Watermarking Cryptographic Capabilities*

Aloni Cohen† Justin Holmgren‡ Ryo Nishimaki§ Vinod Vaikuntanathan¶
Daniel Wichs‖

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 F, we create a marked program $\widetilde{C}$ that evaluates $F(\cdot)$. 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, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang (CRYPTO'01 and Journal of 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 also significantly extend the impossibility results to our relaxed setting.

Our watermarking schemes are *public key*, meaning that we use a secret marking key to embed marks in programs, and a public detection key that allows anyone to detect marks in programs. Our schemes are secure against *chosen program attacks* where the adversary is given 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, in contrast to the prior works (Nishimaki, EUROCRYPT '13).

*This work is a merged version of [NW15] and [CHV15] with additional results. This work was sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contract number W911NF-15-C-0226. This work was done in part while the first two authors were visiting the Weizmann Institute of Science, and in part while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant CNS-1523467.

†MIT, aloni@mit.edu. Research supported in part by the NSF Graduate Student Fellowship.

‡MIT, holmgren@mit.edu. Research supported in part by NSF Frontier CNS-1413920.

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

¶MIT CSAIL, vinodv@csail.mit.edu. Research supported in part by DARPA Safeware grant, NSF grants CNS-1350619 and CNS-1414119, Alfred P. Sloan Research Fellowship, Microsoft Faculty Fellowship, the NEC Corporation, and a Steven and Renee Finn Career Development Chair from MIT.

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

Contents

1 Introduction

Digital watermarking enables us to embed some special information called a *mark* into digital objects such as images, movies, music files, or programs. 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. (We briefly survey some of these works in Section 2.7). Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang [BGI+01, BGI+12], in their seminal work that laid the mathematical foundations of program obfuscation, also proposed definitions for program watermarking. Unfortunately, their results were all negative, showing that certain definitions of watermarking are impossible to achieve. The work of Hopper, Molnar and Wagner [HMW07] proposes general and rigorous definitions for watermarking schemes, and explores in depth connections between the definitions, but does not provide any actual constructions.

Watermarking Programs. Our first contribution is to define the notion of *public-key watermarking*, building on the work of Hopper, Molnar and Wagner [HMW07] who introduced a secret-key definition. We speak of a watermarking scheme for a circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ where each $\mathcal{C}_\lambda$ is a set of circuits. A watermarking scheme for $\mathcal{C}$ consists of procedures $\mathsf{Mark}(mk,\cdot)$ and $\mathsf{Extract}(ek,\cdot)$ with a *secret marking key* mk and a *public extraction key* ek. Given a circuit C, the marking procedure $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C)$ creates a marked circuit $\widetilde{C}$ that evaluates C. Although we will see that we cannot achieve perfect correctness, in which $\widetilde{C}(x) = C(x)$ for all inputs x, we will be able to achieve *statistical correctness* where we allow a negligible error probability. The extraction procedure $\mathsf{Extract}(ek, C^*)$ outputs either that the circuit is marked or $\mathsf{unmarked}$.

For security, we consider a game where a challenger chooses a *random circuit* $C \leftarrow \mathcal{C}_\lambda$ and gives the adversary the marked circuit $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C)$. Intuitively, we require that the adversary cannot come up with *any* circuit that correctly evaluates C but does not have the mark embedded in it. This property is called *unremovability*. Following [HMW07] and adapting it to the public-key setting, we require that unremovability holds against *chosen circuit attackers*, namely adversaries that have oracle access to $\mathsf{Mark}(mk,\cdot)$.

More precisely, the adversary produces a circuit C^* and we insist that either:

(a) $\mathsf{Extract}$ correctly detects that the circuit is marked by outputting $\mathsf{marked} \leftarrow \mathsf{Extract}(ek, C^*)$; or

(b) The circuit C^* *does not even approximately* compute C, meaning that $C^*(x) = C(x)$ on at most an ε fraction of the inputs x.

The parameter ε is called the "approximation factor" and we can set it to some small constant or even to any $1/\mathrm{poly}$ fraction. (The smaller the ε, the better the security guarantee). During the attack, the adversary is also given the public extraction key ek and access to the marking oracle $\mathsf{Mark}(mk,\cdot)$ that he can query on arbitrary circuits of his choice (even ones that are not in $\mathcal{C}_\lambda$). At this point, it is prudent to note that the very first idea that comes to mind, namely signing the circuit C using mk, is not a particularly good watermarking strategy as the adversary can simply strip off the signature leaving a perfectly functional circuit.

We call the above type of watermarking "messageless" to denote that it only distinguishes between marked and unmarked circuits. We also consider a stronger version called "message-embedding" watermarking where the marking procedure can be used to embed an arbitrary message into the circuit and the extraction procedure should recover the message. Similar to the above, the adversary's goal is to force the extraction procedure to recover a different message. (We refer the reader to Section 4 for formal definitions).

Why Cryptographic Programs? In this work, we focus on watermarking circuits that are cryptographic in nature, such as circuits evaluating a pseudorandom function (PRF) or implementing a signing or decryption procedure. One could reasonably ask: *why cryptographic programs*?

First, we observe that in the security definition for watermarking, the challenge circuit C has to be unknown to the adversary. For, if not, the adversary has a trivial watermark removing strategy: given the marked circuit $\widetilde{C}$, simply output C as the mark-removed circuit. Since C is an arbitrary program, it is very likely to be unmarked; on the other hand, C is (approximately) equivalent $\widetilde{C}$ in functionality.[1] Thus, it is natural for the challenger to pick C from a distribution with high min-entropy (In this work, for simplicity, we consider picking circuits uniformly at random from $\mathcal{C}_\lambda$).

Secondly, we observe that circuit families that are exactly learnable are not watermarkable. This is because the adversary can simply invoke $\widetilde{C}$ as a black box and recover a description of the original circuit C (or an equivalent version thereof) which is again very likely to be unmarked.

This naturally leads us to consider families of circuits where random circuits from the family are not exactly learnable, canonical examples of which are cryptographic programs: pseudo-random functions, signing algorithms and decryption algorithms. Jumping ahead, we remark that unlearnability is a necessary but not sufficient condition for being able to watermark a family of circuit. Indeed, we show families of pseudo-random functions that, despite being strongly unlearnable, cannot be watermarked even with approximate correctness.

That said, we regard the question of coming up with meaningful definitions and constructions of watermarking for general circuits (and even families of evasive circuits) as a challenging open question arising from this work.

Watermarking Cryptographic Programs: An Application. To further highlight the usefulness of watermarking cryptographic functions, we describe an application of watermarking pseudorandom functions. However, we emphasize that the concept should have broader applicability beyond this example.

Consider an automobile manufacturer that wants to put electronic locks on its cars; the car contains a PRF F and can only be opened by running an identification protocol where it chooses a random input x and the user must respond with $F(x)$. 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 $F(\cdot)$ 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 a new piece of software that would still open the car while removing information about the original owner.

Impossibility of Watermarking? 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 appears significantly weaker than the definitions we consider in this work (it is in the symmetric-key setting with no marking/detection oracles given to the adversary), *but* requires 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 (or even extractability

[1] One can attempt to get around this issue by requiring that the program output by the watermark remover should be distinct from C and $\widetilde{C}$. However, it is also easy to defeat these definitions by asking the watermarked remover to output an indistinguishability obfuscation of C.

obfuscation or VGB) would be sufficient and what type of watermarking security can be achieved with this approach. Nevertheless, this idea serves as the starting point of our work.

1.1 Our Results

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 result described above, our definition allows for statistical rather than perfect correctness. That is, for every circuit $C \in \mathcal{C}_\lambda$ and every input x,

$$\Pr[\widetilde{C}(x) \neq C(x) \;\mid\; \widetilde{C} \leftarrow \mathsf{Mark}(mk, C)] \leq \mathsf{negl}(\lambda)$$

where the probability is over the choice of the keys and the coin tosses of the Mark algorithm. We call this *strong approximate correctness*.

This seemingly small relaxation allows us to circumvent the impossibility results and show algorithms to watermark large classes of pseudo-random functions, signature algorithms and decryption algorithms. Our main technical contribution is a method of watermarking any family of puncturable PRFs.[2] Our scheme has a public-key extraction procedure and achieves security in the presence of a marking oracle. We get a messageless scheme that allows for any $\varepsilon = 1/\mathrm{poly}(\lambda)$ approximation factor and a message-embedding scheme that allows for approximation factors $\varepsilon = 1/2 + 1/\mathrm{poly}(\lambda)$. In the case of message-embedding constructions, we show that there is an inherent lower bound of $\varepsilon = 1/2$. Both schemes rely on (polynomially secure) indistinguishability obfuscation (iO).

Theorem 1.1 (Informal) *Assuming indistinguishability obfuscation and injective one-way functions, there is a watermarking scheme that is secure against chosen circuit attacks for any family of puncturable PRFs.*

We then extend this to watermarking other cryptographic primitives such as the decryption procedure of a public-key encryption scheme, and 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 evaluations [SW14]. (In contrast to our PRF result where we watermark any punctured PRF, here we design special watermarkable signature schemes and decryption algorithms).

Theorem 1.2 (Informal) *Assuming indistinguishability obfuscation and injective one-way functions, there are signature and decryption algorithms that can be watermarked with chosen circuit security.*

Theorem 1.1 and 1.2 show that relaxing the correctness requirement to strong approximate correctness allows us to watermark any family of puncturable PRFs, and certain families of signature and decryption algorithms. A natural question is whether one can watermark arbitrary PRF, signature and decryption circuits. We show impossibility results matching our constructions by demonstrating families of PRFs, signature and decryption algorithms that cannot be watermarked. We call such schemes *waterproof*.

We start by observing that learnable functions are waterproof, simply because an adversary can learn a canonical representation of the function given any program (even any oracle) that computes the function. Indeed, it is sufficient for the function family to be *non black-box learnable*. That is, the adversary should be able to use any program that computes the function to extract a canonical representation. Such function families were defined in the work of Barak et al. [BGI+01] and are called *unobfuscatable functions*. Indeed, [BGI+01, BGI+12] show PRFs, signature and decryption algorithms that are strongly unobfuscatable – that is, an adversary can extract the canonical representation even given a program that only computes a function with strong approximate correctness. This immediately gives us waterproof PRFs, signature and decryption algorithms. (See Section 9 for more details.)

[2] Puncturable pseudo-random functions (pPRFs) [BW13a, BGI14a, KPTZ13a] are PRFs where the owner of the key K can produce a punctured key Kx that allows computation of the PRF on all inputs $y \neq x$. Moreover, given the punctured key, $\mathsf{PRF}_K(x)$ is pseudorandom. Puncturable PRFs can be constructed from one-way functions [BW13a, BGI14a, KPTZ13a] or more efficiently, from several number-theoretic assumptions. [BLMR13, BV15, BFP+15].

Theorem 1.3 (Informal) *Assuming the existence of one-way functions, there are waterproof PRFs and signature and decryption algorithms, even if: (a) we only require symmetric-key watermarking; and (b) we only require unremovability against stand-alone adversaries that do not have access to* Mark *or* Extract *oracles.*

We continue this line of thought and ask if we can further weaken the correctness requirement and overcome this impossibility result. Namely, we consider a weak approximate correctness requirement which states that the marked program $\widetilde{C}$ agrees with the original program C on most inputs. (In contrast to strong approximate correctness, here $\widetilde{C}$ can always make a mistake on some fixed set of inputs). We show that even this relaxation does not help. Our proof of this result involves constructing new types of *robust* unobfuscatable PRFs. (See Section B for more details).

Theorem 1.4 (Informal) *Assuming the existence of one-way functions, there are waterproof PRFs even under weak approximate correctness (and even with relaxations (a) and (b) as in Theorem 1.3).*

2 Overview of Our Techniques

2.1 Simplification: Token-Based Watermarking

Although our full watermarking scheme relies on indistinguishability obfuscation (iO), our main technical insights are largely unrelated to obfuscation. In order to elucidate our techniques clearly without getting entangled in details of iO, for the purposes of this introduction we consider a simplified model of watermarking that we call *token-based* watermarking. We treat watermarked programs $\widetilde{C} \leftarrow \mathsf{Mark}(mk, \cdots)$ as tamper-proof hardware tokens which the adversary can only access as a black box.[3] The adversary can arbitrarily compose hardware tokens $\widetilde{C}_1, \ldots, \widetilde{C}_q$ and create a new token $C^* = C^*[\widetilde{C}_1, \ldots, \widetilde{C}_q]$ that has oracle access to the tokens $\widetilde{C}_i$ embedded inside of it. More formally, we can think of C^* as an oracle circuit with oracle-gates to $\widetilde{C}_1, \ldots, \widetilde{C}_q$. The extraction procedure $\mathsf{Extract}(ek, C^*)$ will also treat any such token C^* as a black box. The goal of the adversary is to create a token C^* which functionally approximates the challenge watermarked program $\widetilde{C}$ but on which the extraction procedure fails to recover the correct embedded message. Most of the interesting aspects of constructing watermarking schemes already come up in the token-based setting.[4] However, the constructions in the token-based setting become simpler and do not rely on obfuscation. Therefore, we view it as a useful stepping stone to building intuition for our full results where the adversary gets the complete code of the watermarked programs.

2.2 A High Level Approach

At a high level, to watermark a PRF $F : \{0,1\}^n \to \{0,1\}^m$, we create a token $\widetilde{C}$ that evaluates F correctly on almost all inputs x, except for some special set of "marked-points" $\mathcal{X} \subseteq \{0,1\}^n$ which have negligible density in $\{0,1\}^n$. On the marked points, the watermarked program outputs specially constructed incorrect values that allow the extraction procedure to recover the embedded message. We will ensure that marked points are indistignuishable to the adversary from random inputs. Therefore, the adversary cannot create a new token C^* that agrees with $\widetilde{C}$ on a large fraction of random inputs (i.e., approximates F) but disagrees with $\widetilde{C}$ on sufficiently many marked points so as to cause the extraction procedure to fail.

[3] Alternately, one can think of this setting as assuming that $\widetilde{C}$ is obfuscated with an "ideal obfuscation" scheme. However, since software-only ideal obfuscation schemes don't exist, it's more accurate to think of $\widetilde{C}$ as a physical hardware token.

[4] For example, it's immediately clear that *exact* watermarking, where the marked program $\widetilde{C}$ is functionally equivalent to the original program C, is impossible in this setting since in that case the extraction procedure cannot distinguish between black-box access to the original unmarked program C and the marked program $\widetilde{C}$.

2.3 A Simple Scheme with Weak Security

We start by considering a weak notion of token-based watermarking security, where both the marking key mk and the extraction key ek are secret and the adversary does not have access to either the marking oracle $\mathsf{Mark}(mk, \cdot)$ or the extraction oracle $\mathsf{Extract}(ek, \cdot)$. We also consider a messageless scheme where programs can only be marked or unmarked. In particular, in the security game the adversary gets a single marked token $\widetilde{C} \leftarrow \mathsf{Mark}(mk, F)$ as a challenge, where $F : \{0,1\}^n \to \{0,1\}^m$ is chosen at random from a PRF family $F \leftarrow \mathcal{F}$ (and n, m are super-logarithmic). The adversary's goal is to come up with some new token $C^* = C^*[\widetilde{C}]$ that approximately evaluates F but on which the extraction procedure fails to detect that the program is marked: $\mathsf{Extract}(ek, C^*) = \mathsf{unmarked}$.

This can be easily achieved as follows. Choose a polynomial set of ℓ "marked-points" $\mathcal{X} = \{x_1, \ldots, x_\ell\} \subseteq \{0,1\}^n$ uniformly at random with corresponding random outputs $y_1, \ldots, y_\ell \leftarrow \{0,1\}^m$. Set $mk = ek = (x_1, \ldots, x_\ell, y_1, \ldots y_\ell)$. To mark a PRF F, the marking procedure $\widetilde{C} \leftarrow \mathsf{Mark}(mk, F)$ outputs a token $\widetilde{C}$ that contains $x_1, \ldots, x_\ell, y_1, \ldots, y_\ell$ hard-coded and, on input x, if $x = x_i$ for some $i \in [\ell]$ it outputs y_i else it outputs $F(x)$. The extraction procedure $\mathsf{Extract}(ek, C^*)$ tests if on *at least one* of the ℓ marked points $x_i \in \mathcal{X}$ the program evaluates to $C^*(x_i) = y_i$. If so, it outputs that the program is marked, and otherwise outputs unmarked.

To prove that the above scheme is secure, we notice that an adversary that gets black-box access to a token $\widetilde{C} \leftarrow \mathsf{Mark}(mk, F)$ for a random unknown $F \leftarrow \mathcal{F}$ cannot distinguish between the marked points $\mathcal{X} = \{x_1, \ldots, x_\ell\}$ and ℓ uniformly random and independent inputs without breaking PRF security. This implies that the adversary cannot come up with a new token $C^* = C^*[\widetilde{C}]$ such that $C^*(x) = \widetilde{C}(x)$ is "correct" on a large fraction of inputs $x \in \{0,1\}^n$, but $C^*(x_i) \neq \widetilde{C}(x_i) = y_i$ for all marked points $x_i \in \mathcal{X}$, as this would imply distinguishing between random points and marked points. More precisely, by setting $\ell = \Omega(\lambda/\varepsilon)$ where λ is the security parameter, we ensure that if the adversary creates any token $C^* = C^*[\widetilde{C}]$ that agrees with the marked token $\widetilde{C}$ on even an ε-fraction of inputs $x \in \{0,1\}^n$, then $C^*(x_i) = y_i$ for at least one marked point $x_i \in \mathcal{X}$ with overwhelming probability $1 - (1-\varepsilon)^\ell$ and therefore $\mathsf{Extract}(ek, C^*) = \mathsf{marked}$ as desired.

