked-points for the challenge key K. However, to remove a mark from the challenge marked program, an adversary would need to change the behavior of this program on a large fraction of either the find points or the marked points.

4.2.1 A Mark-Embedding Construction

We now give a detailed description of our mark-embedding scheme (no message) in the public-detection key setting and with security in the presence of a marking oracle.

Setup, Parameters. Recall that our goal is to construct a watermarking scheme for a pPRF family $\mathcal{F}$ with $F(K,\cdot) : \{0,1\}^n \to \{0,1\}^m$. We assume $\mathcal{F}$ has the injective property explained in Section 2.2. As in Section 4.1, we use notation $\mathsf{D} = \{0,1\}^n$ and $\mathsf{R} = \{0,1\}^m$. (Note that n, m are polynomials in λ but we write n instead of $n(\lambda)$ to simplify notation.)

Let $n'' \leq n' \leq n$ be polynomials such that $n - n' = \omega(\log\lambda)$, $n' - n'' = \omega(\log\lambda)$ and $n'' = \omega(\log\lambda)$. Our construction will rely on three auxiliary pPRF families $\mathcal{F}_1$ with $F(K_1,\cdot) : \{0,1\}^{n'} \to \{0,1\}^n$, $\mathcal{F}_2$ with $F(K_2,\cdot) : \{0,1\}^{n'+m} \to \{0,1\}^{n-n'}$, and $\mathcal{F}_3$ with $F(K_3,\cdot) : \{0,1\}^{n'+m} \to \{0,1\}^m$ where $\mathcal{F}_2$ and $\mathcal{F}_3$ are prefix pPRF families as in Definition 2.5. Our construction will also rely on a pseudorandom number generator (PRG) $\mathsf{G}(\cdot) : \{0,1\}^{n''} \to \{0,1\}^{n'}$. In addition to the pPRF families $\mathcal{F}, \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$ and the PRG G, we also rely on an indistinguishability obfuscator $i\mathcal{O}$.

Our proof will consist of $O(\tau)$ hybrids where $\tau := 2^{n'}$ is super-polynomial. In each of these hybrids we will rely on the security of $\mathcal{F}, \mathcal{F}_3$ or $i\mathcal{O}$. This requires us to rely on complexity leveraging, where we assume that a PPT attacker cannot break the security of the schemes $\mathcal{F}, \mathcal{F}_3$ or $i\mathcal{O}$ with probability better than $\mu = \mu(\lambda)$ such that $\tau \cdot \mu$ remains negligible. This can be done if we (e.g.,) assume that $\mathcal{F}, \mathcal{F}_3$ and the $i\mathcal{O}$ have sub-exponential security. We do not need sub-exponential security for $\mathcal{F}_1, \mathcal{F}_2$, and G.

For a string $x \in \mathsf{D}$, we will parse it as $x = x^{\mathsf{L}} \| x^{\mathsf{R}}$ where $x^{\mathsf{L}} \in \{0,1\}^{n'}$ and $x^{\mathsf{R}} \in \{0,1\}^{n-n'}$.

Construction. For any inverse-polynomial approximation factor $\varepsilon = \varepsilon(\lambda)$ we set $\ell = \ell(\lambda) = \lambda/\varepsilon^2$ and define our construction as follows.

$\mathsf{Gen}(1^\lambda)$**:** Choose the marking key $mk := (K_1, K_2, K_3)$ consisting of randomly chosen keys for the PRFs $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$ respectively. Output (dk, mk) where the detection key $dk := (\widetilde{D}_1, \widetilde{D}_2)$ consists of two obfuscated programs $\widetilde{D}_1 \leftarrow i\mathcal{O}(D_1[K_1])$ and $\widetilde{D}_2 \leftarrow i\mathcal{O}(D_2[K_2, K_3])$ defined as:

$$D_1[K_1](r) := F(K_1, \mathsf{G}(r)) \quad , \quad D_2[K_2, K_3](r, \beta) := (F(K_2, \mathsf{G}(r)\|\beta), F(K_3, \mathsf{G}(r)\|\beta)).$$

$\mathsf{Mark}(mk, K)$**:** Output $\widetilde{C} \leftarrow i\mathcal{O}(C[K, K_1, K_2, K_3])$ where the program $C[K, K_1, K_2, K_3]$ is described in Figure 5.

Constants: pPRF keys K, K_1, K_2, K_3.

Inputs: $x \in \mathsf{D}$

Code: Parse $x = x^{\mathsf{L}} \| x^{\mathsf{R}}$. Set $\alpha := F(K_1, x^{\mathsf{L}})$, $\beta := F(K, \alpha)$, $\hat{x}^{\mathsf{R}} := F(K_2, x^{\mathsf{L}} \| \beta)$.
If $x^{\mathsf{R}} = \hat{x}^{\mathsf{R}}$ then output $F(K_3, x^{\mathsf{L}} \| \beta)$ else output $F(K, x)$.

Figure 5: Program $C[K, K_1, K_2, K_3]$

$\mathsf{Detect}(dk, C')$**:** For $i \in [\ell]$: select $r_i \leftarrow \{0,1\}^{n''}$ at random, set $x_i^{\mathsf{L}} := \mathsf{G}(r_i)$, $\alpha_i := \widetilde{D}_1(r_i)$, $\beta_i := C'(\alpha_i)$ and $(x_i^{\mathsf{R}}, y_i) := \widetilde{D}_2(r_i, \beta_i)$. If there is some $i \in [\ell]$ such that $C'(x_i^{\mathsf{L}} \| x_i^{\mathsf{R}}) = y_i$ then output marked, else output unmarked.

The size of program C (resp. D_1, D_2) in this construction is appropriately padded to be the maximum size of all modified programs of C (resp. D_1, D_2), which will appear in the security proof.

We can easily verify that the construction above satisfies statistical correctness, detection correctness, and meaningful properties.

Security Proofs for the Mark-Embedding Construction. We prove the non-removability security of the mark-embedding construction.

Theorem 4.4 *The above watermarking construction is a secure mark-embedding scheme with a public-key detection and security in the presence of a marking oracle, assuming that: $\mathcal{F}, \mathcal{F}_1$ are pPRFs, and $\mathcal{F}_2, \mathcal{F}_3$ are prefix pPRFs, G is a PRG and $i\mathcal{O}$ is a indistinguishability obfuscator and $\mathcal{F}, \mathcal{F}_3, i\mathcal{O}$ are sub-exponentially secure with distinguishing advantage μ such that $\tau\mu$ is negligible where $\tau = 2^{n'}$.*

Lemma 4.5 *Under the same conditions as in Theorem 4.4, for all PPT distinguishers $\mathcal{D}$ it holds that*

$$\left|\Pr[\mathcal{D}^{\mathcal{MO}(mk,\cdot)}(\widetilde{C}, dk, \boldsymbol{\alpha}, \boldsymbol{x}) = 1] - \Pr[\mathcal{D}^{\mathcal{MO}(mk,\cdot)}(\widetilde{C}, dk, \boldsymbol{\alpha'}, \boldsymbol{x'}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

where $(mk = (K_1, K_2, K_3), dk) \leftarrow \mathsf{Gen}(1^\lambda)$, K is a random key for $\mathcal{F}$, $\widetilde{C} \leftarrow \mathsf{Mark}(mk, K)$ and $\mathcal{MO}(mk, \cdot)$ is the marking oracle. The values $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell)$ and $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ correspond to random find-points and mark-points as sampled by the detection procedure and are chosen as follows. For $i = 1, \ldots, \ell$: choose $r_i \leftarrow \{0,1\}^{n''}$ at random, $x_i^{\mathsf{L}} := \mathsf{G}(r_i)$, $\alpha_i := F(K_1, x_i^{\mathsf{L}})$, $\beta_i := F(K, \alpha_i)$, $x_i^{\mathsf{R}} := F(K_2, x_i^{\mathsf{L}} \| \beta_i)$ and $x_i := x_i^{\mathsf{L}} \| x_i^{\mathsf{R}}$. On the other hand $\boldsymbol{\alpha'} \leftarrow \mathsf{D}^\ell$, $\boldsymbol{x'} \leftarrow \mathsf{D}^\ell$ are uniformly random.

Lemma 4.6 *If a PPT adversary $\mathcal{A}$ breaks non-removability security of our construction with probability $\delta(\lambda)$, then we can construct a PPT distinguisher $\mathcal{D}^{\mathcal{MO}(mk,\cdot)}$ such that*

$$\left|\Pr[\mathcal{D}^{\mathcal{MO}(mk,\cdot)}(\widetilde{C}, dk, \boldsymbol{\alpha}, \boldsymbol{x}) = 1] - \Pr[\mathcal{D}^{\mathcal{MO}(mk,\cdot)}(\widetilde{C}, dk, \boldsymbol{\alpha'}, \boldsymbol{x'}) = 1]\right| \geq \delta(\lambda) - \mathsf{negl}(\lambda).$$

where the distributions are defined as in 4.5.

Lemma 4.5 and 4.6 immediately imply Theorem 4.4.

Proof of Lemma 4.6. Given a distribution $(\widetilde{C}, dk, \boldsymbol{\alpha}^*, \boldsymbol{x}^*)$, we construct a distinguisher $\mathcal{D}$ as follows. $\mathcal{D}$ gives dk and $\widetilde{C}$ to the adversary $\mathcal{A}$ as a detection key and a challenge marked program, respectively. If $\mathcal{A}$ attempts to query a PRF key $K^{(j)}$ to the marking oracle, then $\mathcal{D}$ passes it to $\mathcal{MO}(mk, \cdot)$ and replies with the answer of $\mathcal{MO}(mk, K^{(j)})$. Eventually, $\mathcal{A}$ outputs a program C^* and $\mathcal{D}$ proceeds as follows:

1. $\mathcal{D}$ runs $\mathsf{Test}(C^*, \widetilde{C})$. If Test outputs 0, then $\mathcal{D}$ outputs 0. Test is described in Figure 2.
2. If for all $i \in [\ell]$, $(\widetilde{C}(x_i^*) \neq C^*(x_i^*) \vee \widetilde{C}(\alpha_i^*) \neq C^*(\alpha_i^*))$, $\mathcal{D}$ outputs 1, otherwise 0.

The analysis of the distinguishing advantage is similar to that in the proof of Lemma 4.3. In particular, the analysis of Test is the same and ensures that if C^* does ε-approximate $\widetilde{C}$ then the test outputs 1 with overwhelming probability and if C^* does not $\varepsilon/2$-approximate $\widetilde{C}$ then the test outputs 0 with overwhelming probability. We now continue to the analysis of $\mathcal{D}$.

Analysis of $\mathcal{D}$. If the given distribution is the real one $(\widetilde{C}, dk, \boldsymbol{\alpha}, \boldsymbol{x})$, then $\mathcal{A}$ wins the non-removability game with probability δ, meaning that it outputs a program C^* that ε-approximates $\widetilde{C}$ but is not detected as marked by an execution of the detection procedure which evaluates C^* on the points $\boldsymbol{x}, \boldsymbol{\alpha}$. If this happens, then Test passes and outputs 1 with overwhelming probability $1 - \mathsf{negl}(\lambda)$ and, for all i, $(C^*(x_i) \neq \widetilde{C}(x_i) \vee \widetilde{C}(\alpha_i^*) \neq C^*(\alpha_i^*))$. Therefore, in this case, $\mathcal{D}$ outputs 1 with probability at least $\delta - \mathsf{negl}(\lambda)$.

If the given distribution is random one $(\widetilde{C}, dk, \boldsymbol{\alpha}', \boldsymbol{x}')$, then $\boldsymbol{x}'$ and $\boldsymbol{\alpha}'$ are uniformly random and independently distributed. We analyze two cases. One case is that C^* does not $\varepsilon/2$-approximate $\widetilde{C}$. In this case,

the probability that Test passes (outputs 1) is negligible. The other case is that C^* does $\varepsilon/2$-approximate $\widetilde{C}$. In this case, since $x'_1, \ldots, x'_\ell, \alpha'_1, \ldots, \alpha'_\ell$ are 2ℓ random values independent of $\widetilde{C}, C^*$, it holds that

$$\begin{aligned}
&\Pr\left[\forall i \ \left(\widetilde{C}(x'_i) \neq C^*(x'_i)\right) \vee \left(\widetilde{C}(\alpha'_i) \neq C^*(\alpha'_i)\right)\right] \\
=&\prod_i \left(1 - \Pr\left[\left(\widetilde{C}(x'_i) = C^*(x'_i)\right) \wedge \left(\widetilde{C}(\alpha'_i) = C^*(\alpha'_i)\right)\right]\right) \\
&\leq (1 - (\varepsilon/2)^2)^\ell \leq \mathsf{negl}(\lambda)
\end{aligned}$$

if $\ell = O(\lambda/\varepsilon^2)$. Therefore, if the distribution is a random the probability that $\mathcal{D}$ outputs 1 is at most $\mathsf{negl}(\lambda)$. This means that $\mathcal{D}$ has distinguishing advantage $\delta - \mathsf{negl}(\lambda)$. □

Proof of Lemma 4.5. We define a sequence of hybrid games to prove this lemma. We call the left-hand side distribution in Lemma 4.5 the real distribution REAL and the right-hand side distribution is the random distribution RAND. Similarly in the proof of Lemma 4.2 in Section 4.1, we will transform the challenge program in the real distribution into another program that does not depend on K_1, K_2, K_3 and then replace $\boldsymbol{x}$ and $\boldsymbol{\alpha}$ with uniformly and independently random $\boldsymbol{x'}$ and $\boldsymbol{\alpha'}$, respectively. We define the following main hybrid games (but proving the indistinguishability of some of these will require more intermediate hybrids). See Table 1 for an abbreviated overview of the hybrids emphasizing the differences between them.

REAL**:** This is the real distribution (left-hand side in Lemma 4.5). The distinguisher is given the values

$$(\widetilde{C}, dk = (\widetilde{D}_1, \widetilde{D}_2), \boldsymbol{\alpha}, \boldsymbol{x})$$

and access to the marking oracle $\mathcal{MO}(mk, \cdot)$. Recall that $(mk = (K_1, K_2, K_3), dk) \leftarrow \mathsf{Gen}(1^\lambda)$, K is a random key for $\mathcal{F}$ and $\widetilde{C} \leftarrow \mathsf{Mark}(mk, K)$. The values $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell)$ and $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ are chosen as follows. For $i = 1, \ldots, \ell$: choose $r_i \leftarrow \{0,1\}^{n''}$ at random, $x_i^{\mathsf{L}} := \mathsf{G}(r_i)$, $\alpha_i := F(K_1, x_i^{\mathsf{L}})$, $\beta_i := F(K, \alpha_i)$, $x_i^{\mathsf{R}} := F(K_2, x_i^{\mathsf{L}} \| \beta_i)$ and $x_i := x_i^{\mathsf{L}} \| x_i^{\mathsf{R}}$.

Hyb_1**:** For $i \in [\ell]$, instead of choosing $x_i^{\mathsf{L}} := \mathsf{G}(r_i)$, we now choose $x_i^{\mathsf{L}} \leftarrow \{0,1\}^{n'}$ uniformly at random.

Hyb_2**:** Instead of choosing challenge program $\widetilde{C} \leftarrow i\mathcal{O}(C[K, K_1, K_2, K_3])$ where the program $C[K, K_1, K_2, K_3]$ is described in Figure 5, we now set $\widetilde{C} \leftarrow i\mathcal{O}(C[K])$ where $C[K](x) := F(K, x)$ just evaluates the PRF. Also, instead of choosing the detection key component $\widetilde{D}_2 \leftarrow i\mathcal{O}(D_2[K_2, K_3])$ as defined in the scheme description, we now choose $\widetilde{D}_2 \leftarrow i\mathcal{O}(D'_2[K, K_1, K_2, K_3])$ as defined in Figure 6. [4]

Constants: pPRF keys K, K_1, K_2, K_3.

Inputs: r, β

Code: Set $x^{\mathsf{L}} := \mathsf{G}(r)$, $\alpha := F(K_1, x^{\mathsf{L}})$, $x^{\mathsf{R}} := F(K_2, x^{\mathsf{L}} \| \beta)$.
If $\beta = F(K, \alpha)$, then set $y := F(K, x^{\mathsf{L}} \| x^{\mathsf{R}})$ else set $y := F(K_3, x^{\mathsf{L}} \| \beta)$. Output (x^{R}, y).

Figure 6: Program $D'_2[K, K_1, K_2, K_3]$

Hyb_3**:** In this hybrid the values $\boldsymbol{x}$ and $\boldsymbol{\alpha}$ are replaced with uniformly random elements $\boldsymbol{x'} \leftarrow \mathsf{D}^\ell$ and $\boldsymbol{\alpha'} \leftarrow \mathsf{D}^\ell$, respectively.

RAND**:** This is the random distribution (right-hand side in Lemma 4.5). The only changes from Hyb_3 are that the challenge program $\widetilde{C}$ and detection key $\widetilde{D}_2$ are changed back to the original programs but the values $\boldsymbol{x'}, \boldsymbol{\alpha'}$ remain random.