2.4 Challenges in Allowing Mark/Extract Oracles

Unfortunately, the above scheme becomes completely insecure if the adversary has access to *either* a marking oracle $\mathsf{Mark}(mk, \cdot)$ or the extraction oracle $\mathsf{Extract}(ek, \cdot)$, let alone if the extraction key ek is made public. Let us describe the attacks.

Attack using the extraction oracle. If the adversary gets the challenge marked program $\widetilde{C} \leftarrow \mathsf{Mark}(mk, F)$ as a token, he can create new tokens $C' = C'[\widetilde{C}]$ such that $C'(x) = \widetilde{C}(x)$ only for x satisfying $P(x) = 1$ where P is some predicate. By querying the extraction oracle $\mathsf{Extract}(ek, C')$ to see if such tokens are deemed marked or unmarked, the adversary will learn whether there exists some marked point x_i with $P(x_i) = 1$. By choosing such predicates carefully, these queries can completely reveal the marked points. [5]

Attack using the marking oracle. Assume the adversary makes just one call to the marking oracle with an arbitrary known PRF function $F' \in \mathcal{F}$ and gets back a token $\widetilde{C}' \leftarrow \mathsf{Mark}(mk, F')$. In addition, the adversary gets a challenge token $\widetilde{C} \leftarrow \mathsf{Mark}(mk, F)$ corresponding to a random unknown PRF $F \leftarrow \mathcal{F}$. The adversary

[5] For example, a concrete instantiation of the above attack uses predicates of the form $P_w(x) = 1$ iff $x[1, \ldots, |w|] = w$ for some $w \in \{0,1\}^*$ (i.e., the first $|w|$ bits of x match w). By starting with w being the empty string, the adversary can iteratively add a bit to learn if there exists some marked point x_i with $P_{w||b}(x_i) = 1$ for $b \in \{0,1\}$. Whenever the above occurs for exactly one choice of $b \in \{0,1\}$, the adversary extends $w := w||b$ and continues to the next iteration. If this happens for both choices of $b \in \{0,1\}$ then the adversary branches the above process and continues down both paths for $w := w||0$ and $w := w||1$. Since there are ℓ marked points, this process will only branch ℓ times and the adversary will eventually recover all of the points $\mathcal{X} = \{x_1, \ldots, x_\ell\}$. Once the adversary learns $\mathcal{X}$, he can create a circuit $C^*[\widetilde{C}]$ such that $C^*(x) = \widetilde{C}(x)$ for any $x \notin \mathcal{X}$ and otherwise $C^*(x)$ outputs some incorrect value (e.g., an independent pseudorandom output). The circuit C^* closely approximates $\widetilde{C}$ (on all but a negligible fraction of inputs) yet the extraction procedure fails to detect C^* as marked.

can easily remove the mark by creating a new token $C^*[\widetilde{C}', \widetilde{C}]$ that gets oracle access to $\widetilde{C}'$ and $\widetilde{C}$ and does the following: on input x, if $\widetilde{C}'(x) = F'(x)$ then output $\widetilde{C}(x)$ else output some incorrect value (e.g., an independent pseudorandom output). The circuit C^* only differs from $\widetilde{C}$ on the marked points $x_i \in \mathcal{X}$ and therefore closely approximates $\widetilde{C}$ on all but a negligible fraction of inputs. However, the extraction procedure will fail to detect C^* as marked.

2.5 Toward a Fully Secure Token-Based Scheme

We now outline the main ideas for how to thwart the above attacks and get a token-based watermarking scheme with a public extraction key ek and with security in the presence of a marking oracle $\mathsf{Mark}(mk, \cdot)$.

Overview. Our first idea is to make the set of marked points $\mathcal{X} \subseteq \{0,1\}^n$ super-polynomial, yet still of negligible density inside of $\{0,1\}^n$. This will allow us to thwart the attack using an extraction oracle and even make the extraction key ek public. In particular, we ensure that even given the extraction key ek, which can be used to sample random marked points $x \leftarrow \mathcal{X}$, the adversary still cannot distinguish such points from uniformly random inputs. Thwarting the marking oracle attack is more difficult. We need to ensure that the set of marked points $\mathcal{X}_F$ is different for each PRF F that we will mark so that, even if the adversary can test if a point belongs to $\mathcal{X}_{F_i}$ for various PRFs F_i that were queried to the marking oracle, the marked points $\mathcal{X}_F$ for the challenge (unknown) PRF F will remain indistinguishable from uniform. However, this creates a difficulty since the extraction procedure $\mathsf{Extract}(ek, \widetilde{C})$ must test the marked program $\widetilde{C}$ on the correct set of marked points $\mathcal{X}_F$ without knowing the function F from which $\widetilde{C}$ was created. We solve this by ensuring that one can find a marked point for the function F by querying F. In particular, the extraction procedure first queries $\widetilde{C}(z)$ on some special (pseudo-random) "find point" z and then, assuming $\widetilde{C}(z) = F(z)$, uses the output $\widetilde{C}(z)$ to sample a marked point $x \leftarrow \mathcal{X}_F$.

A Concrete Scheme. Let $\mathcal{F}$ be a PRF family consisting of functions $F : \{0,1\}^n \to \{0,1\}^\lambda$ where λ is the security parameter and n is sufficiently large. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CCA secure public-key encryption scheme with *pseudorandom ciphertexts* having message space $\{0,1\}^{3\lambda}$ and ciphertext space $\{0,1\}^n$.[6] Let $G : \{0,1\}^\lambda \to \{0,1\}^n$ be a PRG.

Keys: We sample a key pair for the encryption scheme $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and define the marking/extraction key mk, ek to be the secret/public key respectively: $mk = sk, ek = pk$.

Marking: For a PRF $F \in \mathcal{F}$, we define the set of "marked points" as:

$$\mathcal{X}_F = \{x \in \{0,1\}^n : \mathsf{Dec}_{sk}(x) = (a||b||c) \in \{0,1\}^{3\lambda}, F(G(a)) = b\}.$$

To mark a PRF F the procedure $\widetilde{C} \leftarrow \mathsf{Mark}(mk, F)$ creates a token $\widetilde{C}$ defined as follows:

Hard-Coded Constants: F, sk.
Input: $x \in \{0,1\}^n$

1. Try to decrypt $a||b||c \leftarrow \mathsf{Dec}_{sk}(x)$ with $a, b, c \in \{0,1\}^\lambda$.
2. If decryption succeeds and $F(G(a)) = b$ output c. // $x \in \mathcal{X}_F$ is a marked point
3. Otherwise output $F(x)$.

Extraction: The extraction procedure $\mathsf{Extract}(ek, C^*)$ repeats the following ℓ times:

- Choose random $a, c \leftarrow \{0,1\}^\lambda$ and let $z = G(a)$ and $b = C^*(z)$. // z is a find point
- Choose $x \leftarrow \mathsf{Enc}_{pk}(a||b||c)$ and if $C^*(x) = c$ then output marked. // if $b = F(z)$ then $x \in \mathcal{X}_F$.

[6] For simplicity, we assume ciphertexts are pseudorandom in $\{0,1\}^n$. For our full construction we will construct such schemes with additional puncturability properties using PRFs and iO. However, we can generalize this to other domains beside $\{0,1\}^n$ and, in the token-based setting, we could then rely on standard constructions of CCA secure encryption such as e.g., Cramer-Shoup [CS03].

If all ℓ iterations fail, output $\mathsf{unmarked}$.

Intuitively, the construction relies on the fact that the marked program $\widetilde{C}$ can recognize marked points by using the decryption key. On the other hand the extraction procedure can find the marked points for a function F given a circuit C^* that approximates F by querying $C^*(z)$ where $z = G(a)$ is a "find point". If the circuit answers correctly on z so that $F(z) = C^*(z) = b$ then the extraction procedure will be able to correctly sample a marked point $x \leftarrow \mathsf{Enc}_{pk}(a||b||c)$.

Security Analysis Overview. For the security analysis, consider an adversary that gets an extraction key $ek = pk$ and makes q queries to the marking oracle with arbitrary PRF functions $F_i \in \mathcal{F}$ and gets back marked tokens $\widetilde{C}_i \leftarrow \mathsf{Mark}(mk, F_i)$. The adversary then gets a challenge marked token $\widetilde{C} \leftarrow \mathsf{Mark}(mk, F)$ for a random unknown PRF $F \leftarrow \mathcal{F}$. The adversary can only query the tokens as a black box.

Firstly, we claim that even given the above view, the adversary cannot distinguish between getting random find/mark points z, x and completely random values z', x':

$$(\mathsf{view}, z, x) \approx (\mathsf{view}, z', x') \quad : a, c \leftarrow \{0,1\}^n, z = G(a), b = F(z), x \leftarrow \mathsf{Enc}_{pk}(a||b||c), z', x' \leftarrow \{0,1\}^n.$$

To show this, we can first rely on CCA security to switch x to a uniformly random x'. This is because black-box access to the marked tokens $\widetilde{C}_i$ can be simulated by a CCA oracle that never decrypts x (it's unlikely that $F(z) = F_i(z)$ for some i, and therefore x is not a marked point for the queried functions F_i with overwhelming probability) while the challenge program $\widetilde{C}$ outputs $\widetilde{C}(x) = c$ but this is indistinguishable from $\widetilde{C}(x') = F(x')$ since both outcomes look random. We then rely on PRG security to switch z to uniform.

Secondly, we claim that the above "indistinguishability" property immediately implies "unremovability". In particular, if the adversary manages to produce a token C^* that ε-approximates the challenge program $\widetilde{C}$ then, for a random $z', x' \leftarrow \{0,1\}^n$ the probability that $C^*(z') = \widetilde{C}(z')$ and $C^*(x') = \widetilde{C}(x')$ is at least ε^2. Therefore, the same must hold (up to a negligible difference) when x, z are a random find/marked point. This means that each iteration of the extraction procedure outputs marked with probability at least ε^2 and therefore the probability that none of the iterations outputs marked is at most $(1-\varepsilon^2)^\ell$ which is negligible as long as $\ell = \Omega(\lambda/\varepsilon^2)$.

This analysis only provides *lunch-time security* where the adversary can query the marking oracle only prior to seeing the challenge program $\widetilde{C}$. This is because we relied on the fact that, with overwhelming probability, none of the queried functions F_i will satisfy $F_i(z) = F(z)$ where F is the challenge PRF. This may not hold in a stronger security model where the adversary can adaptively query the marking oracle with function F_i after seeing the watermarked version $\widetilde{C}$ of the challenge PRF F. However, we can salvage the same analysis and make it hold in the stronger model if we assume the PRF family satisfies an additional *injective* property, meaning that when $F \neq F'$ then $F(z) \neq F'(z)$ for all inputs z. We can construct such PRFs under natural assumptions such as DDH or LWE.

Embedding a Message. We can extend the above construction to embed a message in the marked program. We do so by ensuring that the outputs of the marked circuit on the marked points x encode information about the message msg, which can then be recovered by the extraction procedure. In particular, instead of simply having the marked circuit output the value c encrypted in the marked point x, we make it output $c \oplus \mathsf{msg}$ where msg is message we wish to embed. The extraction procedure can work as above but in each iteration $i = 1, \ldots, \ell$ it recovers a candidate message msg_i. We simply test if there is a message which is recovered in a majority of the iterations. If so we output it, and otherwise we output $\mathsf{unmarked}$. A naive implementation of this approach would only work for an approximation factor $\varepsilon > 1/\sqrt{2}$ since only in that case could we expect that C^* answers correctly on both the find point and the marked point simultaneously with probability $> 1/2$ so as to get a correct majority. We show how to tweak the above approach to make it work for optimal approximation factor $\varepsilon > 1/2$ by testing C^* on many marked points for each find point and taking a majority-of-majorities.

2.6 Using Indistinguishability Obfuscation.

Lastly, we briefly mention our techniques for moving beyond token-based watermarking. On a high level, we can simply obfuscate the watermarked programs $\widetilde{C}$, instead of thinking of them as hardware tokens. However, the fact that we only have iO rather than ideal obfuscation makes this step non-trivial. Indeed, the token-based model can give false intuition since it allows us to watermark *any* PRF family but we show that in the standard model there are PRF families that cannot be watermarked. Nevertheless, it turns out that we can adapt the techniques from the token-based model to also work in the standard model using iO. The main differences are that: (1) we need the PRF family F that we are watermarking to be a *puncturable* PRF family, (2) instead of a standard CCA secure encryption, we need a special type of puncturable encryption scheme where we can create a punctured secret key which doesn't decrypt a particular ciphertext. The latter primitive may be of independent interest and we show how to construct it using iO. We use a careful sequence on hybrids to show that the above changes are sufficient to get a provably secure watermarking scheme in the standard model.

2.7 Related Work

There has been a large body of work on watermarking in the applied research community. Notable contributions of this line of research include the discovery of *protocol attacks* such as the copy attack by Kutter, Voloshynovskiy and Herrigel [KVH00] and the ambiguity attack by Adelsback, Katzenbeisser and Veith [AKV03]. However, these works do not formally define security guarantees, and have resulted in a cat-and-mouse game of designing watermarking schemes that are broken fairly immediately.

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). In particular, for all of these schemes, the mark 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.

Barak et al. [BGI+01, BGI+12] proposed simulation-based and indistinguishability-based definitions of watermarking security; their main contribution is a negative result, described earlier in the introduction, which shows that indistinguishability obfuscation rules out any meaningful form of watermarking that exactly preserves functionality. Finally, Hopper, Molnar and Wagner [HMW07] formalized strong notions of watermarking security with approximate functionality; our definitions are inspired by their work. Their definition considers not just unremovability, as we do, but also the dual notion of unforgeability which requires that the only marked programs that an adversary can produce are functionally similar to circuits already marked by a marking oracle. Though we have some partial results in this direction [CHV15], achieving unforgeability and unremovability simultaneously is a challenging and interesting open problem.

Organization of the Paper. We describe our main result, namely watermarking PRFs, in Section 6. Due to space limitation, the rest of our technical material can be found in the appendices: the definition of watermarking in Appendix 4, a new cryptographic object called puncturable encryption and its construction in Appendix 5, the proof of our PRF watermarking in Appendix 6.3, watermarking other cryptographic primitives in Appendix 8, negative results on watermarking in Appendix 9 and B and several extensions to our main construction in Appendix 7.

3 Preliminaries

Notation

For any $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$. For two strings x_1 and x_2, $x_1 \| x_2$ denotes a concatenation of x_1 and x_2.

When D is a distribution, we write $y \leftarrow D$ to denote that y is randomly sampled from D. If S is a set, then we will also write S to denote the uniform distribution on that set.

We say that a function $f : \mathbb{N} \to \mathbb{R}$ is negligible if for all constants $c > 0$, there exists $N \in \mathbb{N}$ such that for all $n > N$, $f(n) < n^{-c}$.

If $\mathcal{X} = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ are two ensembles of random variables indexed by $\lambda \in \mathbb{N}$, we say that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable if for all p.p.t. algorithms $\mathcal{D}$, there exists a negligible function ν such that for all λ,

$$\Pr\left[\mathcal{D}(x_b) = b \;\middle|\; \begin{array}{l} x_0 \leftarrow X_\lambda \\ x_1 \leftarrow Y_\lambda \\ b \leftarrow \{0,1\} \end{array}\right] \leq \frac{1}{2} + \nu(\lambda).$$

We write $\mathcal{X} \overset{\mathsf{c}}{\approx} \mathcal{Y}$ to denote that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable.

For two circuits C and D, we write $C \equiv D$ if C and D compute exactly the same function. If C and D agree on an ε fraction of their inputs, we write $C \cong_\varepsilon D$.

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 3.1 (Indistinguishability Obfuscation [BGI+12, GGH+13]) An indistinguishability obfuscator is a p.p.t. algorithm $i\mathcal{O}$ satisfying the following two conditions.

Functionality: For every security parameter $\lambda \in \mathbb{N}$ and every circuit C, it holds with probability 1 that

$$i\mathcal{O}(1^\lambda, C) \equiv C$$

Indistinguishability: For all circuit families $C^0 = \{C^0_\lambda\}$ and $C^1 = \{C^1_\lambda\}$ such that $C^0_\lambda \equiv C^1_\lambda$ are functionally equivalent and $|C^0_\lambda| = |C^1_\lambda|$, it holds that

$$\left\{i\mathcal{O}(1^\lambda, C^0_\lambda)\right\}_\lambda \overset{\mathsf{c}}{\approx} \left\{i\mathcal{O}(1^\lambda, C^1_\lambda)\right\}_\lambda$$

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 Generators and Functions. We review pseudorandom generators and several variants of pseudorandom functions (PRFs).

Definition 3.2 (Pseudorandom Generator) A pseudorandom generator (PRG) $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{\lambda + \ell(\lambda)}$ with stretch $\ell(\lambda)$ (ℓ is some polynomial function) is a polynomial-time computable function that satisfies $\mathsf{G}(U_\lambda) \overset{\mathsf{c}}{\approx} U_{\lambda+\ell(\lambda)}$ where U_m denotes the uniform distribution over $\{0,1\}^m$.

Definition 3.3 (Pseudorandom Functions) A pseudorandom function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ is a function family where each function $F \in \mathcal{F}_\lambda$ maps a domain D to a range R and satisfies the following condition. For all PPT adversary $\mathcal{A}$ and $F \leftarrow \mathcal{F}_\lambda$, it holds

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

where $F(\cdot) : \mathsf{D} \to \mathsf{R}$ is a deterministic function and $\mathcal{R}$ is chosen uniformly at random from the set of all functions with the same domain/range.

In this paper, we basically set $\mathsf{D} := \{0,1\}^{n(\lambda)}$ and $\mathsf{R} := \{0,1\}^{m(\lambda)}$ for a pair of polynomial-time computable functions $n(\cdot)$ and $m(\cdot)$.

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

Definition 3.4 (Puncturable Pseudorandom Functions) A puncturable pseudorandom function (pPRF) family $\mathcal{F}$ is a function family with a "puncturing" algorithm $\mathsf{Puncture}$ where each function $F \in \mathcal{F}_\lambda$ maps a domain $\{0,1\}^{n(\cdot)}$ to a range $\{0,1\}^{m(\cdot)}$ that satisfies 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

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

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 $\mathcal{A}$,

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

where $F \leftarrow \mathcal{F}_\lambda$, $F\{S\} := \mathsf{Puncture}(F, S)$ and U_ℓ denotes the uniform distribution over ℓ bits.

Theorem 3.5 ([GGM86, BW13b, BGI14b, KPTZ13b]) *If one-way functions exist, 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.*

Definition 3.6 (Injective pPRF) If a pPRF family $\mathcal{F} = \{\mathcal{F}_\lambda\}_\lambda$ satisfies the following, we call it an injective prefix pPRF family. For all $F \in \mathcal{F}_\lambda$ and $x, x' \in \mathsf{D}$, if $x \neq x'$, then $F(x) \neq F(x')$.

Sahai and Waters showed that we can convert any pPRF into a statistically injective pPRF [SW14]. Here, "statistically" means with probability $1 - \mathsf{negl}(\lambda)$ over the random choice of $F \leftarrow \mathcal{F}_\lambda$, $F(\cdot)$ is injective.

Definition 3.7 (Injective Bit-Commitment) An injective bit-commitment function is a p.p.t. algorithm Com which takes as input a security parameter λ and a bit $b \in \{0,1\}$, and outputs a commitment c, satisfying the following properties.

Computationally Hiding:

$$\left\{\mathsf{Com}(1^\lambda, 0)\right\}_\lambda \approx \left\{\mathsf{Com}(1^\lambda, 1)\right\}_\lambda$$

Perfectly Binding: For every λ, it holds that

$$\Pr\left[c_0 = c_1 \;\middle|\; \begin{array}{l} c_0 \leftarrow \mathsf{Com}(1^\lambda, 0) \\ c_1 \leftarrow \mathsf{Com}(1^\lambda, 1) \end{array}\right] = 0$$

Injective: For every security parameter λ, there is a bound ℓ_{rand} on the number of random bits used by Com such that $\mathsf{Com}(1^\lambda, \cdot\,; \cdot)$ is an injective function on $\{0,1\} \times \{0,1\}^{\ell_{rand}}$.

Definition 3.8 (Universal One-Way Hash Function) A universal one-way hash function (UOWHF) family $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ is a function family where each function $H \in \mathcal{H}_\lambda$ maps a domain D to a range R and satisfies the following condition. For all PPT adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$, it holds

$$\Pr\left[x \neq x^* \wedge H(x) = H(x^*) \;\middle|\; \begin{array}{l}(x,s) \leftarrow \mathcal{A}_1(1^\lambda),\\ H \leftarrow \mathcal{H}_\lambda,\\ x^* \leftarrow \mathcal{A}_2(1^\lambda, H, x, s)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Theorem 3.9 ([Rom90]) *If one-way functions exist, then UOWHFs exists.*

Hoeffding's Inequality We will use the following well-known bound. If $X_1, \ldots, X_N$ are independent Bernoulli variables with parameter p, then

$$\Pr\left[\sum_i X_i \geq (p+\varepsilon)\cdot N\right] \leq e^{-2\varepsilon^2 N}$$

In particular, if $N > \frac{\lambda}{\varepsilon^2}$, then this probability is exponentially small in λ.

4 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+12, Definition 8.4] (It is called occasional watermarking) 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), (3) we strengthen the definition to the public-key extraction setting and give the attacker access to the marking oracle.

Definition 4.1 (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 probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Mark}, \mathsf{Extract})$.

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

Mark: $\mathsf{Mark}(mk, C, \mathsf{msg})$ takes as input a mark key, an arbitrary circuit C (not necessarily in $\mathcal{C}_\lambda$) and a message $\mathsf{msg} \in \mathcal{M}_\lambda$ and outputs a marked circuit $\widetilde{C}$.

.

Extract: $\mathsf{msg}' \leftarrow \mathsf{Extract}(xk, C')$ takes as input an extraction key and an arbitrary circuit C', and outputs $\mathsf{msg}' \leftarrow \mathsf{Extract}(xk, C')$ where $\mathsf{msg}' \in \mathcal{M} \cup \{\mathsf{unmarked}\}$.

We are now ready to define the required correctness and security properties of a watermarking scheme.

Definition 4.2 (Watermarking Security) A watermarking scheme $(\mathsf{Gen}, \mathsf{Mark}, \mathsf{Extract})$ 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

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

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

$$\Pr[\mathsf{msg}' \neq \mathsf{msg} \;\big|\; \mathsf{msg}' \leftarrow \mathsf{Extract}(xk, \mathsf{Mark}(mk, C, \mathsf{msg}))] \leq \mathsf{negl}(\lambda).$$

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

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

ε-Unremovability: For every PPT $\mathcal{A}$ we have

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

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.

We say a watermarking scheme is ε-secure if it satsifies these properties.

Definition 4.3 (ε-Unremovability Security Game) The game $\mathsf{Exp}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon)$ is defined as follows.

1. The challenger generates $(xk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$ and gives xk 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 (not necessarily in $\mathcal{C}_\lambda$) and message msg_i, then it answers with $\mathsf{Mark}(mk, C_i, \mathsf{msg}_i)$.
3. At some point, the adversary makes a query to the challenge oracle $\mathcal{CO}$. If $\mathcal{CO}$ is queried with a message $\mathsf{msg} \in \mathcal{M}_\lambda$, it samples a circuit $C \leftarrow \mathcal{C}_\lambda$ uniformly at random and answers $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})$.
4. Again, $\mathcal{A}$ queries many pairs of a circuit and a message to $\mathcal{MO}$.
5. Finally, the adversary outputs a circuit C^*. If it holds that $C^* \cong_\varepsilon C$ and $\mathsf{Extract}(xk, C^*) \neq \mathsf{msg}$ then the experiment outputs 1, otherwise 0. [7]

Our main construction achieves what we call "lunch-time security", in which step 4 of the above game is omitted. This and other variations are discussed in Section 7.

5 Puncturable Encryption

One of our main abstractions is a *puncturable encryption* system. This is a public-key encryption system in which the decryption key can be punctured on a set of ciphertexts. We will rely on a strong ciphertext pseudorandomness property which holds even given access to a punctured decryption key. We will additionally require that valid ciphertexts are *sparse*, and that a decryption key punctured at two ciphertexts $\{c_0, c_1\}$ is functionally equivalent to the non-punctured decryption key, except possibly on $\{c_0, c_1\}$.

In this section we define the puncturable encryption abstraction that we use in Section 6. We instantiate this definition in Section A.1 and prove its security Section A.2.

Definition 5.1 (Puncturable Encryption Syntax) Syntactically, a puncturable encryption scheme PE for a message space $\mathcal{M} = \{0,1\}^\ell$ is a triple of probabilistic algorithms $(\mathsf{Gen}, \mathsf{Puncture}, \mathsf{Enc})$ and a deterministic algorithm Dec. The space of ciphertexts will be $\{0,1\}^n$ where $n = \mathrm{poly}(\ell, \lambda)$. For clarity and simplicity, we will restrict our exposition to the case when $\lambda = \ell$.

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

Key Generation: $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ takes the security parameter in unary, and outputs an encryption key pk and a decryption key sk.

Puncturing: $sk\{c_0, c_1\} \leftarrow \mathsf{Puncture}(sk, c_0, c_1)$ takes a decryption key sk, and a set $\{c_0, c_1\} \subset \{0,1\}^n$.[8] Puncture outputs a "punctured" decryption key $sk\{c_0, c_1\}$.

Encryption: $c \leftarrow \mathsf{Enc}(pk, m)$ takes an encryption key pk and a message $m \in \{0,1\}^\ell$, and outputs a ciphertext c in $\{0,1\}^n$.

Decryption: m or $\perp \leftarrow \mathsf{Dec}(sk, c)$ takes a possibly punctured decryption key sk and a string $c \in \{0,1\}^n$. It outputs a message m or the special symbol $\perp$.

Definition 5.2 (Puncturable Encryption Security) A puncturable encryption scheme $\mathsf{PE} = (\mathsf{Gen}, \mathsf{Puncture}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ is required to satisfy the following properties.

Correctness: We require that for all messages m,

$$\Pr\left[\mathsf{Dec}(sk, c) = m \;\middle|\; \begin{array}{l}(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda), \\ c \leftarrow \mathsf{Enc}(pk, m)\end{array}\right] = 1.$$

Punctured Correctness: We also require the same to hold for keys which are punctured. For all possible keys $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$, all strings $c_0, c_1 \in \{0,1\}^n$, all punctured keys $sk' \leftarrow \mathsf{Puncture}(sk, c_0, c_1)$, and all potential ciphertexts $c \in \{0,1\}^n \setminus \{c_0, c_1\}$:

$$\mathsf{Dec}(sk, c) = \mathsf{Dec}(sk', c).$$

Ciphertext Pseudorandomness: We require that in the following game, all PPT adversaries $\mathcal{A}$ have negligible advantage.

Game 5.3 (Ciphertext Pseudorandomness)

1. *$\mathcal{A}$ sends a message m^* to the challenger.*
2. *The challenger does the following:*
   - *Samples* $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$
   - *Computes encryption* $c^* \leftarrow \mathsf{Enc}(pk, m^*)$.
   - *Samples* $r^* \leftarrow \{0,1\}^n$.
   - *Generates the punctured key* $sk' \leftarrow \mathsf{Puncture}(sk, \{c^*, r^*\})$
   - *Samples* $b \leftarrow \{0,1\}$ *and sends the following to* $\mathcal{A}$*:*

$$\begin{array}{ll}(c^*, r^*, pk, sk') & \text{if } b = 0 \\ (r^*, c^*, pk, sk') & \text{if } b = 1\end{array}$$

3. *The adversary outputs b' and wins if $b = b'$.*

Sparseness: We also require that most strings are not valid ciphertexts:

$$\Pr\left[\mathsf{Dec}(sk, c) \neq \perp \;\middle|\; (pk, sk) \leftarrow \mathsf{Gen}(1^\lambda), c \leftarrow \{0,1\}^n\right] \leq \mathsf{negl}(\lambda).$$

One of our contributions is the following theorem.

Theorem 5.4 *A puncturable encryption system can be constructed using indistinguishability obfuscation and injective one-way functions.*

A full construction and proof is provided in appendices A.1 and A.2.

[8] We can assume that the set $\{c_0, c_1\}$ is represented as a list in sorted order.

6 Watermarking PRFs

In this section, we construct schemes for watermarking any puncturable PRF family. One is secure against lunch-time attacks and the other is fully secure. Both of them are in the public-key extraction setting. As we explain in Section 2.3, the simple scheme is not secure in these settings (the attacker has access to the marking or extraction oracles).

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

Theorem 6.1 *Assuming the existence of injective one-way functions, and an indistinguishability obfuscator for all circuits, for all $\varepsilon(\lambda) = \frac{1}{2} + 1/\mathrm{poly}(\lambda)$, all message spaces $\mathcal{M} = \{0,1\}^w$ (for $w = \mathrm{poly}(\lambda)$), all integer functions $n(\lambda) = \Omega(\lambda)$ and $m(\lambda) = \Omega(\lambda)$ there exists a watermarking scheme with message space $\mathcal{M}$ which is ε-secure against lunch-time attacks for every pPRF ensemble $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ such that functions C in $\mathcal{C}_\lambda$ map $\{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$.*

If we assume pPRF family $\mathcal{C} = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ satisfies a "nice" property, that is, the injective property in Definition 7.1, then we can show the full security in Definition 4.3 where the adversary has access to the mark oracle even after the challenge program is given. See Section 7.1 for the details.

Construction: Public-Key Extraction and Security against Lunch-Time Attacks. We now construct a watermarking scheme with public-key extraction and with security against lunch-time attacks in the presence of a marking oracle (see Definition 4.3). We have already explained the challenges in constructing such a scheme in Section 2.4. We start with the scheme outline.

6.1 Scheme Outline

Assume we want to mark a PRF family $\mathcal{C}$ with domain $\mathsf{D} = \{0,1\}^n$ and range $\mathsf{R} = \{0,1\}^m$, where both n and m are sufficiently large. In this overview, suppose for simplicity that the space of marks is $\{0,1\}^m$. Our construction relies on a puncturable encryption scheme PE with ciphertext space $\mathcal{C} = \{0,1\}^n$ and message space $\mathcal{M} = \{0,1\}^\ell$ for sufficiently large ℓ. We follow the watermarking framework described in the introduction, in which a marked program is changed on a small set of "marked points", determined by a set of "find points" which are *not* changed.

Roughly speaking, a marked point in our scheme is a valid ciphertext of PE. A valid ciphertext when marking a program C is defined as any encryption of any plaintext $a\|b\|c$ such that $b = H(C(\mathsf{G}(a)))$. On such inputs, the marked program's output is changed to $\mathsf{G}'(c) \oplus \mathsf{msg}$ where G' is a publicly known pseudorandom generator and msg is the desired mark. Note that there are super-polynomially many marked-points, but yet they are only a negligible fraction of the total domain.

Given the above marking scheme, there is a natural procedure to extract the mark msg. We first pick random values $a, c \leftarrow \{0,1\}^{\ell/3}$ and compute the corresponding find-point $\alpha := \mathsf{G}(a)$. Then we compute $b := H(C'(\alpha))$ and use this to find the corresponding marked-point $x \leftarrow \mathsf{PE.Enc}(pk, a\|b\|c)$. Finally, we compute $y = C'(x)$ and record $\mathsf{msg}' := y \oplus \mathsf{G}'(c)$ as a candidate for the embedded message. If $C' = \mathsf{Mark}(C)$, correctness is obvious. The bulk of our work is making extraction work for arbitrary efficiently computable $C' \approx_\varepsilon \mathsf{Mark}(C)$.

In order to guarantee that the correct message is extracted with high probability , we amplify our procedure in two steps. First, we fix $a\|b$ and sample multiple independent c's, extract as above, and take the majority result. We then repeat this process with independently sampled a's, and again taking the majority result. Compared to earlier versions of this work [CHV15, NW15], this "majority of majorities" approach allows us to attain optimal thresholds for unremovability (any $\frac{1}{2} + \frac{1}{\mathrm{poly}(\lambda)}$).

6.2 A Message-Embedding Construction

In this section, we formally construct our main message-embedding watermarking scheme. We show it satisfies unremovability in the public-key extraction setting and in the presence of a marking oracle. We obtain a scheme in which unremovability holds for any approximation factor $\varepsilon(\lambda) = \frac{1}{2} + 1/\mathrm{poly}(\lambda)$.

Setup. Our goal is to construct a watermarking scheme for a pPRF family $\mathcal{C}$ with domain $\{0,1\}^n$ and range $\{0,1\}^m$. For any positive integer w, let $\mathcal{M} = \{0,1\}^{w \cdot m}$ denote the message space. We will think of messages $\mathsf{msg} \in \mathcal{M}$ as consisting of w/m chunks in $\{0,1\}^m$, so we will write $\mathsf{msg} = \mathsf{msg}_1 \| \cdots \| \mathsf{msg}_w$. Let PE be a puncturable encryption scheme with ciphertext length n and plaintext length $\ell + \log w$. Let $\mathsf{G} : \{0,1\}^{\ell/3} \to \{0,1\}^n$ and $\mathsf{G}' : \{0,1\}^{\ell/3} \to \{0,1\}^m$ be PRGs, and let $H : \{0,1\}^m \to \{0,1\}^{\ell/3}$ be a UOWHF.

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

$\mathsf{Gen}(1^\lambda)$**:** Sample a key pair $(pk, sk) \leftarrow \mathsf{PE.Gen}(1^\lambda)$. Output (xk, mk) where $xk = pk$ and $mk = sk$.

$\mathsf{Mark}(mk, C, \mathsf{msg})$**:** Outputs the iO-obfuscation of circuit M constructed from C in Fig. 1, i.e., $i\mathcal{O}(M)$.