Table 1: An overview of hybrid games

Hybrid game	Challenge: $i\mathcal{O}(\cdot)$	dk: $i\mathcal{O}(\cdot)$	$\boldsymbol{\alpha}$	$\boldsymbol{x}^{\mathsf{L}}$	$\boldsymbol{x}^{\mathsf{R}}$
REAL	$C[K, K_1, K_2, K_3]$	$D_2[K_2, K_3]$	$F(K_1, \boldsymbol{x}^{\mathsf{L}})$	$\mathsf{G}(\boldsymbol{r})$	$F(K_2, \boldsymbol{x}^{\mathsf{L}} \Vert \boldsymbol{\beta})$
Hyb_1	$C[K, K_1, K_2, K_3]$	$D_2[K_2, K_3]$	$F(K_1, \boldsymbol{x}^{\mathsf{L}})$	random	$F(K_2, \boldsymbol{x}^{\mathsf{L}} \Vert \boldsymbol{\beta})$
Hyb_2	$C[K]$	$D_2'[K, K_1, K_2, K_3]$	$F(K_1, \boldsymbol{x}^{\mathsf{L}})$	random	$F(K_2, \boldsymbol{x}^{\mathsf{L}} \Vert \boldsymbol{\beta})$
Hyb_3	$C[K]$	$D_2'[K, K_1, K_2, K_3]$	random	random	random
RAND	$C[K, K_1, K_2, K_3]$	$D_2[K_2, K_3]$	random	random	random

Lemma 4.7 *If* G *is a pseudo-random number generator, then* $\mathsf{REAL} \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_1$.

Proof of Lemma 4.7. These hybrid games are indistinguishable due to the pseudorandomness of G. □

Notation. Before we proceed to the rest of the proof, we introduce some notation for puncturing PRFs. We write $K\{x \to y\}$ to denote a PRF key K which is punctured at the points x with a *hard-coded output* y. In other words this denotes the process of first creating a punctured key $K\{x\} := \mathsf{Puncture}(K, x)$ and storing the tuple $K\{x \to y\} = (K\{x\}, x, y)$. We define $F(K\{x \to y\}, x) = y$ and else $F(K\{x \to y\}, x') = F(K\{x\}, x') = F(K, x')$ when $x' \neq x$.

We also use similar notation for prefix pPRFs where we write $K\{s\Vert* \to y\} = (K\{s\Vert*\}, s, y)$. For any input x that has s as a prefix, we define $F(K\{s\Vert* \to y\}, x) = y$ and for all other x' we define $F(K\{s\Vert* \to y\}, x') = F(K\{s\Vert*\}, x') = F(K, x')$.

Lastly we use the notation $K\{x \to \bot\}$ to denote $K\{x \to y_\bot\}$ for some fixed "dummy value" $y_\bot$ in the range of $F(K, \cdot)$, say the all 0 string.

Lemma 4.8 *If* $\mathcal{F}$ *and* $\mathcal{F}_3$ *are sub-exponentially secure pPRFs,* $\mathcal{F}$ *satisfies the injective property in Definition 2.6, and* $i\mathcal{O}$ *is a sub-exponentially secure indistinguishability obfuscation, so that the distinguishing advantage for each of these is bounded by some function* μ *such that* $\tau\mu = \mathsf{negl}(\lambda)$*, then* $\mathsf{Hyb}_1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2$.

Proof of Lemma 4.8. To prove indistinguishability of $\mathsf{Hyb}_1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2$, we a introduce of intermediate hybrid games corresponding to each one of the values $x^{\mathsf{L}} \in \{0,1\}^{n'}$. We interchangeably identify the strings $x^{\mathsf{L}} \in \{0,1\}^{n'}$ with integers $j \in \{0, \ldots, \tau - 1\}$ for $\tau = 2^{n'}$ via the usual binary representation. We define the following hybrids games (see Table 2 for a concise overview).

Hyb_1^j**:** Everything is the same as in Hyb_1 except that the challenge program $\widetilde{C}$ and the detection key component $\widetilde{D}_2$ are changed to:

- $\widetilde{C} \leftarrow i\mathcal{O}(C^j[K, K_1, K_2, K_3])$ where $C^j[K, K_1, K_2, K_3]$ is described in Figure 7,
- $\widetilde{D}_2 \leftarrow i\mathcal{O}(D_2^j[K, K_2, K_3])$ where $D_2^j[K, K_2, K_3]$ is described in Figure 8.

Hyb_1^j**-1:** Everything is the same as in Hyb_1^j except that we always use the punctured keys

$$K\{(j\Vert x_j^{\mathsf{R}}) \to \bot\} \quad \text{and} \quad K_3\{(j\Vert\beta_j) \to y_j\}$$

in all obfuscated programs in place of K, K_3, where we define $\alpha_j = F(K_1, j)$, $\beta_j = F(K, \alpha_j)$, $x_j^{\mathsf{R}} = F(K_2, j\Vert\beta_j)$ and $y_j = F(K_3, j\Vert\beta_j)$. In particular, this means that we obfuscate the following programs:

[4] As a sanity check, note that this ensures that the detection procedure still outputs marked when executed on $\widetilde{C}$.

Constants: pPRF keys K, K_1, K_2, K_3.

Inputs: $x \in \mathsf{D}$

Code: Parse $x = x^{\mathsf{L}} \| x^{\mathsf{R}}$. Set $\alpha := F(K_1, x^{\mathsf{L}})$, $\beta := F(K, \alpha)$, $\hat{x}^{\mathsf{R}} := F(K_2, x^{\mathsf{L}} \| \beta)$.
If $x^{\mathsf{R}} \neq \hat{x}^{\mathsf{R}}$ or $x^{\mathsf{L}} < j$, then output $F(K, x)$ else output $F(K_3, x^{\mathsf{L}} \| \beta)$.

Figure 7: Program $C^j[K, K_1, K_2, K_3]$

Constants: pPRF keys K, K_1, K_2, K_3.

Inputs: r, β

Code: Set $x^{\mathsf{L}} := \mathsf{G}(r)$, $\alpha := F(K_1, x^{\mathsf{L}})$, $x^{\mathsf{R}} := F(K_2, x^{\mathsf{L}} \| \beta)$.
If $x^{\mathsf{L}} < j$ and $\beta = F(K, \alpha)$, then set $y := F(K, x^{\mathsf{L}} \| x^{\mathsf{R}})$ else set $y := F(K_3, x^{\mathsf{L}} \| \beta)$.
Output (x^{R}, y).

Figure 8: Program $D_2^j[K, K_1, K_2, K_3]$

Table 2: An overview of hybrid games from Hyb_1 to Hyb_2

Hybrid	Challenge: $i\mathcal{O}(C[\cdot])$	dk: $i\mathcal{O}(D_2[\cdot])$	i-th query: $i\mathcal{O}(C[\cdot])$	hard-wired y_j
Hyb_1^0	$C[K, K_1, K_2, K_3]$	$D_2[K_2, K_3]$	$C[K^{(i)}, K_1, K_2, K_3]$	none
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Hyb_1^j	$C^j[K, K_1, K_2, K_3]$	$D_2^j[K, K_1, K_2, K_3]$	$C[K^{(i)}, K_1, K_2, K_3]$	none
Hyb_1^j-1	C^j[punctured]*	D_2^j[punctured]*	C[punctured]*	$F(K_3, j\|\beta)$
Hyb_1^j-2	C^j[punctured]*	D_2^j[punctured]*	C[punctured]*	$F(K, j\|x_j^{\mathsf{R}})$
Hyb_1^{j+1}	$C^{j+1}[K, K_1, K_2, K_3]$	$D_2^{j+1}[K, K_1, K_2, K_3]$	$C[K^{(i)}, K_1, K_2, K_3]$	none
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Hyb_2	$C^{\tau+1}[K, K_1, K_2, K_3]$	$D_2^{\tau+1}[K, K_1, K_2, K_3]$	$C[K^{(i)}, K_1, K_2, K_3]$	none

* In this table, [punctured] means keys K and K_3 are punctured. Note that keys K_1, K_2, and $K^{(i)}$ are *not* punctured. See the definitions of hybrid games for details.

- The challenge program $\widetilde{C}$ obfuscates $C^j[K\{j\|x_j^{\mathsf{R}} \to \bot\}, K_1, K_2, K_3\{j\|\beta_j \to y_j\}]$.
- The detection key component $\widetilde{D}_2$ obfuscates $D_2^j[K\{j\|x_j^{\mathsf{R}} \to \bot\}, K_1, K_2, K_3\{j\|\beta_j \to y_j\}]$.
- Each query to the marking oracle with some PRF key $K^{(i)}$ outputs a program $\widetilde{C}^{(i)}$ that obfuscates $C[K^{(i)}, K_1, K_2, K_3\{j\|\beta_j \to y_j\}]$.

Hyb_1^j-2: Everything is the same as in Hyb_1^j-1 except that use the hard-wired value $y_j := F(K, j\|x_j^{\mathsf{R}})$ instead of $y_j = F(K_3, j\|\beta_j)$.