Constants: PE decryption key sk, pPRF F, circuit C, and message $\mathsf{msg} = \mathsf{msg}_1 \| \ldots \| \mathsf{msg}_w$
Inputs: $x \in \{0,1\}^n$

1. Try to parse $a \| b \| c \| i \leftarrow \mathsf{PE.Dec}(sk, x)$, where $|a| = |b| = |c| = \ell/3$ and $i \in [w]$.
2. If $a \| b \| c \| i \neq \perp$ and $H(C(\mathsf{G}(a))) = b$, output $\mathsf{G}'(c) \oplus \mathsf{msg}_i$.
3. Otherwise, output $C(x)$.

Figure 1: The program M, which is a modification of C (pre-obfuscated program)

$\mathsf{Extract}(xk, C')$**:** For each $i \in [w]$, let $\mathsf{msg}_i = \mathsf{Extract}_i(xk, C')$, where $\mathsf{Extract}_i$ is defined in Fig. 2. $\mathsf{Extract}_i$ makes use of a subroutine $\mathsf{WeakExtract}_i$, which is defined in Fig. 3. Output $\mathsf{msg}_1 \| \ldots \| \mathsf{msg}_w$.

$\mathsf{Extract}_i(xk, C')$:

1. For $j = 1, \ldots, Q$,
   (a) Sample uniformly random $a_j \leftarrow \{0,1\}^{\ell/3}$.
   (b) Compute $b_j = H(C'(\mathsf{G}(a_j)))$
   (c) Run $\mathsf{msg}_i^{(j)} \leftarrow \mathsf{WeakExtract}_i(xk, C', a_j, b_j)$
2. If there exists a "majority-of-majorities message" $\mathsf{msg}_i \neq \perp$ such that $|\{j : \mathsf{msg}_i^{(j)} = \mathsf{msg}_i\}| > Q/2$, then output msg_i; else output unmarked.

Figure 2: The sub-routine algorithm $\mathsf{Extract}_i(xk, C')$

It is easy to check that this construction satisfies statistical and extraction correctness, and meaningfulness.

Proposition 6.2 *The above construction satisfies Theorem 6.1.*

6.3 Security Proofs

To prove the proposition, we must prove ε-unremovability against lunch-time attacks.

$\mathsf{WeakExtract}_i(xk, C', a, b)$:

1. For $k = 1, \ldots, R$,
   (a) Sample $c_k \leftarrow \{0,1\}^{\ell/3}$ and $x_k \leftarrow \mathsf{PE.Enc}(pk, a\|b\|c_k\|i)$.
   (b) Compute $\mathsf{msg}_i^{(k)} = \mathsf{G}'(c_k) \oplus C'(x_k)$.
2. Define the "majority message" msg_i such that $|\{k : \mathsf{msg}_i^{(k)} = \mathsf{msg}_i\}| > R/2$ if such a msg_i exists; otherwise, define $\mathsf{msg}_i = \bot$.

Figure 3: The sub-routine algorithm $\mathsf{WeakExtract}_i(xk, C', a, b)$

Overview. Recall that in our scheme, there are two sparse sets of points: "find points", which are unchanged between a marked and unmarked program, and "mark points", which are changed. To extract from a circuit C', one repeatedly performs the following 4 steps, which we will refer to as *weak extraction*:

1. Sample a find point x, and queries $C(x)$
2. Use the resulting value to sample many mark points $x_1, \ldots, x_k$, where $k = \lambda/\rho^2$
3. For each x_i, query $C(x_i)$ to compute a guess msg_i
4. If some msg_i occurs more than $k/2$ times, return it. Otherwise, return $\bot$.

If this procedure returns some message msg many times (more than half), then msg is the extracted value.

Weak extraction can fail if $C(x)$ has been changed by the remover, or if most of $C(x_1), \ldots, C(x_k)$ have been changed. The first happens with probability at most $1 - \varepsilon$, by the pseudorandomness of find points. The second happens with negligible probability by a Chernoff bound. By repeating this process with many find points, the error probability is reduced to negligible.

Proof of ε-unremovability. First, we define two security experiments to state a useful lemma that is used to prove Proposition 6.2. These two experiments are similar to the unremovability security game, but the goal of the adversary is now to distinguish a mark-point of a marked program from a uniformly random string of the same length, while first given access to a marking oracle and also given the corresponding find point.

For any PPT adversary $\mathcal{D}$, we define the following two experiments, $\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}(\lambda, i)$ and $\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}(\lambda)$.

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}(\lambda, i)$**:**

1. $(xk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$
2. $(s, \mathsf{msg}) \leftarrow \mathcal{D}^{\mathsf{Mark}(mk,\cdot,\cdot)}(xk)$
3. $C \leftarrow \mathcal{C}$ and $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})$
4. $a \leftarrow \{0,1\}^{\ell/3}$, $b = H(\widetilde{C}(\mathsf{G}(a)))$
5. $c \leftarrow \{0,1\}^{\ell/3}$
6. $x_{\mathsf{REAL}} \leftarrow \mathsf{PE.Enc}(pk, a\|b\|c\|i)$
7. Finally, output $\mathcal{D}(s, \widetilde{C}, a, x_{\mathsf{REAL}})$

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}(\lambda)$**:**

1. $(xk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$
2. $(s, \mathsf{msg}) \leftarrow \mathcal{D}^{\mathsf{Mark}(mk,\cdot,\cdot)}(xk)$
3. $C \leftarrow \mathcal{C}$ and $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})$

4. $a \leftarrow \{0,1\}^{\ell/3}$
5. $x_{\mathsf{RAND}} \leftarrow \{0,1\}^n$
6. Finally, output $\mathcal{D}(s, \widetilde{C}, a, x_{\mathsf{RAND}})$

Lemma 6.3 *Under the same conditions as in Theorem 6.1, for all PPT distinguishers $\mathcal{D}$ and for all $i \in [w]$, it holds that*

$$\left|\Pr[\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}(\lambda, i) = 1] - \Pr[\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}(\lambda) = 1]\right| < \mathsf{negl}(\lambda)$$

We also define a "many-message" version of these two experiments:

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}^R}(\lambda, i)$**:**

1. $(xk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$
2. $(s, \mathsf{msg}) \leftarrow \mathcal{D}^{\mathsf{Mark}(mk,\cdot,\cdot)}(xk)$
3. $C \leftarrow \mathcal{C}$ and $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})$
4. $a \leftarrow \{0,1\}^{\ell/3}$, $b = H(\widetilde{C}(\mathsf{G}(a)))$.
5. $c \leftarrow \{0,1\}^{\ell/3}$
6. For $j = 1, \ldots, R$:
   sample $x_{\mathsf{REAL},j} \leftarrow \mathsf{PE.Enc}(pk, a\|b\|c\|i)$
7. Finally, output $\mathcal{D}(s, \widetilde{C}, a, \boldsymbol{x}_{\mathsf{REAL}})$, where $\boldsymbol{x}_{\mathsf{REAL}} = (x_{\mathsf{REAL},1}, \ldots, x_{\mathsf{REAL},R})$.

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}^R}(\lambda)$**:**

1. $(xk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$
2. $(s, \mathsf{msg}) \leftarrow \mathcal{D}^{\mathsf{Mark}(mk,\cdot,\cdot)}(xk)$
3. $C \leftarrow \mathcal{C}$ and $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})$
4. $a \leftarrow \{0,1\}^{\ell/3}$
5. For $j = 1, \ldots, R$:
   sample $x_{\mathsf{RAND},j} \leftarrow \{0,1\}^n$
6. Finally, output $\mathcal{D}(s, \widetilde{C}, a, \boldsymbol{x}_{\mathsf{RAND}})$, where $\boldsymbol{x}_{\mathsf{RAND}} = (x_{\mathsf{RAND},1}, \ldots, x_{\mathsf{RAND},R})$.

Corollary 6.4 *For all p.p.t. $\mathcal{D}$ and for all $i \in [w]$, it holds that*

$$\left|\Pr[\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}^R}(\lambda, i) = 1] - \Pr[\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}^R}(\lambda) = 1]\right| < \mathsf{negl}(\lambda)$$

Proof. This follows from a simple hybrid argument. □

Before proving Lemma 6.3, we first show that it would imply Proposition 6.2.

Proof of Proposition 6.2. We show that for every i and every p.p.t. adversary $(\mathcal{A}_1, \mathcal{A}_2)$,

$$\Pr\left[\mathsf{Extract}_i(xk, C^*) \neq \mathsf{msg}^{(i)} \wedge C^* \cong_\varepsilon \widetilde{C} \;\middle|\; \begin{array}{l} (xk, mk) \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{msg}, s) \leftarrow \mathcal{A}_1^{\mathsf{Mark}(mk,\cdot,\cdot)}(1^\lambda, xk, mk) \\ C \leftarrow \mathcal{C} \\ \widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg}) \\ C^* \leftarrow \mathcal{A}_2(s, \widetilde{C}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Suppose for the sake of contradiction that a p.p.t. adversary $(\mathcal{A}_1, \mathcal{A}_2)$ wins this game with non-negligible probability. That is, with non-negligible probability, $\mathcal{A}_2$ outputs a program $C^* \cong_\varepsilon \widetilde{C}$ such that

$\mathsf{Extract}_i(C^*) \neq \mathsf{msg}^{(i)}$ with non-negligible probability. For convenience of notation, let Δ denote the point-wise xor of $\widetilde{C}$ and C^*. That is, let $\Delta(x) = C^*(x) \oplus \widetilde{C}(x)$. Recall that $\varepsilon(\lambda) = \frac{1}{2} + \rho(\lambda)$. Because $\mathsf{Extract}_i$ takes the majority answer after running $\mathsf{WeakExtract}_i$ many $(\lambda/\rho(\lambda)^2)$ times, it must be (by a Chernoff bound) that for any such C^*,

$$p_{C^*} := \Pr\left[\mathsf{WeakExtract}_i(C^*) \neq \mathsf{msg}^{(i)}\right] \geq \frac{1}{2} - \rho(\lambda) + \frac{1}{\mathrm{poly}(\lambda)}$$

for some polynomial poly. Since $\mathsf{WeakExtract}_i$ only accesses C^* in a black-box way, and since we know that $\mathsf{WeakExtract}_i(\widetilde{C}) = \mathsf{msg}^{(i)}$ with high probability, it must be the case that C^* differs from $\widetilde{C}$ at some of the points queried by $\mathsf{WeakExtract}_i$. Furthermore $\mathsf{WeakExtract}_i$ is robust against differences at mark points (since it suffices for C^* to agree with $\widetilde{C}$ at a majority of the queried mark points). Thus we have (by a union bound) that

$$p_{C^*} \leq \Pr_a\left[\Delta(\mathsf{G}(a)) \neq 0\right] + \Pr_{\substack{a \leftarrow \{0,1\}^{\ell/2} \\ x_k \leftarrow \mathsf{PE.Enc}(a\|b\|i)}}\left[|\{k : \Delta(x_k) \neq 0 \wedge k \in [R]\}| > \frac{R}{2}\right] + \mathsf{negl}(\lambda)$$

The first term corresponds to the probability of $\mathcal{A}$ changing the find point queried by $\mathsf{WeakExtract}_i$, and the second corresponds to the probability of $\mathcal{A}$ changing many mark points. The third term is the probability that $\mathsf{WeakExtract}_i(\widetilde{C}) \neq \mathsf{msg}_i$.

For the first term, we note that by the pseudorandomness of $\mathsf{G}(a)$, it must hold that for all polynomials poly, there is a negligible negl such that

$$\Pr\left[C^* \cong_\varepsilon \widetilde{C} \wedge \Pr_a\left[\Delta(\mathsf{G}(a)) \neq 0\right] \geq 1 - \varepsilon(\lambda) + \frac{1}{\mathrm{poly}(\lambda)}\right] \leq \mathsf{negl}(\lambda).$$

Indeed, otherwise we can break the security of G by running $\mathcal{A}$, and empirically testing whether the Δ output by $\mathcal{A}_2$ exhibits a $\frac{1}{\mathrm{poly}(\lambda)}$ advantage in distinguishing $\mathsf{G}(a)$ points from random points. If it does, we evaluate Δ on our challenge to try to distinguish; otherwise we guess randomly.

For the other term, Corollary 6.4 states that the x_i's are jointly indistinguishable from i.i.d. random x_i's sampled from $\{0,1\}^m$, even though $\mathcal{A}_1$ has oracle access to $\mathsf{Mark}(mk, \cdot, \cdot)$. Combined with a Chernoff bound, which states that

$$\Pr\left[C^* \cong_\varepsilon \widetilde{C} \wedge \Pr_{x_1,\ldots,x_R \leftarrow \{0,1\}^n}\left[|\{x_k : \Delta(x_k) \neq 0\}| > \frac{R}{2}\right] \geq \frac{1}{\mathrm{poly}(\lambda)}\right] = 0,$$

this implies that for every polynomial poly,

$$\Pr\left[C^* \cong_\varepsilon \widetilde{C} \wedge \Pr_{\substack{a \leftarrow \{0,1\}^{\ell/2} \\ x_1,\ldots,x_R \leftarrow \mathsf{PE.Enc}(a\|b\|i)}}\left[|\{x_k : \Delta(x_k) \neq 0\}| > \frac{R}{2}\right] \geq \frac{1}{\mathrm{poly}(\lambda)}\right] \leq \mathsf{negl}(\lambda).$$

Combining these four inequalities yields a contradiction.

□

Now we turn to proving Lemma 6.3.

Proof of Lemma 6.3. We define a sequence of hybrid experiments to prove this lemma. We call all variables that $\mathcal{D}$ sees in the experiment Exp a view of $\mathcal{D}$ and denote it by $\mathsf{view}(\mathsf{Exp})$.

Hyb_0**:** This experiment is exactly the same as $\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}(\lambda, i)$.

Constants: punctured PE decryption key $sk' := sk\{x_0, x_1\}$, pPRF key $C^{(\iota)}$, values x_0, x_1, message $\mathsf{msg}^{(\iota)} = \mathsf{msg}_1^{(\iota)} \| \cdots \| \mathsf{msg}_w^{(\iota)}$

Inputs: $x \in \{0,1\}^n$

1. If $x \in \{x_0, x_1\}$, then output $C^{(\iota)}(x)$.
2. Compute $a \| b \| c \| i \leftarrow \mathsf{PE.Dec}(sk', x)$, where $|a| = |b| = |c| = \ell/3$, and $i \in [w]$.
3. If $a \| b \| c \| i \neq \bot$ and $H(C^{(\iota)}(\mathsf{G}(a))) = b$, output $\mathsf{G}'(c) \oplus \mathsf{msg}_i^{\iota}$.
4. Otherwise, output $C^{(\iota)}(x)$.

Figure 4: Program $M^{(\iota)}\{x_0, x_1\}$ in Hyb_1

Hyb_1**:** In this hybrid experiment, we change the marking oracle. For the adversary's queries $(C^{(1)}, \mathsf{msg}^{(1)}), \ldots, (C^{(q)}, \mathsf{msg}^{(q)})$, instead of generating marked program $\widetilde{C}^{(\iota)} \leftarrow i\mathcal{O}(M^{(\iota)})$, we set $\widetilde{C}^{(\iota)} \leftarrow i\mathcal{O}(M^{(\iota)}\{x_0, x_1\})$ where $M^{(\iota)}\{x_0, x_1\}$ is defined in Figure 4, having hard-coded $C^{(\iota)}$, $sk' \leftarrow \mathsf{PE.Puncture}(sk, x_0, x_1)$, $x_0 := x_{\mathsf{REAL}}$, $x_1 \leftarrow \{0,1\}^n$, and $\mathsf{msg}^{(\iota)}$.

Hyb_2**:** In this hybrid experiment, we change the marked challenge program $\widetilde{C}$. We use the punctured decryption key sk' and hard-code the output values corresponding to x_0 and x_1 as $y_0 = \mathsf{G}'(c)$ and $y_1 \leftarrow \{0,1\}^m$ respectively. That is, we set $\widetilde{C} \leftarrow i\mathcal{O}(M\{x_0, x_1\})$ where $M\{x_0, x_1\}$ is defined in Figure 5.

Constants: punctured PE decryption key $sk' := sk\{x_0, x_1\}$, pPRF key F, pPRF key C, values x_0, x_1, y_0, y_1, message $\mathsf{msg} = \mathsf{msg}_1 \| \cdots \| \mathsf{msg}_w$

Inputs: $x \in \{0,1\}^n$

1. If $x = x_\sigma$ for $\sigma \in \{0,1\}$, then output y_σ.
2. Compute $a \| b \| c \| i \leftarrow \mathsf{PE.Dec}(sk', x)$, where $|a| = |b| = |c| = \ell/3$ and $i \in [w]$.
3. If $a \| b \| c \| i \neq \bot$ and $H(C(\mathsf{G}(a))) = b$, output $\mathsf{G}'(c) \oplus \mathsf{msg}_i$.
4. Otherwise, output $C(x)$.

Figure 5: Program $M\{x_0, x_1\}$ in Hyb_2

Hyb_3 In this experiment, x_0 is changed to be uniformly sampled from $\{0,1\}^n$.

Hyb_4 In this experiment, y_0 is changed to be uniformly sampled from $\{0,1\}^m$

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}$**:** The only changes from Hyb_3 are that the challenge program $\widetilde{C}$ and marked keys $\widetilde{C}^{(\iota)}$ for all $\iota \in [q]$ are changed back to the original programs but the values x_0 remain random.

We describe an overview of the main hybrid experiments in Table 1.