Let μ be the maximal distinguishing advantage of a PPT attacker against the iO scheme and the pPRFs $\mathcal{F}, \mathcal{F}_3$. Let q be the number of queries to the marking oracle. Then the following hybrids are indistinguishable for all $j \in \{0, \ldots, \tau\}$.

Claim: $\mathsf{Hyb}_1^j \overset{\mathsf{c}}{\approx}_{(q+2)\mu} \mathsf{Hyb}_1^j$**-1.** This holds by iO security since the programs being obfuscated in both hybrids are functionally equivalent. In particular, none of the programs ever evaluates $F(K, j\|x_j^{\mathsf{R}})$ and

therefore we can use a punctured key $K\{(j\|x_j^{\mathsf{R}}) \to \bot\}$ in place of K without changing functionality. On the other hand the keys K_3 and $K_3\{(j\|\beta_j) \to y_j\}$ are functionally equivalent.

There are $q+2$ obfuscated programs in total and therefore we pay a factor of $q+2$ in the distinguishing advantage: the q outputs of the marking oracle $\widetilde{C}^{(i)}$, the challenge program $\widetilde{C}$, and the detection key program $\widetilde{D}_2$.

Claim: Hyb_1^j**-1** $\overset{\mathsf{c}}{\approx}_{2\mu}$ Hyb_1^j**-2.** This holds by the pPRF security of $\mathcal{F}$ and $\mathcal{F}_3$. In particular, given the punctured keys $K\{j\|x_j^{\mathsf{R}}\}$ and $K_3\{j\|\beta_j\}$, the value $F(K_3, j\|\beta_j)$ is indistinguishable from a uniformly random value which is in turn indistinguishable from $F(K, j\|x_j^{\mathsf{R}})$.

Claim: Hyb_1^j**-2** $\overset{\mathsf{c}}{\approx}_{(q+2)\mu}$ Hyb_1^{j+1}**.** This holds by iO security since the programs being obfuscated in both hybrids are functionally equivalent. In particular, in both hybrids, the obfuscated challenge program $\widetilde{C}$ and detection-key program $\widetilde{D}_2$ are using $F(K, j\|\beta_j)$ in place of $F(K_3, j\|\beta_j)$. On the other hand, each of the programs output by the marking oracle $\widetilde{C}^{(i)}$ never evaluates $F(K_3, j\|\beta_j)$ and hence it does not matter whether these programs use a punctured key or not. This follows since, each program $\widetilde{C}^{(i)}$ evaluates $F(K_3, j\|\beta_{i,j})$ where $\beta_{i,j} = F(K^{(i)}, \alpha_j)$ is different from $\beta_j = F(K, \alpha_j)$ by the injective property of the pPRF $\mathcal{F}$.[5]

Moreover, we have $\mathsf{Hyb}_1 \overset{\mathsf{c}}{\approx}_{2\mu} \mathsf{Hyb}_1^0$ and $\mathsf{Hyb}_1^\tau \overset{\mathsf{c}}{\approx}_{2\mu} \mathsf{Hyb}_2$ since the programs being obfuscated to get $\widetilde{C}$ and $\widetilde{D}_2$ in these hybrids are functionally equivalent. Combining this and the the above claims, we get $\mathsf{Hyb}_1 \overset{\mathsf{c}}{\approx}_{O(q\tau\mu)} \mathsf{Hyb}_2$. Since $\tau\mu = \mathsf{negl}(\lambda)$ and $q = \mathrm{poly}(\lambda)$ we get the computational indistinguishability $\mathsf{Hyb}_1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2$ which proves the lemma.

□

Lemma 4.9 *If $\mathcal{F}_1$ is a secure pPRF, $\mathcal{F}_2, \mathcal{F}_3$ are secure prefix pPRFs, and $i\mathcal{O}$ is secure indistinguishability obfuscation, then it holds* $\mathsf{Hyb}_2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_3$.

Proof of Lemma 4.9. To prove the indistinguishability of these hybrid games, we introduce intermediate hybrid games for $i = 1, \ldots, \ell + 1$.

Hyb_2^i**:** This is the same as Hyb_2 except for how the values $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell), \boldsymbol{x} = (x_1, \ldots, x_\ell)$ are chosen. For $i' < i$, instead of choosing $\alpha_{i'} = F(K_1, x_{i'}^{\mathsf{L}})$, $\beta_{i'} = F(K, \alpha_{i'})$, $x_{i'}^{\mathsf{R}} = F(K_2, x_{i'}^{\mathsf{L}}\|\beta_{i'})$ and setting $x_{i'} = x_{i'}^{\mathsf{L}}\|x_{i'}^{\mathsf{R}}$ we now choose uniformly random values $\alpha'_{i'} \leftarrow \mathsf{D}$, $x'_{i'} \leftarrow \mathsf{D}$.

Hyb_2^i**-1:** This is the same as Hyb_2^i but instead of the PRF keys K_1 and K_2, K_3 we use punctured keys $K_1\{x_i^{\mathsf{L}}\}$, $K_2\{x_i^{\mathsf{L}}\|*\}$, $K_3\{x_i^{\mathsf{L}}\|*\}$ in all obfuscated programs – namely, in the detection key $\widetilde{D}_1, \widetilde{D}_2$ and in the marked programs $\widetilde{C}^{(j)}$ output by the marking oracle (note that K_1, K_2, K_3 do not appear in the challenge program $\widetilde{C}$ at this point). We do this as follows:

- For the detection key $\widetilde{D}_1, \widetilde{D}_2$ we can simply output $\bot$ on the punctured inputs. In other words, $\widetilde{D}_1, \widetilde{D}_2$ are respectively obfuscations of

$$D_1[K_1\{x_i^{\mathsf{L}} \to \bot\}] \quad , \quad D_2'[K, K_1\{x_i^{\mathsf{L}} \to \bot\}, K_2\{x_i^{\mathsf{L}}\|* \to \bot\}, K_3\{x_i^{\mathsf{L}}\|* \to \bot\}].$$

[5]Technically, we need to ensure that the distinguisher never queries the marking oracle with the key $K^{(i)} = K$ contained in the challenge program. This is without loss of generality since if the distinguisher can ever guess K then it can already perfectly distinguish the two challenge distributions without making additional calls to the marking oracle. In other words, we can convert any distinguisher into one with the above restriction while maintaining the same distinguishing advantage.

Table 3: An overview of hybrid games from Hyb_2^i to Hyb_2^{i+1}

Hybrid	K_1	K_b $b \in \{2,3\}$	$\boldsymbol{\alpha}_i$	$\boldsymbol{x}_i^{\mathsf{R}}$	hard-wired PRF outputs in $C[K^{(j)}, \cdot]$		hard-wired PRF outputs in $D_1[\cdot]$ and $D_2'[\cdot]$
Hyb_2^i	K_1	K_b	$F(K_1, \boldsymbol{x}_i^{\mathsf{L}})$	$F(K_2, \boldsymbol{x}_i^{\mathsf{L}} \Vert \boldsymbol{\beta}_i)$	none		none
Hyb_2^i-1	$K_1\{\boldsymbol{x}_i^{\mathsf{L}}\}$	$K_b\{\boldsymbol{x}_i^{\mathsf{L}} \Vert *\}$	$F(K_1, \boldsymbol{x}_i^{\mathsf{L}})$	$F(K_2, \boldsymbol{x}_i^{\mathsf{L}} \Vert \boldsymbol{\beta}_i)$	$\hat{\boldsymbol{x}}_{j,i}^{\mathsf{R}}$	$y_{j,i}$	$\perp$
Hyb_2^i-2	$K_1\{\boldsymbol{x}_i^{\mathsf{L}}\}$	$K_b\{\boldsymbol{x}_i^{\mathsf{L}} \Vert *\}$	$F(K_1, \boldsymbol{x}_i^{\mathsf{L}})$	random	random	random	$\perp$
Hyb_2^i-3	$K_1\{\boldsymbol{x}_i^{\mathsf{L}}\}$	$K_b\{\boldsymbol{x}_i^{\mathsf{L}} \Vert *\}$	random	random	random	random	$\perp$
Hyb_2^i-4	$K_1\{\boldsymbol{x}_i^{\mathsf{L}}\}$	$K_b\{\boldsymbol{x}_i^{\mathsf{L}} \Vert *\}$	random	random	$\hat{\boldsymbol{x}}_{j,i}^{\mathsf{R}}$	$y_{j,i}$	$\perp$
Hyb_2^{i+1}	K_1	K_b	random	random	none		none

- For each query $K^{(j)}$ to the marking oracle, we compute $\alpha_i = F(K_1, x_i^{\mathsf{L}})$, $\beta_{j,i} = F(K^{(j)}, \alpha_i)$, $x_{j,i}^{\mathsf{R}} = F(K_2, x_i^{\mathsf{L}} \Vert \beta_{j,i})$, $y_{j,i} = F(K_3, x_i^{\mathsf{L}} \Vert \beta_{j,i})$ and we obfuscate the program

$$C[K^{(j)}, K_1\{x_i^{\mathsf{L}} \to \perp\}, K_2\{x_i^{\mathsf{L}} \Vert * \to x_{j,i}^{\mathsf{R}}\}, K_3\{x_i^{\mathsf{L}} \Vert * \to y_{j,i}\}].$$

  We can rewrite the code of the above program in a simplified (but functionally equivalent) manner as shown Figure 9.

Constants: pPRF keys $K^{(j)}, K_1\{x_i^{\mathsf{L}} \to \perp\}, K_2\{x_i^{\mathsf{L}} \Vert * \to x_{j,i}^{\mathsf{R}}\}, K_3\{x_i^{\mathsf{L}} \Vert * \to y_{j,i}\}$.

Inputs: $x \in \mathsf{D}$

Code: Parse $x = x^{\mathsf{L}} \Vert x^{\mathsf{R}}$. If $x^{\mathsf{L}} = x_i^{\mathsf{L}}$, then set $\hat{x}^{\mathsf{R}} := x_{j,i}^{\mathsf{R}}$ and $\hat{y} = y_{j,i}$.
Else, set $\alpha := F(K_1\{x_i^{\mathsf{L}}\}, x^{\mathsf{L}})$, $\beta := F(K^{(j)}, \alpha)$, $\hat{x}^{\mathsf{R}} := F(K_2\{x_i^{\mathsf{L}} \Vert *\}, x^{\mathsf{L}} \Vert \beta)$, $\hat{y} = F(K_3, x^{\mathsf{L}} \Vert \beta)$.
If $x^{\mathsf{R}} = \hat{x}^{\mathsf{R}}$, then output $\hat{y}$ else output $F(K^{(j)}, x)$.

Figure 9: Program $C[K^{(j)}, K_1\{\boldsymbol{x}_i^{\mathsf{L}} \to \perp\}, K_2\{\boldsymbol{x}_i^{\mathsf{L}} \Vert * \to \hat{\boldsymbol{x}}_{j,i}^{\mathsf{R}}\}, K_3\{x_i^{\mathsf{L}} \Vert * \to y_{j,i}\}]$

Hyb_2^i-2: This is the same as Hyb_2^i-1 except that:

- When answering the j'th marking-oracle query, we choose the values $x_{j,i}^{\mathsf{R}}$ and $y_{j,i}$ uniformly at random instead of computing $\alpha_i = F(K_1, x_i^{\mathsf{L}})$, $\beta_{j,i} = F(K^{(j)}, \alpha_i)$, $x_{j,i}^{\mathsf{R}} = F(K_2, x_i^{\mathsf{L}} \Vert \beta_{j,i})$, $y_{j,i} = F(K_3, x_i^{\mathsf{L}} \Vert \beta_{j,i})$.
- In $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ we now sample $x_i = x_i^{\mathsf{L}} \Vert x_i^{\mathsf{R}}$ by choosing x_i^{R} uniformly at random instead of computing $\beta_i = F(K, \alpha_i)$, $x_i^{\mathsf{R}} = F(K_2, x_i^{\mathsf{L}} \Vert \beta_i)$. This is equivalent to sampling $x_i \leftarrow \mathsf{D}$ uniformly at random.

Hyb_2^i-3: This is the same as Hyb_2^i-2 except that, in $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell)$, we now select $\alpha_i \leftarrow \mathsf{D}$ uniformly at random instead of $\alpha_i = F(K_1, x_i^{\mathsf{L}})$.

Hyb_2^i-4: This is the same as Hyb_2^i-3 except that, when answering the j'th marking-oracle query, we switch back to choosing $x_{j,i}^{\mathsf{R}}$ and $y_{j,i}$ by computing $\alpha_i = F(K_1, x_i^{\mathsf{L}})$, $\beta_{j,i} = F(K^{(j)}, \alpha_i)$, $x_{j,i}^{\mathsf{R}} = F(K_2, x_i^{\mathsf{L}} \Vert \beta_{j,i})$, $y_{j,i} = F(K_3, x_i^{\mathsf{L}} \Vert \beta_{j,i})$. Note that the value α_i in the vector $\boldsymbol{\alpha}$ is still chosen uniformly at random.

We describe an overview of hybrid games in Table 3 to emphasize differences. The following hybrids are indistinguishable for all $i \in \{1, \ldots, \ell + 1\}$.

Claim: $\mathsf{Hyb}_2^i \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2^i$**-1.** This follows by iO security since the programs being obfuscated are functionally equivalent. This is clear for the responses of the marking oracle. For the detection key components $\widetilde{D}_1, \widetilde{D}_2$, note that the programs D_1, D_2' only evaluate K_1, K_2, K_3 on the points of the form x^{L} or $x^{\mathsf{L}} \| *$ where $x^{\mathsf{L}} = \mathsf{G}(r)$ for some r. However, since x_i^{L} is chosen uniformly at random, it is not of this form (with overwhelming probability) and therefore we can puncture the keys at x_i^{L} or $x_i^{\mathsf{L}} \| *$ without changing functionality.

Claim: Hyb_2^i**-1** $\overset{\mathsf{c}}{\approx}$ Hyb_2^i**-2.** This follows by the prefix pPRF security of $\mathcal{F}_2, \mathcal{F}_3$. In particular, since we use punctured keys $K_2\{x_i^{\mathsf{L}} \| *\}$, $K_3\{x_i^{\mathsf{L}} \| *\}$ their outputs are indistinguishable from random on inputs of the form $x_i^{\mathsf{L}} \| *$. Note that by injectivity of $\mathcal{F}$ the values β_i and $\beta_{j,i}$ are distinct and therefore we can replace the PRF outputs by random and independent values.

Claim: Hyb_2^i**-2** $\overset{\mathsf{c}}{\approx}$ Hyb_2^i**-3.** This follows by the pPRF security of $\mathcal{F}_1$. In particular, since we use a punctured key $K_1\{x_i^{\mathsf{L}}\}$ we can replace $F(K_1, x_i^{\mathsf{L}})$ by uniform.

Claim: Hyb_2^i**-3** $\overset{\mathsf{c}}{\approx}$ Hyb_2^i**-4.** This follows by the prefix pPRF security of $\mathcal{F}_2, \mathcal{F}_3$ (same as showing Hyb_2^i-1 $\overset{\mathsf{c}}{\approx}$ Hyb_2^i-2).

Claim: Hyb_2^i**-4** $\overset{\mathsf{c}}{\approx}$ Hyb_2^{i+1}**:** This follows by iO security (same as showing $\mathsf{Hyb}_2^i \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2^i$-1).

Lastly, by noting that $\mathsf{Hyb}_2 = \mathsf{Hyb}_2^1$ and $\mathsf{Hyb}_2^{\ell+1} = \mathsf{Hyb}_3$ we get $\mathsf{Hyb}_2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_3$ which proves the lemma. □

Lemma 4.10 $\mathsf{Hyb}_3 \overset{\mathsf{c}}{\approx} \mathsf{RAND}$.

Proof of Lemma 4.10. This proof is the same as the proof of Lemma 4.8 (in reverse manner). □

By Lemma 4.7, 4.8, 4.9, 4.10, we complete the proof of Lemma 4.5. □

4.2.2 A Message-Embedding Construction

In this section, we propose a message-embedding watermarking scheme in the public-key detection setting and with selective-message security in the presence of a marking oracle. We can get a scheme that allows for any approximation factor $\varepsilon(\lambda) = \frac{1}{\sqrt{2}} + 1/\mathrm{poly}(\lambda)$.

Setup. Our construction can be thought of as an "upgrade" of the construction in Section 4.2.1. As before, our goal is to construct a watermarking scheme for a pPRF family $\mathcal{F}$ with $F(K, \cdot) : \{0,1\}^n \to \{0,1\}^m$. We set the message space to be some $\mathcal{M} \subseteq \{0,1\}^m$. Let the pPRF $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$ and the PRG G be defined as in Section 4.2.1.

Construction. For any approximation factor $\varepsilon(\lambda) = \frac{1}{\sqrt{2}} + \rho(\lambda)$ where $\rho(\lambda)$ is some inverse polynomial, we set $\ell = \ell(\lambda) = \lambda/\rho^2$ and define our construction as follows.