Lemma 6.5 *If $\mathcal{F}$ is a pPRF family, H is a UOWHF,* PE *satisfies the punctured correctness and sparseness, and $i\mathcal{O}$ is a secure indistinguishability obfuscator, then* $\mathsf{view}(\mathsf{Hyb}_0) \overset{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}_1)$.

Proof of Lemma 6.5. To prove the lemma, we define auxiliary hybrid experiments Hyb_0^{ι} for $\iota \in [q]$ where the mark oracle gives $i\mathcal{O}(M^{(\iota)}\{x_0, x_1\})$ for the first ι queries $C^{(1)}, \ldots, C^{(\iota)}$ of $\mathcal{D}$.

Table 1: An overview of hybrid experiment

Hybrid experiment	Challenge: $i\mathcal{O}(\cdot)$	Answers of $\mathcal{MO}$: $i\mathcal{O}(\cdot)$	x_0	x_1
$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}$	M	$M^{(\iota)}$	x_{REAL}	none
Hyb_1	M	$M^{(\iota)}\{x_0, x_1\}$	x_{REAL}	random
Hyb_2	$M\{x_0, x_1\}$	$M^{(\iota)}\{x_0, x_1\}$	x_{REAL}	random
Hyb_3	$M\{x_0, x_1\}$	$M^{(\iota)}\{x_0, x_1\}$	x_{RAND}	random
$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}$	M	$M^{(\iota)}$	x_{RAND}	none

Claim: In Hyb^{ι}_0, the probability that $H(C^{(\iota+1)}(\mathsf{G}(a))) = b$ is negligible, where $b := H(\widetilde{C}(\mathsf{G}(a)))$.

Proof. If for some p.p.t. $\mathcal{D}$, this event happens with non-negligible probability, we show how to invert H at a random input with nearly the same non-negligible probability, thus contradicting the one-wayness of H.

We use the fact that $C(\mathsf{G}(a))$, and therefore $\widetilde{C}(\mathsf{G}(a))$, is pseudorandom, because up until this point in the game, the only information $\mathcal{D}$ has about C comes from the marking oracle hard-coding $x_0 = \mathsf{Enc}(a\|b\|c)$ in its answers. So if b is replaced by a random challenge $H(r)$, $C^{(\iota+1)}(\mathsf{G}(a))$ must still be a pre-image of b with non-negligible probability.

□

Claim: $\mathsf{view}(\mathsf{Hyb}^{\iota}_0) \overset{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}^{\iota+1}_0)$ for all $\iota \in [q]$.

Proof. The only difference between Hyb^{ι}_0 and $\mathsf{Hyb}^{\iota+1}_0$ is the $(\iota+1)$-th answer by the mark oracle. We show that the mark oracle's answers are functionally equivalent in the two games, so indistinguishability follows from the security of $i\mathcal{O}$.

There are only two possible inputs on which $M^{(\iota+1)}$ may differ in Hyb^{ι}_0 and $\mathsf{Hyb}^{\iota+1}_0$: namely, x_0 and x_1 due to the punctured correctness at non-punctured points of PE. We show that (with high probability) they respectively mapped to $C^{(\iota+1)}(x_0)$ and $C^{(\iota+1)}(x_1)$ without our changes, just as they do with our changes.

It holds that $\mathsf{PE.Dec}(sk, x_1) = \bot$ with high probability since x_1 is uniformly random and PE satisfies sparseness. Thus, $M^{(\iota+1)}(x_1) = C^{(\iota+1)}(x_1)$ in Hyb^{ι}_0. This is also true in $\mathsf{Hyb}^{\iota+1}_0$ since $M^{(\iota+1)}\{x_0, x_1\}(x_1)$ goes to the punctured-points branch.

On the other hand, x_0 decrypts as $a\|b\|c\|i$, but by our previous claim, it cannot be the case that $H(C^{(\iota+1)}(\mathsf{G}(a))) = b$. Thus, $M^{(\iota+1)}(x_0) = C^{(\iota+1)}(x_0)$ in Hyb^{ι}_0.

□

We completed the proof of the lemma by the two claims. □

Lemma 6.6 *If C is a pPRF, PE satisfies the punctured correctness, and $i\mathcal{O}$ is a secure indistinguishability obfuscator, then* $\mathsf{view}(\mathsf{Hyb}_1) \overset{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}_2)$.

Proof of Lemma 6.6. We define auxiliary hybrid experiments as follows.

Hyb^1_1: Instead of choosing challenge program $\widetilde{C} \leftarrow i\mathcal{O}(M)$ where the program M is described in Figure 1, we now use punctured keys sk' and $C\{x_1\}$ and set $\widetilde{C} \leftarrow i\mathcal{O}(M_1\{x_0, x_1\})$ where $M_1\{x_0, x_1\}$ is defined in Figure 6, $y_0 := \mathsf{G}'(c) \oplus \mathsf{msg}_i$ and $y_1 := C(x_1)$.

Constants: punctured PE decryption key $sk' := sk\{x_0, x_1\}$, punctured pPRF key $C' = C\{x_1\}$, values x_0, x_1, y_0, y_1, message $\mathsf{msg} = \mathsf{msg}_1 \| \cdots \| \mathsf{msg}_w$

Inputs: $x \in \{0,1\}^n$

1. If $x = x_\sigma$ for $\sigma \in \{0,1\}$, then output y_σ.
2. Compute $a\|b\|c\|i \leftarrow \mathsf{PE.Dec}(sk', x)$, where $|a| = |b| = |c| = \ell/3$, and $i \in [w]$.
3. If $a\|b\|c\|i \neq \perp$ and $H(C'(\mathsf{G}(a))) = b$, output $\mathsf{G}'(c) \oplus \mathsf{msg}_i$.
4. Otherwise, output $C'(x)$.

Figure 6: Program $M_1\{x_0, x_1\}$ in Hyb_1^1

Hyb_1^2 : We choose uniformly random $y_1 \leftarrow \{0,1\}^m$ and hard-code it in the program $M\{x_0, x_1\}$.

Claim: $\mathsf{view}(\mathsf{Hyb}_1) \stackrel{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}_1^1)$

Proof. Program $M_1\{x_0, x_1\}$ is functionally equivalent to Program M in Hyb_1, because we just hard-coded the values for y_0 and y_1 which would be output anyways. Also, replacing C by $C\{x_1\}$ does not change functionality because, by line 1, C is never evaluated at x_1.Thus, the claim holds due to the security of $i\mathcal{O}$. □

Claim: $\mathsf{view}(\mathsf{Hyb}_1^1) \stackrel{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}_1^2)$

Proof. This follows from the pseudorandomness of $C\{x_1\}$ at x_1. □

Claim: $\mathsf{view}(\mathsf{Hyb}_1^2) \stackrel{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}_2)$

Proof. In Hyb_2, C is un-punctured in the challenge program $i\mathcal{O}(M\{x_0, x_1\})$, but $M\{x_0, x_1\}$ is still functionally equivalent to the program in Hyb_1^2 due to line 1. Therefore, the claim holds due to the security of $i\mathcal{O}$. □

The proof of the lemma follows from these three claims. □

Lemma 6.7 *If* PE *satisfies ciphertext randomness, then* $\mathsf{view}(\mathsf{Hyb}_2) \stackrel{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}_3)$.

Proof of Lemma 6.7. This reduces to the ciphertext randomness property of PE. If some p.p.t. distinguisher $\mathcal{D}$ distinguishes Hyb_2 from Hyb_3, we construct a p.p.t. $\mathcal{A}$ with non-negligible advantage in the ciphertext pseudorandomness game.

First, $\mathcal{A}$ chooses $a \leftarrow \{0,1\}^{\ell/3}$, $c \leftarrow \{0,1\}^{\ell/3}$, $C \leftarrow \mathcal{C}_\lambda$, and a UOWHF H, computes $b := H(C(\mathsf{G}(a)))$, and sends $m_0 := a\|b\|c\|i$ and uniformly random $m_1 \leftarrow \{0,1\}^{\ell+|w|}$ as a challenge. Then, the challenger of PE returns $(c_\sigma, c_{1-\sigma}, pk, sk')$ where $\sigma \in \{0,1\}$, $c_0 \leftarrow \mathsf{PE.Enc}(pk, m_0)$, $c_1 \leftarrow \{0,1\}^n$, and $sk' = \mathsf{PE.Puncture}(sk, m_0, m_1)$.

Now, $\mathcal{A}$ can perfectly simulate Hyb_2 and Hyb_3 to $\mathcal{D}$, using c_σ as x_0. If $\sigma = 0$, then $\mathcal{A}$ perfectly simulates Hyb_2. If $\sigma = 1$, then $\mathcal{A}$ perfectly simulates Hyb_3. Thus, $\mathcal{A}$ can break the ciphertext pseudo-randomness by outputting whatever $\mathcal{D}$ outputs. □

Lemma 6.8 *If* PE *satisfies ciphertext randomness, then* $\mathsf{view}(\mathsf{Hyb}_3) \stackrel{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}_4)$.

Proof. In Hyb_4, we change y_0 from $\mathsf{G}(a)$ to a truly random point. The indistinguishability of this change follows from the PRG security of G, since the adversary receives no other information about a. □

Lemma 6.9 *Under the same assumptions of Theorem 6.1,* $\mathsf{view}(\mathsf{Hyb}_4) \overset{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}})$.

Proof of Lemma 6.9. This proof mirrors the proof of Lemma 6.5 and 6.6 (in reverse manner). □

Finally, Lemma 6.3 follows from Lemma 6.5, 6.6, 6.7, 6.8, and 6.9. □

7 Extensions and Variants of Watermarking

7.1 Stronger Unremovability in a Different Model

In this section, we show that if pPRF family $\mathcal{C}$ satisifies a special injective property, then the watermarking scheme for $\mathcal{C}$ in the previous section satisfies the strongest security (Definition 4.3).

Difficulty with Full Security. There is only one part of the above security proof which does not transfer to a "CCA2" version of the unremovability game. This is the claim in the proof of Lemma 6.5, which states that the adversary cannot query the marking oracle on a program $C^{(\iota)}$ such that $H \circ C^{(\iota)}$ agrees with the $H \circ \widetilde{C}$ on a given point $\mathsf{G}(a)$, where $\widetilde{C}$ is the marked challenge program, H is a UOWHF, and a is a random string.

This clearly does not hold for queries made after seeing $\widetilde{C}$. Indeed, $\mathcal{D}$ could then query $\widetilde{C}$ itself. We show that if:

- The inputs to the mark oracle are pPRF keys instead of arbitrary circuits and
- The pPRF family satisfies a strong "key injectivity" property

then unremovability still holds.

In order to achieve the strongest notion of watermarking unremovability, we need to restrict ourselves to marking a pPRF family that satisfies the following key-injectivity condition. We further change the syntax of Mark, so that its input is no longer an arbitrary circuit, but is actually restricted to functions in the family $\mathcal{C}$.

Definition 7.1 (Key-Injective pPRFs)

$$\Pr_{F \leftarrow \mathcal{F}_\lambda}[\exists \alpha, F' \text{ s. t. } F' \neq F \wedge F(\alpha) = F'(\alpha)] \leq \mathsf{negl}(\lambda)$$

In other words this says that with high probability over the choice of F, no other $F' \in \mathcal{F}$ agrees with F *anywhere*. See Appendix C for concrete instantiations. If we assume $\mathcal{C}$ satisfies the injective property in Definition 7.1, then there are only negligible fraction of inputs causes the collision $\widetilde{C}(\alpha) = C^{(\iota+1)}(\alpha)$, that is, Lemma 6.5 still holds.

Corollary 7.2 *Assuming the existence of injective one-way functions, and an indistinguishability obfuscator for all circuits, for all* $\varepsilon(\lambda) = \frac{1}{2} + 1/\mathrm{poly}(\lambda)$*, all message spaces* $\mathcal{M} = \{0,1\}^w$*, all integer functions* $n(\lambda) = \Omega(\lambda)$ *and* $m(\lambda) = \Omega(\lambda)$ *there exists a watermarking scheme with message space* $\mathcal{M}$ *which is* ε*-secure for every key-injective pPRF ensemble* $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *such that functions* C *in* $\mathcal{C}_\lambda$ *map* $\{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$.

Proposition 7.3 *(Informal) Assuming the DDH assumption or LWE assumption, there exist key-injective families of pPRFs.*

7.2 Optimality of $(\frac{1}{2} + \frac{1}{\text{poly}(\lambda)})$-Unremovability

We now show that ε-unremovable message-embedding watermarking is impossible when $\varepsilon \leq \frac{1}{2}$. This is because an adversary can obtain two independent uniformly sampled circuits $\tilde{C}_0$ and $\tilde{C}_1$, each marked with different messages (respectively msg_0 and msg_1). The adversary then outputs a program C^* such that $C^* \cong_{1/2} \tilde{C}_0$ and $C^* \cong_{1/2} \tilde{C}_1$. Since C^* can be generated in a way which treats $\tilde{C}_0$ and $\tilde{C}_1$ symmetrically, we must have

$$\Pr\left[\mathsf{Extract}(C^*) = \mathsf{msg}_0\right] = \Pr\left[\mathsf{Extract}(C^*) = \mathsf{msg}_1\right] \leq \frac{1}{2}.$$

This impossibility clearly holds even in a setting where the adversary is extremely limited in e.g. the number and type of oracle queries he may make.

7.3 Variants

Variant: List Decoding. We note that our construction could also be modified to satisfy ε-unremovability for *any* $\varepsilon = 1/\text{poly}(\lambda)$ by relaxing the correctness requirement on $\mathsf{Extract}$, allowing it to output a (small) list of possible messages rather than a single message. For unremovability, we only require that the correct message appear in the list. For example, in our construction, instead of outputting the "majority value" msg such that $|\{i \ : \ \mathsf{msg} = \mathsf{msg}_i\}|$ is sufficiently large, we could just output all $O(1/\varepsilon^2)$ values of msg_i. By signing the messages with a standard signature scheme, we can in a black-box way ensure that the list of messages output by the detection procedure *only* contain (in addition to the correct message) the messages that were embedded in some watermarked circuit by some previous call to the marking oracle.

Variant: Messageless Watermarking. In the case of *messageless* watermarking, there is no challenge message. Instead, the message space is the singleton set $\mathcal{M} := \{\mathsf{marked}\}$. As a corollary of list-decodable watermarking scheme, we can achieve messageless watermarking with security against any $\varepsilon > 1/\text{poly}(\lambda)$.

Variant: Marking PRFs With Single-Bit Outputs In our construction, we assumed we were marking a pPRF whose outputs was $\{0,1\}^m$ for $m = \Omega(\lambda)$. This assumption on m was not necessary. Indeed, any pPRF family mapping $\{0,1\}^n \to \{0,1\}$ can equally be construed as a pPRF family mapping $\{0,1\}^{n-\log m} \to \{0,1\}^m$, and can be marked as such. In doing so, we incur a loss in parameters. If the watermarking scheme for m-bit outputs satisfied $(1-\varepsilon)$-unremovability, the watermarking scheme for single bit outputs will only satisfy $(1-\frac{\varepsilon}{m})$-unremovability.

Variant: Unforgeability The classic Irish folk tale of "Clever Tom and the Leprechaun" [Kei70] tells of a farmer's son who one day captures a Leprechaun. Compelled to abide by his every request, the Leprechaun leads Tom to the field of bolyawns and indicates the one under which is buried treasure. Before Tom goes to fetch a spade, he ties his red garter round the bolyawn and forbids the Leprechaun from untying it. The story finishes: *"lo an' behould, not a bolyawn in the field, but had a red garther, the very idintical model o' his own, tied about it."* Though the Leprechaun was unable to remove the garter, there was nothing to prevent him from tying identical garters around the neighboring trees, making it impossible for Tom to discover the gold.

In their treatment of watermarking definitions, Hopper et al. [HMW07] define a notion of unforgeability that is dual to unremovability. Intended to prevent attacks like the Leprechaun's, unforgeability requires that the only marked programs circuits that an adversary can produce are functionally similar to circuits marked by a marking oracle. Whereas unremovability requires that a circuit is marked if it is ε-similar to some honestly-marked circuit, unforgeability requires that a circuit is marked *only if* it is δ-similar to an honestly-marked circuit, for some parameter $\delta < \varepsilon$.

Though we have some partial results in this direction, achieving unforgeability and unremovability simultaneously is a challenging and interesting open problem.

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 $\widetilde{C} \cong_\varepsilon C$ for some $\varepsilon = 1 - \mathsf{negl}(\lambda)$. Additionally, the statistical correctness requirement better captures the intuition that any algorithm from which mk and xk are unknown should never see a differing input. Similar reasoning motivated [BGI+12] to adopt the analogous correctness requirement in the context of approximate obfuscation.

8 Watermarking Other Cryptographic Primitives

In this section, we show how to use our pPRF watermarking scheme to watermark an encryption scheme and a signature scheme. That is, no p.p.t. adversary can produce a program which decrypts most ciphertexts or signs most messages, without revealing a given watermarked message.

Intuitively, this follows from the fact that cryptosystems and signature schemes exist for which decryption (respectively signature generation) are nothing more than a pPRF evaluation [SW14]. However, in order for this to be valid, we must prove that a *marked* pPRF in our watermarking scheme remains puncturable. Unfortunately, it doesn't seem to be. The problem is that for a pPRF C, $\mathsf{Mark}(C)$ in our construction internally contains two copies of C: one for generating outputs, and another for checking whether an input is a mark point. Replacing C by $C\{x\}$ will change the functionality of $\mathsf{Mark}(C)$ at many points iff x is in the image of the PRG G.

There are work-arounds, however. In the [SW14] construction of PKE, the proof of security only requires a weaker form of puncturability, which our construction *does* achieve. Namely, the PRF only needs to be puncturable at a random point x. Since a random point is with high probability not in the image of G, replacing C by $C\{x\}$ will only change the functionality of $\mathsf{Mark}(C)$ at x. The signature scheme described in [SW14] requires a pPRF which is puncturable at an arbitrary point, but can be easily modified for a pPRF puncturable at a random point. The idea is to introduce a random offset, so that any message corresponds to PRF evaluation at a random point. This offset can even be public.

Below, we formalize what it means to have a watermarkable public-key encryption scheme and a watermarkable signature scheme.

Definition 8.1 (Watermarkable Public-Key Encryption) A watermarkable public-key encryption scheme is a tuple of algorithms $(\mathsf{WM.Gen}, \mathsf{PKE.Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Extract})$ satisfying the following properties.

Correctness:

For every mark mark, we have

$$\Pr\left[\mathsf{Extract}(xk, \mathsf{Dec}_{sk}) \neq \mathsf{mark} \;\middle|\; \begin{array}{l}(xk, mk) \leftarrow \mathsf{WM.Gen}(1^\lambda) \\ (pk, sk) \leftarrow \mathsf{PKE.Gen}(1^\lambda, mk, \mathsf{mark})\end{array}\right] \leq \mathsf{negl}(\lambda)$$

where Dec_{sk} is any circuit computing $\mathsf{Dec}(sk, \cdot)$. For every mark mark and every bit $b \in \{0,1\}$, we have

$$\Pr\left[\mathsf{Dec}(sk, c) \neq b \;\middle|\; \begin{array}{l}(xk, mk) \leftarrow \mathsf{WM.Gen}(1^\lambda) \\ (pk, sk) \leftarrow \mathsf{PKE.Gen}(1^\lambda, mk, \mathsf{mark}) \\ c \leftarrow \mathsf{Enc}(pk, b)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

IND-CPA Security: For every mark mark and every p.p.t. algorithm $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A}(1^\lambda, pk, xk, mk, c_b) = b \;\middle|\; \begin{array}{l}(xk, mk) \leftarrow \mathsf{WM.Gen}(1^\lambda) \\ (pk, sk) \leftarrow \mathsf{PE.Gen}(1^\lambda, mk, \mathsf{mark}) \\ b \leftarrow \{0,1\} \\ c_b \leftarrow \mathsf{Enc}(pk, b)\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

ε-Unremovability: For every p.p.t. adversary $\mathcal{A}$ and every mark mark,

$$\Pr\left[\begin{array}{l|l} C^* \cong_\varepsilon \mathsf{Dec}_{sk} \wedge & (xk, mk) \leftarrow \mathsf{WM.Gen}(1^\lambda) \\ \mathsf{Extract}(xk, C^*) \neq \mathsf{mark} & (pk, sk) \leftarrow \mathsf{PE.Gen}(1^\lambda, mk, \mathsf{mark}) \\ & C^* \leftarrow \mathcal{A}(1^\lambda, sk, xk) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Theorem 8.2 *(Informal) Assuming the existence of indistinguishability obfuscation and injective one-way functions, there is a watermarkable public-key encryption scheme.*

We can define a watermarkable signature scheme similarly.

Definition 8.3 (Watermarkable Signature Scheme) A watermarkable signature scheme is a tuple of p.p.t. algorithms $(\mathsf{WM.Gen}, \mathsf{SIG.Gen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Extract})$ satisfying the following properties.

Correctness: For every message msg and every mark mark,

$$\Pr\left[\begin{array}{l|l} \mathsf{Vrfy}(pk, \mathsf{msg}, \sigma) \neq 1 \vee & (xk, mk) \leftarrow \mathsf{WM.Gen}(1^\lambda) \\ \mathsf{Extract}(xk, \mathsf{Sign}_{sk}) \neq \mathsf{mark} & (pk, sk) \leftarrow \mathsf{SIG.Gen}(1^\lambda, mk, \mathsf{mark}) \\ & \sigma \leftarrow \mathsf{Sign}(sk, \mathsf{msg}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

(Selective) Existential Unforgeability Under Chosen Message Attack: For every message msg^*, every mark mark, and every p.p.t. algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, the following experiment outputs 1 with negligible probability.

$\mathsf{Exp}^{\mathsf{wm\text{-}sel}}_{\mathcal{A}}(\lambda)$**:**

1. $\mathsf{msg}^* \leftarrow \mathcal{A}_0(1^\lambda)$
2. $(xk, mk) \leftarrow \mathsf{WM.Gen}(1^\lambda)$
3. $(pk, sk) \leftarrow \mathsf{SIG.Gen}(1^\lambda, mk, \mathsf{mark})$
4. $\sigma^* \leftarrow \mathcal{A}_1^{\mathsf{Sign}_{\mathsf{msg}^*}(sk,\cdot)}(pk, xk)$, where $\mathsf{Sign}_{\mathsf{msg}^*}(sk, \cdot)$ is an oracle that signs any message except for msg^*. That is,

$$\mathsf{Sign}_{\mathsf{msg}^*}(sk, m) = \begin{cases} \bot & \text{if } m = \mathsf{msg}^* \\ \mathsf{Sign}(sk, m) & \text{otherwise} \end{cases}$$

5. If $\mathsf{Vrfy}(pk, \mathsf{msg}^*, \sigma^*) = 1$, output 1. Else 0.

ε-Unremovability: For every p.p.t. adversary $\mathcal{A}$ and every mark mark,

$$\Pr\left[\begin{array}{l|l} C^* \cong_\varepsilon \mathsf{Sign}_{sk} \wedge & (xk, mk) \leftarrow \mathsf{WM.Gen}(1^\lambda) \\ \mathsf{Extract}(xk, C^*) \neq \mathsf{mark} & (pk, sk) \leftarrow \mathsf{SIG.Gen}(1^\lambda, mk, \mathsf{mark}) \\ & C^* \leftarrow \mathcal{A}(1^\lambda, sk, xk) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Theorem 8.4 *(Informal) Assuming the existence of indistinguishability obfuscation and injective one-way functions, there is a watermarkable signature scheme.*

Remark 8.5 We remark that stronger notions of watermarkable public-key encryption and watermarkable signature schemes are certainly definable, but we omit these as they are more complex and not central to our work.

9 The Limits of Watermarking

A natural question is whether there are families of functions that for which there does not exist any watermarking scheme. Barak et al. [BGI+01] observed that general-purpose indistinguishability obfuscation rules out a notion of watermarking that *exactly* preserves functionality, but not watermarking schemes that change functionality on even a negligible fraction of the domain (as in section 6). In this section, we demonstrate that some notion of *non-black-box* learnability implies that a family of functions is unwatermarkable. We demonstrate that there exist PRF families that cannot be watermarked (assuming only the existence of one-way functions), and that any family that is learnable with membership queries [KL93] is not watermarkable.

9.1 Impossibilities for statistical correctness

In this section, we discuss a number of conditions sufficient to prove that a family of circuits cannot even be watermarked – even for a significantly weakened form of unremovability. We modify the unremovability game (Definition 4.3): the adversary has no marking oracle, has neither a public extraction key nor an extraction oracle, and is not allowed to choose the message to be embedded in the challenge.We leave the syntax, statistical correctness, extraction correctness, and meaningfulness requirements of the watermarking definition (Definitions 4.1 and 4.2) are unchanged. In Section 9.2, we relax the statistical correctness condition.

Definition 9.1 (Weak ε-Unremovability Game) The game $\mathsf{Exp}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon)$ is defined as follows.

1. The challenger generates $(xk, mk) \leftarrow \mathsf{Gen}(1^\lambda)$
2. The challenger chooses a message $\mathsf{msg} \in \mathcal{M}_\lambda$ arbitrarily, samples a circuit $C \leftarrow \mathcal{C}_\lambda$ uniformly at random and gives to the adversary $\widetilde{C} \leftarrow \mathsf{Mark}(mk, C, \mathsf{msg})$.
3. Finally, the adversary outputs a circuit C^*. If it holds $C^* \cong_\varepsilon \widetilde{C} \wedge \mathsf{Extract}(xk, C^*) \neq \mathsf{msg}$ then the experiment outputs 1, otherwise 0.

Definition 9.2 (ε-Waterproof) Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a circuit ensemble. We say that $\mathcal{F}$ is *ε-waterproof* if there does not exist an weak ε-unremovable watermarking scheme for $\mathcal{F}$.

Informally, if a function family is *non-black-box* learnable given an approximate circuit implementation (corresponding the the challenge watermarked circuit), then the family is waterproof. More formally, consider a family of circuits $\mathcal{F}_\lambda$ and some parameter $\rho = \rho(\lambda) \in [0, 1]$. The learning algorithm will be given an (arbitrary) circuit g that ρ-approximates F, for a uniformly sampled circuit $F \leftarrow \mathcal{F}_\lambda$ from the family. The (randomized) learner will then output some "hypothesis" circuit h. If h is sufficiently close to F, then the learner can be used to reconstruct an unmarked circuit given a watermarking challenge. We conclude that the family $\mathcal{F}$ is waterproof.

We emphasize that we are interested in *non-black-box learning* in which the learning algorithm gets an (approximate) implementation of the function being learned. This is in contrast to the typical computational learning setting.

For the sake of clarity, we now define all the variants of learning we will consider. It may be best to read the definitions individually when required by the discussion that follows.

Definition 9.3 (Non-black-box Learnable Families) Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a circuit ensemble where each family $\mathcal{F}_\lambda = \{F\}$. Let $\rho = \rho(\lambda) \in [0, 1]$. We say a distribution over circuits $\mathcal{C}_F$ *ρ-strongly approximates* $F \in \mathcal{F}_\lambda$ if for all x,

$$\Pr_{C \leftarrow \mathcal{C}_F}[C(x) \neq F(x)] \leq \rho.$$

Let $\{\mathcal{C}_F\}_{F\in\mathcal{F}_\lambda}$ be any collection of ρ-strongly approximating distributions for the circuits $F \in \mathcal{F}_\lambda$.

Robustly Learnable:[9] We say that $\mathcal{F}$ is *ρ-robustly learnable* if there exists an efficient algorithm L outputting a circuit h, such that for all large enough $\lambda \in \mathbb{N}$, random $F \leftarrow \mathcal{F}_\lambda$, and random circuit $C \leftarrow \mathcal{C}_F$ (where $\mathcal{C}_F$ ρ-strongly approximates F):

$$\Pr[h \equiv F \mid h \leftarrow L(C, 1^\lambda)] \text{ is non-negligible.}$$

We say that $\mathcal{F}$ is *robustly* learnable if it is ρ-robustly learnable for any negligible function $\rho(\lambda)$.

Properly Learnable:[10] Additionally, we say that $\mathcal{F}$ is *properly* learnable if for every function $F \in \mathcal{F}_\lambda$, and random $C \leftarrow \mathcal{C}_F$:

$$\Pr[L(C, 1^\lambda) = F] \text{ is non-negligble.}$$

Implementation Independently Learnable: Let $\mathcal{C}_F^1$ and $\mathcal{C}_F^2$ be two distributions that ρ-strongly approximate F. We say that L is *implementation independent* if for all $F \in \mathcal{F}_\lambda$ and for any two distributions $\mathcal{C}_F^1$ and $\mathcal{C}_F^2$ that ρ-strongly approximate F, the distributions $(L(C_1, 1^\lambda) \ : \ C_1 \leftarrow \mathcal{C}_F^1)$ and $(L(C_2, 1^\lambda) \ : \ C_2 \leftarrow \mathcal{C}_F^2)$ are computationally indistinguishable.

ε-Approximately Learnable: A weaker condition than the above, we say that $\mathcal{F}$ is *ε-approximately* learnable if instead, for all F and for random $C \leftarrow \mathcal{C}_F$:

$$\Pr[h \cong_\varepsilon F \mid h \leftarrow L(C, 1^\lambda)] \text{ is non-negligible.}$$

As a warm up, we begin with a very strong notion of learnability, in which the learning algorithm can not only output a hypothesis h which agrees with F on all inputs, but output the circuit F itself.

Proposition 9.4 *If $\mathcal{F}$ is robustly, properly learnable, then $\mathcal{F}$ is ε-waterproof for every $\varepsilon \in [0, 1]$.*

Proof. Given a watermarking scheme for the family $\mathcal{F}$, let $\mathcal{C}_F = \{\mathsf{Mark}(mk, F) \ : \ (xk, mk) \leftarrow \mathsf{Gen}(1^\lambda)\}$. There exists some negligible function $\rho(\lambda)$ such that $\mathcal{C}_F$ ρ-strongly approximates F for all circuits $F \in \mathcal{F}$, by the statistical correctness property. Suppose $\mathcal{F}$ is ρ-robustly, properly learnable with learning algorithm L. Given a challenge marked program $\widetilde{F} \leftarrow \mathsf{Mark}(mk, F)$, evaluate $h \leftarrow L(\widetilde{F}, 1^\lambda)$. With noticeable probability, $h = F$. If $\mathsf{Extract}(xk, F) = \mathsf{unmarked}$ with any noticeable probability, unremovability is violated. On the other hand, if $\mathsf{Extract}(xk, F) \neq \mathsf{unmarked}$ with any noticeable probability, then meaningfulness is violated. □

Surprisingly, this proposition is also enough to construct a PRF family that is waterproof.

Proposition 9.5 ([BGI+12]) *Assuming one-way functions exist, there exists a pseudorandom function family $\mathcal{F}$ that is robustly, properly (non-black-box) learnable.*

[8] The strong-approximation assumption on the distribution of the approximate implementation C arises from the statistical correctness requirement of Definition 4.2. Note that statistical correctness guarantees that for $F \in \mathcal{F}_\lambda$, the distribution $(\widetilde{F} \leftarrow \mathsf{Mark}(mk, F) \ : \ mk \leftarrow \mathsf{Setup}(1^\lambda))$ strongly-approximates F for some negligible function $\rho(\lambda)$.

[9] This is somewhat analogous to the notion of error-tolerance in computational learning [KL93], but in the non-black-box setting.

[10] This is stronger than simply requiring that $h \in \mathcal{F}_\lambda$. In particular, it implies that for every $F \in \mathcal{F}_\lambda$, there are only polynomially-many $F' \in \mathcal{F}_\lambda$ such that $F' \cong_{\rho/2} F$.

Proof. In [BGI+12], the authors extend the impossibility of virtual-black box obfuscation to a notion of approximate obfuscation, where for every input x, the obfuscated circuit $\mathcal{O}(C)$ is required to agree with C on x with high probability over $\mathcal{O}$. They construct a "strongly unobfuscatable circuit ensemble" [BGI+12, Theorem 4.3], which has precisely we need: there exists an algorithm L which given any strongly approximate implementation of $F \in \mathcal{F}_\lambda$, efficiently outputs F with high probability. Additionally, their techniques can be extended to yield a family of strongly unobfuscatable PRFs [BGI+12, Section 4.2]. □

Corollary 9.6 *Assuming one-way functions, there exists a pseudorandom function family $\mathcal{F}$ which for every $\varepsilon \in [0,1]$ is ε-waterproof.*

Improper versus proper learning. What if the family is not properly learnable: instead of outputting F itself, the learning algorithm $L(C)$ can only output a circuit h that was functionally equivalent to F? One might think that this is indeed sufficient to prove Proposition 9.4, but the proof encounters a difficulty.

In the proper-learning setting, it was possible to sample a circuit which for which $\mathsf{Extract}(xk, C) \neq \mathsf{unmarked}$ *independently of* mk, simply by picking $F \leftarrow \mathcal{F}$. In the improper-learning setting, we only know how to sample from this distribution by evaluating $L(\widetilde{F})$ *on the marked program* $\widetilde{F}$. To violate meaningfulness, we need to construct C such that $\mathsf{Extract}(xk, C) \neq \mathsf{unmarked}$ with noticeable probability over both Gen and $\mathsf{Extract}$, suggesting that we should find such a C independently of mk.

To get around this issue, we consider families that are learnable with *implementation independence*; that is, for any strong approximate implementations $\mathcal{C}_F^1$ and $\mathcal{C}_F^2$ of F, the distributions $(L(C_1, 1^\lambda) : C_1 \leftarrow \mathcal{C}_F^1)$ and $(L(C_2, 1^\lambda) : C_2 \leftarrow \mathcal{C}_F^2)$ are computationally indistinguishable.[11]

Approximate versus exact learning. In the preceding, we required that an algorithm learning a family $\mathcal{F}$ is able to exactly recover the functionality F. What can we prove if $h = L(C, 1^\lambda)$ is only required to ε-approximate the original function F? For this case, the proof generalizes quite naturally to show that a family is ε-waterproof.

Proposition 9.7 *If $\mathcal{F}$ is robustly, ε-approximately learnable with implementation independence, then $\mathcal{F}$ is ε-waterproof.*

Proof. As before, we run the learner on the challenge program to get $h = L(\widetilde{F}, 1^\lambda)$. The circuit h is an ε-approximation of F with non-negligible probability. If $\mathsf{Extract}(xk, h) = \mathsf{unmarked}$ with noticeable probability, then unremovability is violated. Therefore, it must be the case that $\mathsf{Extract}(xk, h) \neq \mathsf{unmarked}$ with high probability (even conditioning on the case when h is an ε-approximation).