$\mathsf{Gen}(1^\lambda)$**:** This is exactly the same as the key-generation procedure of our mark-embedding construction. Choose the marking key $mk := (K_1, K_2, K_3)$ consisting of randomly chosen keys for the PRFs $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$ respectively. Output (dk, mk) where the detection key $dk := (\widetilde{D}_1, \widetilde{D}_2)$ consists of two obfuscated programs $\widetilde{D}_1 \leftarrow i\mathcal{O}(D_1[K_1])$ and $\widetilde{D}_2 \leftarrow i\mathcal{O}(D_2[K_2, K_3])$ defined as:

$$D_1[K_1](r) := F(K_1, \mathsf{G}(r)) \quad , \quad D_2[K_2, K_3](r, \beta) := (F(K_2, \mathsf{G}(r)\|\beta), F(K_3, \mathsf{G}(r)\|\beta)).$$

Constants: pPRF keys K, K_1, K_2, K_3 and message msg.

Inputs: $x \in \mathsf{D}$

Code: Parse $x = x^{\mathsf{L}} \| x^{\mathsf{R}}$. Set $\alpha := F(K_1, x^{\mathsf{L}})$, $\beta := F(K, \alpha)$, $\hat{x}^{\mathsf{R}} := F(K_2, x^{\mathsf{L}} \| \beta)$.
If $x^{\mathsf{R}} = \hat{x}^{\mathsf{R}}$ then output $F(K_3, x^{\mathsf{L}} \| \beta) \oplus \mathsf{msg}$ else output $F(K, x)$.

Figure 10: Program $C[K, K_1, K_2, K_3, \mathsf{msg}]$

$\mathsf{Mark}(mk, K, \mathsf{msg})$**:** Output $i\mathcal{O}(C[K, K_1, K_2, K_3, \mathsf{msg}])$ where the program $C[K, K_1, K_2, K_3, \mathsf{msg}]$ is described in Figure 10.

$\mathsf{Detect}(dk, C')$**:** For $i = 1, \ldots, \ell$: Select r_i at random, set $x_i^{\mathsf{L}} := \mathsf{G}(r_i)$, $\alpha_i := \widetilde{D}_1(r_i)$, $\beta_i := C'(\alpha_i)$ and $(x_i^{\mathsf{R}}, y_i) := \widetilde{D}_2(r_i \| \beta_i)$, and compute $\mathsf{msg}_i := C'(x_i^{\mathsf{L}} \| x_i^{\mathsf{R}}) \oplus y_i$. If there exists a "majority value" msg such that $|\{i : \mathsf{msg} = \mathsf{msg}_i\}| > \ell/2$ then output msg, else output $\mathsf{unmarked}$.

We can easily verify that the above construction above satisfies statistical correctness, detection correctness, and meaningful properties.

Security Proofs for the Message-Embedding Construction. We prove the non-removability security of the above construction in the selective-message setting, following essentially the same strategy as the mark-embedding construction with small modifications.

Theorem 4.11 *The above construction is a message-embedding watermarking scheme with public-detection and selective-message security in the presence of a marking oracle for any approximation factor* $\varepsilon(\lambda) = \frac{1}{\sqrt{2}} + 1/\mathrm{poly}(\lambda)$ *under the same conditions as in Theorem 4.4.*

The above theorem follows from the following two lemmas.

Lemma 4.12 *Under the conditions of Theorem 4.4, it holds that for all PPT* $\mathcal{D}$ *and all message* $\mathsf{msg} \in \mathcal{M}$*:*

$$\left|\Pr[\mathcal{D}^{\mathcal{MO}(mk,\cdot,\cdot)}(\widetilde{C}, dk, \boldsymbol{\alpha}, \boldsymbol{x}) = 1] - \Pr[\mathcal{D}^{\mathcal{MO}(mk,\cdot,\cdot)}(\widetilde{C}, dk, \boldsymbol{\alpha'}, \boldsymbol{x'}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

where $(mk = (K_1, K_2, K_3), dk) \leftarrow \mathsf{Gen}(1^\lambda)$*,* K *is a random pPRF key,* $\widetilde{C} \leftarrow \mathsf{Mark}(mk, K, \mathsf{msg})$*,* $\mathcal{MO}(mk, \cdot, \cdot)$ *is the marking oracle that gets as input a pPRF key* $K^{(j)}$ *and a message* msg_j *and outputs* $\widetilde{C} \leftarrow \mathsf{Mark}(mk, K^{(j)}, \mathsf{msg}_j)$*. The values* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell)$ *and* $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ *correspond to random find-points and mark-points as sampled by the detection procedure and are chosen as follows. For* $i = 1, \ldots, \ell$*: choose* $r_i \leftarrow \{0,1\}^{n''}$ *at random,* $x_i^{\mathsf{L}} := \mathsf{G}(r_i)$*,* $\alpha_i := F(K_1, x_i^{\mathsf{L}})$*,* $\beta_i := F(K, \alpha_i)$*,* $x_i^{\mathsf{R}} := F(K_2, x_i^{\mathsf{L}} \| \beta_i)$ *and* $x_i := x_i^{\mathsf{L}} \| x_i^{\mathsf{R}}$*. On the other hand* $\boldsymbol{\alpha'} \leftarrow \mathsf{D}^\ell, \boldsymbol{x'} \leftarrow \mathsf{D}^\ell$ *are uniformly random.*

Lemma 4.13 *If a PPT adversary* $\mathcal{A}$ *breaks selective-message non-removability security of the given scheme with probability* $\delta(\lambda)$ *and approximation factor* $\varepsilon > \frac{1}{\sqrt{2}} + \rho$*, then we can construct a PPT distinguisher* $\mathcal{D}^{\mathcal{MO}(mk,\cdot,\cdot)}$ *and some message* $\mathsf{msg} \in \mathcal{M}$ *such that*

$$\left|\Pr[\mathcal{D}^{\mathcal{MO}(mk,\cdot,\cdot)}(\widetilde{C}, dk, \boldsymbol{\alpha}, \boldsymbol{x}) = 1] - \Pr[\mathcal{D}^{\mathcal{MO}(mk,\cdot,\cdot)}(\widetilde{C}, dk, \boldsymbol{\alpha'}, \boldsymbol{x'}) = 1]\right| \geq \delta(\lambda) - \mathsf{negl}(\lambda).$$

where the distributions are those of 4.12.

Proof of Lemma 4.13. This proof is similar to that of Lemmas 4.3 and 4.6 with some minor differences. In particular, the distinguisher $\mathcal{D}$ sets msg to be the message chosen by the adversary $\mathcal{A}$ at the beginning of the game, and gets the challenge values $(\widetilde{C}, dk, \boldsymbol{\alpha}^*, \boldsymbol{x}^*)$, where $\boldsymbol{\alpha}^*$ and $\boldsymbol{x}^*$ is either $(\boldsymbol{\alpha}, \boldsymbol{x})$ or $(\boldsymbol{\alpha'}, \boldsymbol{x'})$. It gives $\widetilde{C}$, dk to $\mathcal{A}$ and simulates the marking oracle for $\mathcal{A}$ using its own oracle. At some point $\mathcal{A}$ outputs C^*.

1. $\mathcal{D}$ runs a test that finds an estimate $\widehat{\rho}$ for $\Pr_{x \leftarrow \mathsf{D}}[\widetilde{C}(x) = C^*(x)] - \frac{1}{\sqrt{2}}$ which is within $\rho/4$ of the actual probability with overwhelming probability. (This can be done the same way as in Test described in Figure. 2 with appropriate changes in parameters). If $\widehat{\rho} < \frac{3}{4}\rho$ then output 0.

2. If for majority of $i \in [\ell]$, $(\widetilde{C}(x_i^*) \neq C^*(x_i^*)) \vee (\widetilde{C}(\alpha_i^*) \neq C^*(\alpha_i^*))$, $\mathcal{D}$ outputs 1, otherwise 0.

If the distribution of $(\boldsymbol{x}^*, \boldsymbol{\alpha}^*)$ is that of $(\boldsymbol{x}, \boldsymbol{\alpha})$ then with probability δ the adversary $\mathcal{A}$ outputs a "winning program" C^* such that $C^* \cong_\varepsilon \widetilde{C}$ and $(\widetilde{C}(x_i^*) \neq C^*(x_i^*)) \vee (\widetilde{C}(\alpha_i^*) \neq C^*(\alpha_i^*))$ for a majority of i. Therefore, $\mathcal{D}$ outputs 1 with probability $\delta - \mathsf{negl}(\lambda)$.