Observe that for any $F \in \mathcal{F}$, the singleton distribution $\{F\}$ is a strongly approximate implementation of F. To complete the above proof, consider $h' \leftarrow L(F, 1^\lambda)$ for random (unmarked) F (rather than on the marked $\widetilde{F}$). Implementation independence of L guarantees that the distributions of h and h' are indistinguishable and thus for general xk, $\mathsf{Extract}(xk, h') \neq \mathsf{unmarked}$ with high probability. □

Corollary 9.8 *Any family that is (improperly, approximately) learnable with membership queries [KL93] is ε-waterproof for any non-negligible ε.*

Proof. An MQ learning algorithm L can be simulated with any approximate implementation C of F. Because $C \leftarrow \mathcal{C}_F$ for $\mathcal{C}_F$ a strongly approximating implementation of F, both C and F will agree on all the queries made by the MQ learner L with high probability. The views of L are statistically close for every approximating distribution $\mathcal{C}$, implying implementation independence. □

[11] Weaker notions likely suffice because meaningfulness only requires noticeable probability of falsely extracting, whereas this argument gives us a high probability. We consider this input independence notion because it is a simple, natural and, as we will see, powerful case.

Additionally, this proposition captures the impossibility of exact watermarking originally presented in [BGI+12].

Corollary 9.9 *Assuming the existence of indistinguishability obfuscation, exact watermarking schemes are impossible.*

Proof. Indistinguishability obfuscation implies a 0-robust, exact, implementation independent learning algorithm for all polynomial-sized circuits, where L simply obfuscates its input.[12] □

9.2 Impossibilities for weak statistical correctness

It is possible to prove similar impossibility results even if we weaken the statistical correctness property of the watermarking scheme to only require that $\mathsf{Mark}(mk, C, \mathsf{msg})$ changes functionality at few points, but make no restrictions as to the distributions of these errors. We prove that for this weak setting (1) there exist waterproof PRFs and (2) PAC-learnable families are waterproof. The main difficulty in this setting is that Mark may now change the functionality on adversarially-chosen points, preventing a straightforward adaptation of Proposition 9.5 and Corollary 9.8.

We now consider watermarking schemes that satisfy only weak statistical correctness:

Definition 9.10 (Weak Statistical Correctness:) There is a negligible function $\nu(\lambda)$ such that for any circuit $C \in \mathcal{C}_\lambda$, and any message $\mathsf{msg} \in \mathcal{M}_\lambda$:

$$\mathsf{Mark}(mk, C, \mathsf{msg}) \cong_\nu C$$

We can adapt the learning definitions of the prequel to this weaker notion of statistical correctness. The main change in the definitions is that we no longer require strongly-approximating distributions of circuits $\mathcal{C}_F$ for a function F; an arbitrary circuit $C \cong_\rho F$ that is close to F suffices. This is a strictly more general setting.

Definition 9.11 (Learning from arbitrary approximate implementation) For each of the learning definitions in Definition 9.3, we say that the learning algorithm works with *arbitrary approximate implementation* if instead of requiring a ρ-strongly approximate distribution $\mathcal{C}_F$ for F, the learning algorithm will work for arbitrary $C \cong_\rho F$.

Modifying the definition of waterproof to require that the watermarking scheme only satisfies weak statistical correctness, both Proposition 9.7 and 9.4 still hold in this setting.

Though membership query-learnability no longer suffices for waterproof-ness, PAC learnability does.

Corollary 9.12 *Any family that is (improperly) PAC learnable [Val84] is ε-waterproof (with weak statistical correctness) for any non-negligible ε.*

Proof. An PAC learning algorithm L can be simulated with random queries to arbitrary approximate implementation C of F. Because $C \cong_\rho F$, both C and F will agree on all the random queries seen by L with high probability. The views of L are statistically close for every C, implying implementation independence. □

The main technical contribution of this section is the following PRF construction (the proof is in Appendix B:

[12] Observed by Nir Bitansky.

Theorem 9.13 *Assuming one-way functions, there exists a pseudorandom function family $\mathcal{F}$ that is robustly, ε-approximately learnable with implementation independence* from arbitrary approximate implementations.

Corollary 9.14 *Assuming one-way functions, there exists a pseudorandom function family $\mathcal{F}$ which is ε-waterproof (with weak statistical correctness) for any non-negliglbe ε.*

10 Conclusions

We showed how to watermark various cryptographic capabilities: PRF evaluation, ciphertext decryption, and message signing. For all of these, there is a natural and secret "true functionality" f_k that we would like to mark. Given a message msg, we can distribute a "marked" circuit C which closely approximates f_k. Given C, any efficiently findable circuit C^* which even loosely approximates f_k must also contain msg. Furthermore, in our scheme, the procedure for extracting msg is entirely public-key. We show that unmarked circuits cannot approximate the marked capability to within an approximation factor of $\varepsilon = \frac{1}{2} + 1/\mathrm{poly}$ for any poly. If we allow *list decoding*, namely allow the extraction procedure to output a polynomial-sized list of messages containing msg, then ε can be lowered to $1/\mathrm{poly}$.

There are several directions for further research. First, one could explore the connection between obfuscation and watermarking to see whether some form of obfuscation is *necessary* to achieve watermarking or if one can come up with constructions that avoid obfuscation. 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. In the setting where the marking oracle takes keys as input, this kind of watermarking appears plausible. As usual with obfuscation, there is a heuristic construction which obfuscates the secret-key marking procedure to generate a public marking key. Proving such a scheme secure by only relying on iO (as opposed to VBB) appears to require significantly new techniques. Finally, watermarking schemes for richer classes of programs seem to be beyond the reach of our techniques, but would be of obvious interest.

References

[AKV03] André Adelsbach, Stefan Katzenbeisser, and Helmut Veith. Watermarking schemes provably secure against copy and ambiguity attacks. In Moti Yung, editor, *Proceedings of the 2003 ACM workshop on Digital rights management 2003, Washington, DC, USA, October 27, 2003*, pages 111–119. ACM, 2003.

[BFP+15] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In Dodis and Nielsen [DN15], pages 31–60.

[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.

[BGI14a] 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.

[BGI14b] 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.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428. Springer, 2013.

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 241–250, 2013.

[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.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Dodis and Nielsen [DN15], pages 1–30.

[BW13a] 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.

[BW13b] 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.

[CHV15] Aloni Cohen, Justin Holmgren, and Vinod Vaikuntanathan. Publicly verifiable software watermarking. *IACR Cryptology ePrint Archive*, 2015:373, 2015.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[DN15] Yevgeniy Dodis and Jesper Buus Nielsen, editors. *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*. Springer, 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.

[Kei70] Thomas Keightley. *The Fairy Mythology: Illustrative of the Romance and Superstition of Various Countries.* 1870. Retrieved from: http://www.sacred-texts.com/neu/celt/tfm/: 2 November 2015.

[KL93] Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993.

[KPTZ13a] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 669–684. ACM, 2013.

[KPTZ13b] 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.

[KVH00] M. Kutter, S. Voloshynovskiy, and A. Herrigel. The watermark copy attack. In *Proceedings of the SPIE, Security and Watermarking of Multimedia Contents II*, volume 3971, pages 371–379, 2000.

[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.

[NW15] Ryo Nishimaki and Daniel Wichs. Watermarking cryptographic programs against arbitrary removal strategies. *IACR Cryptology ePrint Archive*, 2015:344, 2015.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394. ACM, 1990.

[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.

[Val84] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

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

A Construction and Security Proofs of Puncturable Encryption

We provide a construction of the puncturable encryption defined in section 5.

A.1 Construction

We construct a puncturable encryption scheme in which the length n of ciphertexts is 12 times the length ℓ of plaintexts. Our construction utilizes the following ingredients:

- A length-doubling $\mathsf{PRG} : \{0,1\}^{\ell} \to \{0,1\}^{2\ell}$
- A family of injective pPRFs (See Definition 3.6) $\{\mathcal{F}_\lambda : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}\}$. [13]
- A family of pPRFs $\{G_\lambda : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}\}$.
- An injective bit-commitment Com using randomness in $\{0,1\}^{9\ell}$, which can in fact be constructed by an injective one-way function. We only use this in our security proof.

Construction A.1 (Puncturable Encryption Scheme PE)

$\mathsf{Gen}(1^\lambda)$**:** *Sample functions $F \leftarrow \mathcal{F}_\lambda$ and $G \leftarrow \mathcal{G}_\lambda$, generates pk as the $i\mathcal{O}$-obfuscation of the program E in Figure 7, and returns $(pk, sk) := (i\mathcal{O}(E), D)$, where sk is the (un-obfuscated) program D in Figure 8.*

$\mathsf{Puncture}(sk, c_0, c_1)$**:** *Output sk', where sk' is the $i\mathcal{O}$-obfuscation of the program D' described in Figure 9, that is, $sk' := i\mathcal{O}(D')$.*

$\mathsf{Enc}(pk, m)$**:** *Take $m \in \{0,1\}^{\ell}$, sample $r \leftarrow \{0,1\}^{\ell}$, and outputs $c \leftarrow pk(m, r)$.*

[13] As in [SW14], any puncturable PRF family from $\{0,1\}^k \to \{0,1\}^{2k+\omega(\log\lambda)}$ can be made statistically injective (with no additional assumptions) by utilizing a family of pairwise-independent hash functions.

$\mathsf{Dec}(sk, c)$**:** *Take* $c \in \{0,1\}^{12\ell}$ *and returns* $m := sk(c)$.

The size of the programs is appropriately padded to be the maximum size of all modified programs, which will appear in the security proof.

Remark A.2 We note that in all of our obfuscated programs (including the hybrids), whenever α_i or β_i or γ_i for $i \in \{0,1\}$ are treated symmetrically, then we can and do store them in lexicographical order. A random ordering would also suffice for security.

Constants: Injective pPRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, pPRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$
Inputs: $m \in \{0,1\}^{\ell}, r \in \{0,1\}^{\ell}$

1. Compute $\alpha = \mathsf{PRG}(r)$.
2. Compute $\beta = F(\alpha \| m)$.
3. Compute $\gamma = G(\beta) \oplus m$.
4. Output (α, β, γ).

Figure 7: Encryption Program E (pre-obfuscation)

Constants: Injective pPRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, pPRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$
Inputs: $c = (\alpha \| \beta \| \gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.

1. Compute $m = G(\beta) \oplus \gamma$.
2. If $\beta = F(\alpha \| m)$, output m.
3. Else output $\perp$.

Figure 8: Decryption Program D

Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, injective pPRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, and pPRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$
Inputs: $c = (\alpha \| \beta \| \gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.

1. If $c \in \{c_0, c_1\}$, output $\perp$.
2. Compute $m = G(\beta) \oplus \gamma$.
3. If $\beta = F(\alpha \| m)$, output m.
4. Else output $\perp$.

Figure 9: Punctured Decryption Program D' at $\{c_0, c_1\}$ (pre-obfuscation)

Correctness and Punctured Correctness. Correctness follows from the fact that indistinguishability obfuscation exactly preserves functionality, and observing in the punctured case that sk' is defined to be functionally equivalent to sk except on inputs in $\{c_0, c_1\}$.

Sparseness. Sparseness follows from, for example, the length-doubling PRG; most values of α are not in the image of PRG.

Table 2: An overview of hybrid distributions

Hybrid	α_0	β_0	γ_0	$pk := i\mathcal{O}$ of below	$sk' := i\mathcal{O}$ of below
REAL_0	$\mathsf{PRG}(t)$	$F(\alpha_0 \Vert m^*)$	$G(\beta_0) \oplus m^*$	E	D'
Hyb_1	random	$F(\alpha_0 \Vert m^*)$	$G(\beta_0) \oplus m^*$	E	D'
Hyb_2	random	$F(\alpha_0 \Vert m^*)$	$G(\beta_0) \oplus m^*$	$E\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*\}$	$D'_2\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*\}$
Hyb_3	random	random	$G(\beta_0) \oplus m^*$	$E\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*\}$	$D'_3\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*\}$
Hyb_4	random	random	$G(\beta_0) \oplus m^*$	$E\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*, \beta_0, \beta_1\}$	$D'_4\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*, \beta_0, \beta_1\}$
Hyb_5	random	random	random	$E\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*, \beta_0, \beta_1\}$	$D'_4\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*, \beta_0, \beta_1\}$
RAND	random	random	random	E	D'

A.2 Ciphertext Pseudorandomness

Theorem A.3 *If $\mathcal{F}$ is an injective pPRF family, $\mathcal{G}$ is a pPRF family,* PRG *is a pseudorandom generator,* Com *is a injective bit-commitment function, and $i\mathcal{O}$ is a secure iO, then the* PE *scheme above satisfies the ciphertext pseudorandomness.*

Proof. We give a sequence of main hybrid distributions Hyb_1 through Hyb_5. The goal of the hybrids to reach a game in which the challenge encryption c_0 and the random ciphertext c_1 are treated symmetrically in pk and sk', and in which both are sampled uniformly at random by the challenger. We proceed by iteratively replacing pieces of c_0 by uniformly random values, puncturing F and G as necessary. We give an overview of the hybrids in Table 2.

REAL_b**:** The real distribution is defined by the real security game:

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$ to the challenger.
2. The challenger does the following:
   (a) Samples an injective pPRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$ and pPRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$.
   Samples $t \leftarrow \{0,1\}^{\ell}$,
   $\alpha_0 = \mathsf{PRG}(t) \in \{0,1\}^{2\ell}$,
   $\beta_0 = F(\alpha_0 \Vert m^*)$,
   $\gamma_0 = G(\beta_0) \oplus m^*$.
   Let $c_0 = \alpha_0 \Vert \beta_0 \Vert \gamma_0$.
   (b) Samples $c_1 \leftarrow \{0,1\}^{12\ell}$.
   Parse $c_1 = \alpha_1 \Vert \beta_1 \Vert \gamma_1$.
   (c) Generates pk as the $i\mathcal{O}$-obfuscation of Figure 7 and sk' as the $i\mathcal{O}$-obfuscation of Figure 9.
   (d) Samples $b \leftarrow \{0,1\}$ and sends the following to $\mathcal{A}$:

$$\begin{array}{ll} (c_0, c_1, pk, sk') & \text{if } b = 0 \\ (c_1, c_0, pk, sk') & \text{if } b = 1 \end{array}$$

3. The adversary outputs b' and wins if $b = b'$.

That is, REAL_0 is (c_0, c_1, pk, sk') and REAL_1 is (c_1, c_0, pk, sk').

RAND**:** Before we define several hybrid distributions, we define an intermediate hybrid between REAL_0 and REAL_1. We define RAND as (r', c_1, pk, sk') where r' is a uniformly random element in $\{0,1\}^{12\ell}$.

Hyb_1**:** We sample uniformly random $\alpha_0 \leftarrow \{0,1\}^{2\ell}$ for c_0.

Hyb_2**:** We puncture programs E and D' at $\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*\}$ by puncturing F at $\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*\}$. These modified programs $E\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*\}$ and $D'\{\alpha_0 \Vert m^*, \alpha_1 \Vert m^*\}$ are described in Figure 10 and 11, respectively where $\hat{\beta} = F'(\alpha_1 \Vert m^*)$ and $\hat{\gamma} = G(\hat{\beta}) \oplus m^*$.

Encryption Program $E\{\alpha_0\|m^*, \alpha_1\|m^*\}$

Constants: Punctured $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$ and (not-punctured) G.

Inputs: $m \in \{0,1\}^\ell, r \in \{0,1\}^\ell$

1. Compute $\alpha = \mathsf{PRG}(r)$.
2. Compute $\beta = F'(\alpha\|m)$.
3. Compute $\gamma = G(\beta) \oplus m$.
4. Output (α, β, γ).

Figure 10: Program $E\{\alpha_0\|m^*, \alpha_1\|m^*\}$ (pre-obfuscation)

Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$, G, and the values α_0, α_1, $\hat{\beta}$, $\hat{\gamma}$, m^*.

Inputs: $c = (\alpha\|\beta\|\gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^\ell$.

1. If $\alpha = \alpha_1$ and $\beta = \hat{\beta}$ and $\gamma = \hat{\gamma}$, output m^*.
2. If $c \in \{c_0, c_1\}$, output $\perp$.
3. Compute $m = G(\beta) \oplus \gamma$.
4. If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
5. If $\beta = F'(\alpha\|m)$, output m.
6. Else output $\perp$.

Figure 11: Punctured Program $D'_2\{\alpha_0\|m^*, \alpha_1\|m^*\}$ in Hyb_2 (pre-obfuscation)

Hyb_3**:** We sample uniformly random $\beta_0, \hat{\beta} \leftarrow \{0,1\}^{9\ell}$ for c_0 and slightly modify program $D'_2\{\alpha_0\|m^*, \alpha_1\|m^*\}$. The modified program $D'_3\{\alpha_0\|m^*, \alpha_1\|m^*\}$ is in Figure 12.

Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$, G, and the values α_0, α_1.

Inputs: $c = (\alpha\|\beta\|\gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^\ell$.