If the distribution of $(\boldsymbol{\alpha}^*, \boldsymbol{x}^*)$ as that of $(\boldsymbol{\alpha'}, \boldsymbol{x'})$ then (1) either it is not the case that $C^* \cong_{1/\sqrt{2}+\rho/2} \widetilde{C}$ in which case the test fails and $\mathcal{D}$ outputs 0 with overwhelming probability, or (2) it is the case that $C^* \cong_{1/\sqrt{2}+\rho/2} \widetilde{C}$ in which case: each i we have

$$\Pr[(\widetilde{C}(x_i^*) = C^*(x_i^*)) \wedge (\widetilde{C}(\alpha_i^*) = C^*(\alpha_i^*))] \geq (1/\sqrt{2} + \rho/2)^2 = 1/2 + \Omega(\rho).$$

Moreover, since the events are independent for each i, the probability that equality holds for fewer than a majority of $i \in [\ell]$ is (by Chernoff bound) $2^{-\Omega(\rho^2 \ell)}$ which is negligible. Therefore, $\mathcal{D}$ outputs 1 with probability $\mathsf{negl}(\lambda)$. □

Proof of Lemma 4.12. This proof is essentially the same as that of Lemma 4.5. We explain only the important differences. In the message-embedding construction, for marked points, the output of a marked program is $F(K_3, x^{\mathsf{L}} \| \beta) \oplus \mathsf{msg}$. When moving from Hybrid 1 to Hybrid 2 we remove the marked points from the challenge circuit $\widetilde{C}$ which just outputs $F(K, x)$ on all inputs x, but we need to patch the detection key appropriately so it still detects $\widetilde{C}$ as marked and outputs the right message. To do that we modify the intermediate hybrids Hyb_1^j so as to use the following circuits for the challenge marked circuit $\widetilde{C}$ and the detection key component $\widetilde{D}_2$ respectively:

- $C^j[K, K_1, K_2, K_3, \mathsf{msg}]$ described in Figure 11
- $D_2^j[K, K_1, K_2, K_3, \mathsf{msg}]$ described in Figure 12.

Constants: pPRF keys K, K_1, K_2, K_3, and message msg.

Inputs: $x \in \mathsf{D}$

Code: Parse $x = x^{\mathsf{L}} \| x^{\mathsf{R}}$. Set $\alpha := F(K_1, x^{\mathsf{L}})$, $\beta := F(K, \alpha)$, $\hat{x}^{\mathsf{R}} := F(K_2, x^{\mathsf{L}} \| \beta)$.
If $x^{\mathsf{R}} = \hat{x}^{\mathsf{R}}$ and $x^{\mathsf{L}} \geq j$, then output $F(K_3, x^{\mathsf{L}} \| \beta) \oplus \mathsf{msg}$, else output $F(K, x)$.

Figure 11: Program $C^j[K, K_1, K_2, K_3, \mathsf{msg}]$

In Hybrid 2, the detection key $\widetilde{D}_2$ uses the circuit D_2^ρ. The important difference here is that program $D_2^j[K, K_1, K_2, K_3, \mathsf{msg}]$ used in the detection key must contain the challenge message msg embedded in the challenge program. To embed msg in the hybrid detection key, we must know the message before we give dk to the adversary. Thus, the scheme achieves selective-message security. Note that the adversary can adaptively query pPRF key $K^{(j)}$ and message msg_i to $\mathcal{MO}(mk, \cdot, \cdot)$. The rest of the proof is essentially the same where all the hybrids are defined analogously. □

Constants: pPRF keys K, K_1, K_2, K_3, and message msg.

Inputs: r, β

Code: Set $x^{\mathsf{L}} := \mathsf{G}(r)$, $\alpha := F(K_1, x^{\mathsf{L}})$, $x^{\mathsf{R}} := F(K_2, x^{\mathsf{L}} \| \beta)$.
If $x^{\mathsf{L}} \geq j$ or $\beta \neq F(K, \alpha)$, then set $y := F(K_3, x^{\mathsf{L}} \| \beta)$, else set $y := F(K, x^{\mathsf{L}} \| x^{\mathsf{R}}) \oplus \mathsf{msg}$.
Output (x^{R}, y).

Figure 12: Program $D_2^j[K, K_1, K_2, K_3, \mathsf{msg}]$

Full Security. We can achieve full security with an adaptively (rather than selectively) chosen embedded message by using the standard complexity leveraging technique where we guess the message msg before starting the security game. This loses a factor of $|\mathcal{M}|$ in the security reduction which is super-polynomial if the message space $\mathcal{M}$ is. However, since we already rely on complexity leveraging and assume sub-exponentially secure iO and PRFs to achieve public-detection, this does not impact our full result in any significant way.

Note on the approximation factor. Our scheme only achieves security for approximation factors $\varepsilon > 1/\sqrt{2}$ rather than $\varepsilon > 1/2$ (which we showed optimal in Section 3). This comes from the fact the detection procedure calls the circuit C' twice and requires both calls to return the correct value a majority of the time. In fact, there is a concrete attack on our message-embedding construction for an approximation factor ε in the range $\frac{1}{2} \leq \varepsilon \leq \frac{1}{\sqrt{2}}$. We leave it as an open problem if it is possible to construct a scheme that works for all $\varepsilon > 1/2$.

Improved Rate with ECC. We note that we can think of the above construction as relying on a naive *repetition code* over the alphabet $\Sigma = \{0,1\}^m$, where we just repeat the same message many times. We can modify the construction to allow for a larger message domain $\mathcal{M}$ than just Σ by using better codes, such as the Reed-Solomon code. In this variant we would set $\mathcal{M} = \Sigma^d$ for some polynomial d (without increasing the parameters n, m of the underlying pPRF). We would think of a message $\mathsf{msg} \in \Sigma^d$ as a degree $d-1$ polynomial p_{msg} over $\mathbb{F}_{2^m}$. For simplicity, let's assuming that $n = m$ and in this case we can modify the program in Figure. 10 so that, instead of outputting $F(K_3, x^{\mathsf{L}} \| \beta) \oplus \mathsf{msg}$ we would output $F(K_3, x^{\mathsf{L}} \| \beta) \oplus p_{\mathsf{msg}}(x)$. The detection procedure would apply the error-decoding procedure of the Reed-Solomon code to recover the message.

List Decoding. We note that our construction could also be modified to work for any $\varepsilon = 1/\mathrm{poly}(\lambda)$ approximation factors by (necessarily) relaxing the definition and allowing the detection procedure to output a (small) list of possible messages rather than a single message. For security, we would require that the correct message is contained in the list. For example, in our construction, instead of outputting the "majority value" msg such that $|\{i : \mathsf{msg} = \mathsf{msg}_i\}| > \ell/2$ we could output all $O(1/\varepsilon)$ values msg such that $|\{i : \mathsf{msg} = \mathsf{msg}_i\}| > \varepsilon\ell$ (or output $\mathsf{unmarked}$ if no such value exists). By signing the messages with a standard signature scheme, we could also ensure that the list of messages output by the detection procedure can only contain (in addition to the correct message) the messages that were embedded in some watermarked circuit by a previous call to the marking oracle.

5 Watermarking Other Cryptographic Primitives

We show how to use our watermarking scheme for pPRF to watermark other cryptographic primitives. Sahai and Waters proposed a public-key encryption (PKE) scheme and a signature (SIG) scheme based on pPRF and iO [SW14]. We briefly review their schemes since we apply our watermarking schemes to their PKE and SIG schemes.

The Sahai-Waters PKE scheme. Let G be a PRG.

Key Generation: Choose pPRF key $K \leftarrow \mathsf{Key}(1^\lambda)$ and generate an encryption program $\mathcal{E}$ in Figure 13. Output $(pk, dk) := (i\mathcal{O}(\mathcal{E}), K)$.

Encryption: For plaintext m, choose randomness $r \in \{0,1\}^\lambda$ and output $ct := i\mathcal{O}(\mathcal{E})(\mathsf{m}, r)$.

Decryption: Parse $ct = (c_1, c_2)$ and output $\mathsf{m}' := F(K, c_1) \oplus c_2$.

Constants: pPRF key K.

Inputs: plaintext $\mathsf{m} \in \{0,1\}^m$, randomness $r \in \{0,1\}^\lambda$

Code: Let $t := \mathsf{G}(r)$ and output $ct := (t, F(K,t) \oplus \mathsf{m})$.

Figure 13: Encryption Program $\mathcal{E}$

The Sahai-Waters SIG scheme. Let f be a one-way function.

Key Generation: Choose pPRF key $K \leftarrow \mathsf{Key}(1^\lambda)$ and generate a verification program $\mathcal{V}$ in Figure 14. Output $(vk, sk) := (i\mathcal{O}(\mathcal{V}), K)$.

Signing: For message m, output $\sigma := F(K, \mathsf{m})$.

Verification: Output what $i\mathcal{O}(\mathcal{V})(\mathsf{m}, \sigma)$ outputs.

Constants: pPRF key K.

Inputs: Signature σ, message m

Code: Check if $f(\sigma) = f(F(K, \mathsf{m}))$. If it holds, then output 1, otherwise 0.

Figure 14: Verification Program $\mathcal{V}$

As we reviewed, the decryption procedure of the PKE scheme and the signing procedure of the SIG scheme are just evaluations of a pPRF. Thus, we can watermark these primitives by using our watermarking schemes for pPRF. However, there is a subtle issue when we prove the non-removability of a marked decryption/signing program. In these settings, the adversary is given public values related to a pPRF key, that is, $i\mathcal{O}(\mathcal{E})$ or $i\mathcal{O}(\mathcal{V})$. We should show the PKE and SIG schemes are secure even if $i\mathcal{O}(\mathcal{E})$ and $i\mathcal{O}(\mathcal{V})$ are constructed from marked pPRF key $\widetilde{K} = \mathsf{Mark}(mk, K, \mathsf{msg})$ to reduce the non-removability of marked decryption/signing program to non-removability of pPRF. To achieve this and for modular analysis, we introduce an extension of pPRF, *puncturable marked PRF*.

Definition 5.1 (Puncturable Marked PRF) A puncturable marked PRF (pmPRF) scheme from pPRF $(\mathsf{Key}, F, \mathsf{Puncture})$ consists of watermarking scheme for pPRF $(\mathsf{Gen}, \mathsf{Mark}, \mathsf{Detect})$, a puncture-with-mark algorithm $\mathsf{PunctureM}$, and a evaluation-with-mark algorithm $\widetilde{F}$. They satisfy the following conditions. (we omit $\mathcal{F}$ in subscripts in algorithms.)

Functionality preserving under puncturing: For all polynomial size set $S \subseteq \{0,1\}^{n(\lambda)}$, for all $x \in \{0,1\}^{n(\lambda)} \setminus S$, for all $\mathsf{msg} \in \mathcal{M}$, it holds that

$$\Pr\left[\widetilde{F}(\widetilde{K}, x) = \widetilde{F}(\widetilde{K}\{S\}, x) : \begin{array}{c} K \leftarrow \mathsf{Key}(1^\lambda), (dk, mk) \leftarrow \mathsf{Gen}(1^\lambda), \\ \widetilde{K} := \mathsf{Mark}(mk, K, \mathsf{msg}), \\ \widetilde{K}\{S\} := \mathsf{PunctureM}(mk, K, \mathsf{msg}, S) \end{array}\right] = 1.$$

Pseudorandom at punctured points: For all polynomial size set $S \subseteq \{0,1\}^{n(\lambda)}$ and $\mathsf{msg} \in \mathcal{M}$, it holds that for all PPT adversary $\mathcal{A}$,

$$\left|\Pr[\mathcal{A}(\widetilde{K}\{S\}, \widetilde{F}(\widetilde{K}, S)) = 1] - \Pr[\mathcal{A}(\widetilde{K}\{S\}, U_{m(\lambda)\cdot|S|}) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $(dk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$, $K \leftarrow \mathsf{Key}(1^\lambda)$, $\widetilde{K} := \mathsf{Mark}(mk, K, \mathsf{msg})$, $S = \{x_1, \ldots, x_k\}$, $\widetilde{K}\{S\} := \mathsf{PunctureM}(mk, K, \mathsf{msg}, S)$, $\widetilde{F}(\widetilde{K}, S)$ denotes multiple values $(\widetilde{F}(\widetilde{K}, x_1), \ldots, \widetilde{F}(\widetilde{K}, x_k))$, and U_ℓ denotes the uniform distribution over ℓ bits.

We can easily construct pmPRF from our watermarking schemes by replacing pPRF key K in a marked program with a punctured key $K\{S\}$. For example, $\mathsf{PunctureM}(mk, K, \mathsf{marked}, S)$ can compute a punctured and marked pPRF key $\widetilde{K}\{S\} = C[K\{S\}, \boldsymbol{x}, \boldsymbol{y}]$ in the basic construction if we have $mk = wk = (\boldsymbol{x}, \boldsymbol{y})$ and K. Although we cannot compute $\widetilde{K}\{S\}$ from $K\{S\}$ even if we use mk, it does not cause any problem since we use $\widetilde{K}\{S\}$ only in security proofs. Similarly, we can construct pmPRF from the constructions with public-key detection in Section 4.2.

Security of Sahai-Waters PKE and SIG based on pmPRF. Even if we replace pPRF $F(K, \cdot)$ in Sahai-Waters PKE and SIG with pmPRF $\widetilde{F}(\widetilde{K}, \cdot)$, they remain secure since marked pPRFs have statistical correctness and we can puncture marked pPRF keys as we observed. Thus, the original proofs by Sahai and Waters still work.

Constants: Marked pPRF key $\widetilde{K}$.

Inputs: $ct = (c_1, c_2)$

Code: Output $\widetilde{F}(\widetilde{K}, c_1) \oplus c_2$.

Figure 15: Marked Decryption Program $\widetilde{\mathcal{D}}$

Constants: Marked pPRF key $\widetilde{K}$.

Inputs: m

Code: Output $\widetilde{F}(\widetilde{K}, \mathsf{m})$.

Figure 16: Marked Signing Program $\widetilde{\mathcal{S}}$

On Non-Removability of Marked Decryption and Signing Programs. If there exists adversary $\mathcal{A}$ of the non-removability security game for PKE, then we can construct adversary $\mathcal{B}$ of the non-removability security game for pPRF. We focus on the PKE case and the SIG case is similarly proved. $\mathcal{B}$ is given $\widetilde{K} = \mathsf{Mark}(mk, K, \mathsf{msg})$. $\mathcal{B}$ gives $\mathcal{A}$ program $\widetilde{\mathcal{D}}$ described in Figure 15 and a public key $i\mathcal{O}(\widetilde{\mathcal{E}})$ where $\widetilde{\mathcal{E}}$ is the same as $\mathcal{E}$ except that K is replaced with $\widetilde{K} = \mathsf{Mark}(mk, K, \mathsf{msg})$. Giving $i\mathcal{O}(\widetilde{\mathcal{E}})$ does not give the adversary any more power to break the non-removability since it is constructed from the marked pPRF key.

Eventually, $\mathcal{A}$ outputs un-marked program C^* such that $C^* \cong_\varepsilon \widetilde{\mathcal{D}}$. Thus, $\mathcal{B}$ outputs a program that computes $C^*(c_1, c_2) \oplus c_2$ for input (c_1, c_2) as an un-marked pPRF program. This is a valid attack since we assume $\mathcal{A}$ breaks the non-removability of the marked decryption program $\widetilde{\mathcal{D}}$.

6 Conclusions

We have shown how to watermark various cryptographic programs such as PRFs, public-key encryption and signatures. Our watermarking schemes have a public-key detection procedure and achieve security in the presence of the marking oracle. The attacker cannot remove the mark by coming up with any program that functionally approximates the marked program where we can achieve an approximation factor of $\varepsilon = 1/\mathrm{poly}$ in the case of "mark-embedding" schemes and $\varepsilon = \frac{1}{\sqrt{2}} + 1/\mathrm{poly}$ in the case of "message-embedding" schemes. Several interesting open questions remain.

Firstly, it would be interesting to further explore the connection between obfuscation and watermarking and to see if some form of obfuscation is *necessary* to achieve watermarking or if one can come up with constructions that avoid obfuscation. Less ambitiously, one could hope to at least avoid complexity leveraging in the constructions of watermarking.

Secondly, it would be interesting to achieve a fully public-key watermarking construction where both the marking and the detection procedure only use public keys. This kind of watermarking appears plausible and we can even come up with a heuristic construction by taking our watermarking scheme with a secret-key marking procedure and releasing an obfuscated marking procedure as a public marking key. However, proving such a scheme secure by only relying on iO appears difficult.

Thirdly, it would be interesting to achieve a message-embedding construction with the optimal approximation factor $\varepsilon > 1/2$ rather than $\varepsilon > 1/\sqrt{2}$.

Lastly, it would be interesting to come up with watermarking schemes for richer classes of programs as well as more applications for watermarking cryptographic programs.

References

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 1–18, 2001.

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

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 501–519, 2014. Full version available from http://eprint.iacr.org/2013/401.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 719–737, 2012.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, pages 280–300, 2013. Full version available from http://eprint.iacr.org/2013/352.

[CHN+15] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. *IACR Cryptology ePrint Archive*, 2015:1096, 2015.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49, 2013. Full version available from http://eprint.iacr.org/2013/451.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[HMW07] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, pages 362–382, 2007.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 669–684, 2013. Full version available from http://eprint.iacr.org/2013/379.

[Nis13] Ryo Nishimaki. How to watermark cryptographic functions. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 111–125, 2013. Full version available from http://eprint.iacr.org/2014/472.

[NSS99] David Naccache, Adi Shamir, and Julien P. Stern. How to copyright a function? In *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings*, pages 188–196, 1999.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484, 2014. Full version available from http://eprint.iacr.org/2013/454.

[YF11] Maki Yoshida and Toru Fujiwara. Toward digital watermarking for cryptographic data. *IEICE Transactions*, 94-A(1):270–272, 2011.