1. Removed branch.
2. If $c \in \{c_0, c_1\}$, output $\perp$.
3. Compute $m = G(\beta) \oplus \gamma$.
4. If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
5. If $\beta = F'(\alpha\|m)$, output m.
6. Else output $\perp$.

Figure 12: Punctured Program $D'_3\{\alpha_0\|m^*, \alpha_1\|m^*\}$ in Hyb_3 (pre-obfuscation)

Hyb_4**:** We puncture programs E and D' at $\{\alpha_0\|m^*, \alpha_1\|m^*, \beta_0, \beta_1\}$ by puncturing G at $\{\beta_0, \beta_1\}$. These modified programs are described in Figure 13 and 14.

Hyb_5**:** We sample uniformly random $\gamma_0 \leftarrow \{0,1\}^\ell$ for c_0.

Our goal is to prove $\mathsf{REAL}_0 \overset{c}{\approx} \mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2 \overset{c}{\approx} \mathsf{Hyb}_3 \overset{c}{\approx} \mathsf{Hyb}_4 \overset{c}{\approx} \mathsf{Hyb}_5 \overset{c}{\approx} \mathsf{RAND}$ since we can prove

Constants: Punctured $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$ and punctured $G' = G\{\beta_0, \beta_1\}$.
Inputs: $m \in \{0,1\}^\ell, r \in \{0,1\}^\ell$

1. Compute $\alpha = \mathsf{PRG}(r)$.
2. Compute $\beta = F'(\alpha\|m)$.
3. Compute $\gamma = G'(\beta) \oplus m$.
4. Output (α, β, γ).

Figure 13: Encryption Program $E\{\alpha_0\|m^*, \alpha_1\|m^*, \beta_0, \beta_1\}$ (pre-obfuscation)

Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$, and punctured $G' = G\{\beta_0, \beta_1\}$, and the values α_0, α_1, β_0, β_1, m^*.
Inputs: $c = (\alpha\|\beta\|\gamma)$, where $\alpha \in \{0,1\}^{2\ell}$ and $\beta \in \{0,1\}^{9\ell}$.

1. Removed branch.
2. If $\beta \in \{\beta_0, \beta_1\}$, output $\perp$.
3. Compute $m = G'(\beta) \oplus \gamma$.
4. If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
5. If $\beta = F'(\alpha\|m)$, output m.
6. Else output $\perp$.

Figure 14: Punctured Program $D'_4\{\alpha_0\|m^*, \alpha_1\|m^*\}$ in Hyb_4 (pre-obfuscation)

$\mathsf{RAND} \overset{\mathsf{c}}{\approx} \mathsf{REAL}_1$ in the reverse manner and it means $\mathsf{REAL}_0 \overset{\mathsf{c}}{\approx} \mathsf{REAL}_1$.

Lemma A.4 *If* PRG *is a pseudorandom generator, then* $\mathsf{Hyb}_0 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_1$.

Proof of Lemma A.4. These distributions are indistinguishable due to the pseudorandomness of PRG. □

Lemma A.5 *If $\mathcal{F}$ is an injective pPRF family and iO is a secure iO, then* $\mathsf{Hyb}_1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2$.

Proof of Lemma A.5. To prove this lemma, we define auxiliary hybrids.

Hyb_1^1**:** We alter the generation of pk. We puncture F at $\alpha_0\|m^*$ and $\alpha_1\|m^*$ and use it for pk. That is, we use $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$ to generate the encryption program E.

Hyb_1^2**:** We modify the generation of sk'. The constants $\hat{\beta} = F(\alpha_1\|m^*)$ and $\hat{\gamma} = G(\hat{\beta}) \oplus m^*$ are hard-coded. We add the following line in the beginning of sk': "If $c \in \alpha_1\|\hat{\beta}\|\hat{\gamma}$, output m^*." . For reference, we describe the modified decryption program from Hybrid Hyb_1^2 in Figure 15.

Hyb_1^3**:** We again modify the generation of sk'. We add the following check: "If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$." For reference, we describe the modified decryption for sk' from Hybrid Hyb_1^3 in Figure 16.

Claim: If $\mathcal{F}$ is an injective pPRF family and iO is a secure iO, then $\mathsf{Hyb}_1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_1^1$.

Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured F', and G, and the values α_0, α_1, $\hat{\beta}$, $\hat{\gamma}$, m^*.
Inputs: $c = (\alpha \| \beta \| \gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.

1. If $\alpha = \alpha_1$ and $\beta = \hat{\beta}$ and $\gamma = \hat{\gamma}$, output m^*.
2. If $c \in \{c_0, c_1\}$, output $\perp$.
3. Compute $m = G(\beta) \oplus \gamma$.
4. If $\beta = F'(\alpha \| m)$, output m.
5. Else output $\perp$.

Figure 15: Modified Program of D' in Hyb_1^2 (pre-obfuscation)

Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured F', and G, and the values α_0, α_1, $\hat{\beta}$, $\hat{\gamma}$, m^*.
Inputs: $c = (\alpha \| \beta \| \gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.

1. If $\alpha = \alpha_1$ and $\beta = \hat{\beta}$ and $\gamma = \hat{\gamma}$, output m^*.
2. If $c \in \{c_0, c_1\}$, output $\perp$.
3. Compute $m = G(\beta) \oplus \gamma$.
4. If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
5. If $\beta = F'(\alpha \| m)$, output m.
6. Else output $\perp$.

Figure 16: Modified Program of D' in Hyb_1^3 (pre-obfuscation)

Proof. A modified program that uses F' is functionally equivalent to E because F' is never evaluated on strings of these forms due to the uniform randomness of α_0, α_1. Values α_0 and α_1 are with high probability not in the image of PRG. Thus, the claim holds due to the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If iO is a secure iO, then $\mathsf{Hyb}_1^1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_1^2$.

Proof. The decryption programs in these hybrids are functionally equivalent, as $\alpha_1 \| \hat{\beta} \| \hat{\gamma}$ is already a valid encryption of m^*. Notice, that these $\hat{\beta}$ do not correspond to either the β_0 or β_1 (and similarly for $\hat{\gamma}$). The claim holds due to the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If iO is a secure iO, then $\mathsf{Hyb}_1^2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_1^3$.

Proof. The decryption programs in these hybrids are functionally equivalent by two cases:

1. When $(\alpha, m) = (\alpha_0, m^*)$, then either $c = c_0$, in which case sk' already would output $\perp$, or $c \neq c_0$, in which case sk' rejects c as an invalid ciphertext (because every pair (α, m) together define a unique valid ciphertext due to the injective property of F).
2. When $(\alpha, m) = (\alpha_1, m^*)$, we only reach this line if $c \neq \alpha_1 \| \hat{\beta} \| \hat{\gamma}$ (by the check introduced in Hybrid Hyb_1^2). In this case, sk' already rejects c as an invalid ciphertext.

Thus, the claim holds due to the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If iO is a secure iO, then $\mathsf{Hyb}_1^3 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2$.

Proof. In Hyb_2, instead of using the un-punctured key for F in sk', we puncture F at the points $\alpha_0 \| m^*$ and $\alpha_1 \| m^*$. For sk', the modified program is functionally equivalent to that in the previous hybrid because – by the checks added in the previous hybrid – F will never be evaluated on such inputs. □

Thus, the lemma holds. □

Lemma A.6 *If $\mathcal{F}$ is an injective pPRF family,* Com *is secure injective commitment, and iO is a secure iO, then* $\mathsf{Hyb}_2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_3$

Proof of Lemma A.6. To prove the lemma, we define auxiliary hybrids.

Hyb_2^1**:** We alter the generation of the the key sk' in the security game. Instead of using $\hat{\beta} = F(\alpha_1 \| m^*)$, we sample $\hat{\beta}$ uniformly at random from $\{0,1\}^{9\ell}$.

Hyb_2^2**:** We change Line 1 of Figure 11. Value $\hat{z} := \mathsf{Com}(0;\, \hat{\beta})$ is hard-coded, and we replace the check "$\beta = \hat{\beta}$" with the check "$\mathsf{Com}(0;\, \beta) = \hat{z}$".

Hyb_2^3**:** We change the hard-coded value $\hat{z}$ into "$\mathsf{Com}(1;\, \hat{\beta})$".

Hyb_2^4**:** We replace the expression "$\mathsf{Com}(0;\, \beta) = \hat{z}$" with FALSE.

For reference, we describe sk' from Hybrid Hyb_2^4 in Figure 17.

Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured F', G, and the values α_0, α_1, m^*, $\hat{\gamma}$.
Inputs: $c = (\alpha \| \beta \| \gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.

1. For some i, if $\alpha = \alpha_1$ and FALSE and $\gamma = \hat{\gamma}$, output m^*. (i.e., this never happens)
2. If $c \in C$, output $\perp$.
3. Compute $m = G(\beta) \oplus \gamma$.
4. If $(\alpha, m) \in \{(\alpha_0, m^*),\ (\alpha_1, m^*)\}$, output $\perp$.
5. If $\beta = F'(\alpha \| m)$, output m.
6. Else output $\perp$.

Figure 17: Modified Program of D_2' in Hyb_2^4 (pre-obfuscation)

Claim: If $\mathcal{F}$ is an injective pPRF family, then $\mathsf{Hyb}_2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2^1$

Proof. This holds due to the pseudorandomness of F at punctured points. □

Claim: If Com is a secure injective commitment and $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_2^1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2^2$.

Proof. The modified decryption programs are functionally equivalent by the injective property of Com. Thus, the holds due to the injective property of Com and the security of $i\mathcal{O}$. □

Claim: If Com is a secure injective commitment, then $\mathsf{Hyb}_2^2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2^3$.

Proof. This holds due to the computational hiding property of Com. □

Claim: If Com is a secure injective commitment and $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_2^3 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_2^4$.

Proof. The modified decryption programs are functionally equivalent with high probability because of the perfect binding property of Com (which follows from injectivity). In fact, we remove the entire line 1 as in Hyb_3, which also preserves functionality. Thus, the claim holds due to the functional equivalence explained above and the security of $i\mathcal{O}$, □

Claim: If $\mathcal{F}$ is an injective pPRF family, then $\mathsf{Hyb}_2^4 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_3$.

Proof. This holds due the pseudorandomness of F at the punctured points. □

Thus, the lemma holds. □

Lemma A.7 *If $\mathcal{G}$ is a pPRF family and $i\mathcal{O}$ is a secure iO, then* $\mathsf{Hyb}_3 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_4$.

Proof of Lemma A.7. To prove this lemma, we define auxiliary hybrids.

Hyb_3^1**:** We alter the generation of pk (see Line 2(d) 10). We puncture G in pk at β_0 and β_1.

Hyb_3^2**:** We alter the generation of sk', changing Line 2 of Figure 12. Instead of "If $c \in \{c_0, c_1\}$: output $\perp$", we replace it with "If $\beta \in \{\beta_0, \beta_1\}$: output $\perp$".

Claim: If $\mathcal{G}$ is a pPRF family and $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_3 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_3^1$.

Proof. The encryption programs in these hybrids are functionally equivalent by the sparsity of F since β_0 and β_1 are now chosen at random, with high probability they are not in the image of F. Thus, the claim holds due the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_3^1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_3^2$.

Proof. To see that the modified decryption programs in these hybrids are functionally equivalent, we observe that with high probability, neither of these lines has any effect.

Since with high probability, none of the β_0 and β_1 are in the image of F, if $\beta \in \{\beta_0, \beta_1\}$ – which is the case when $c \in \{c_0, c_1\}$ – then $sk'(c) = \perp$ with high probability, even without the extra check.

We do not remove the check because checking if $\beta \in \{\beta_0, \beta_1\}$ will allow us to puncture G on this set in the following hybrid. This holds due the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If $\mathcal{G}$ is a pPRF family and $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_3^2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_4$.

Proof. In Hyb_4, we alter the generation of sk'. We puncture G at $\{\beta_0, \beta_1\}$ in sk'. This change is functionally equivalent because of the ostensibly useless checks in the previous hybrid. Thus, the claim holds due the functional equivalence explained above and the security of $i\mathcal{O}$. □

Thus, the lemma holds. □

Lemma A.8 *If $\mathcal{G}$ is a pPRF family, then* $\mathsf{Hyb}_4 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_5$

Proof of Lemma A.8. In Hyb_5, we sample γ_0 uniformly at random from $\{0,1\}^\ell$. This change is indistinguishable by the pseudorandomness of G at the punctured set. □

Lemma A.9 *Under the same assumptions as in Theorem A.3,* $\mathsf{Hyb}_5 \overset{\mathsf{c}}{\approx} \mathsf{RAND}$

Proof of Lemma A.9. This is proved in the same way as Lemma A.4, A.5, A.6, A.7, and A.8. □

Therefore, the construction satisfies the ciphertext pseudorandomness. □

B Proof of Theorem 9.13: Waterproof PRFs

The difficulty in this construction is dealing with *arbitrary approximate implementations*. If we try to use the PRF from [BGI+12], changing the functionality on 1 specific point can destroy the learnability. This problem only arises in the case of weak statistical correctness.

We construct a PRF family that has an even stronger form of learnability: from arbitrary approximate implementation C of $f_k \in \mathcal{F}$ that may disagree on $\rho(\lambda) = \mathsf{negl}(\lambda)$ fraction of the domain, we efficiently construct an approximation C' that disagrees with f_k on $\varepsilon(\lambda) = \mathrm{poly}(\lambda)$ fraction of the domain. It seems that we could have done better by simply outputting C! But C' (in particular, the erring inputs) are *completely independent of* C – guaranteeing implementation independence as required to prove that $\mathcal{F}$ is waterproof.

Our starting point is the constructions of unobfuscatable function families in [BGI+12] and [BP13], and an understanding of those constructions will prove helpful towards understanding ours.

The former work was discussed in Proposition 9.5. The latter work handles a very strong form of approximation: the approximate implementation must only agree on some constant fraction of the domain. They achieve this, but sacrifice the total learnability of the earlier construction, instead learning only a single predicate of the PRF key. We require a notion of approximation stronger than [BGI+12] but weaker than [BP13], and a notion of learnability weaker than [BGI+12] but stronger than [BP13], and achieve this by adapting techniques from both works.

B.1 Preliminaries

The construction requires an invoker randomizable pseudorandom function [BGI+12] and a decomposable encryption schemes [BP13]. The following definitions and discussion are taken almost verbatim from those works.

Definition B.1 (Invoker-Randomizable Pseudorandom Functions, [BGI+12]) A function ensemble $\{f_k\}_{k\in\{0,1\}^*}$ such that $f_k : \{0,1\}^{n+m} \to \{0,1\}^m$, where n and m are polynomially related to $|k|$, is called an *invoker-randomizable pseudorandom function ensemble* if the following holds:

1. $\{f_k\}_{k\in\{0,1\}^*}$ is a PRF family.
2. For every k and $x \in \{0,1\}^n$, the mapping $r \mapsto f_k(x,r)$ is a permutation over $\{0,1\}^m$.

Property 2 implies that, for every fixed k and $x \in \{0,1\}^n$, if r is chosen uniformly in $\{0,1\}^m$, then the value $f_k(x,r)$ is distributed uniformly (and independently of x) in $\{0,1\}^m$.

Lemma B.2 ([BGI+12]) *If pseudorandom functions exist, then there exist invoker-randomizable pseudorandom functions.*

Definition B.3 (Decomposable Encryption [BP13]) An encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *decomposable* if there exists an efficient algorithm pub that operates on ciphertexts and satisfies the following conditions:

1. For a ciphertext c, $\mathsf{pub}(c)$ is independent of the plaintext and samplable; that is, there exists an efficient sampler $\mathsf{PubSamp}$ such that, for any secret key $sk \in \{0,1\}^n$:

$$\mathsf{PubSamp}(1^n) \equiv \mathsf{pub}(\mathsf{Enc}_{sk}(0))) \equiv \mathsf{pub}(\mathsf{Enc}_{sk}(1))$$

2. A ciphertext c is deterministeically defined by $\mathsf{pub}(c)$ and the plaintext; that is, for every secret key sk and two distinct ciphertexts c and c', if $\mathsf{pub}(c) = \mathsf{pub}(c')$, then $\mathsf{Dec}_{sk}(c) \neq \mathsf{Dec}_{sk}(c')$.

We use as our decomposable encryption scheme a specific symmetric-key encryption scheme which enjoys a number of other necessary properties. Given a PRF $\{f_k\}_{k\in\{0,1\}^*}$ with one-bit output and for security parameter λ, the secret key is a random $sk \in \{0,1\}^\lambda$, and the encryption of a bit b is computed by sampling a random $r \leftarrow \{0,1\}^\lambda$ and outputting $(r, F_{sk}(r) \oplus b)$. This function satisfies a number of necessary properties [BP13]:

- It is CCA-1 secure.
- It is decomposable.
- The support of $(\mathsf{Enc}_{sk}(0))$ and $(\mathsf{Enc}_{sk}(1))$ are each a non-negligible fraction (in reality, at least $\frac{1}{2} - \mathsf{negl}$) of the cipher-text space.
- For a fixed secret key sk, random samples from $(b, \mathsf{Enc}_{sk}(b))_{b\leftarrow\{0,1\}}$ are indistinguishable from uniformly random strings.

B.2 Construction

The key k for the PRF is given by a tuple $k = (\alpha, \beta, sk, s_1, s_2, s_e, s_h, s_b, s^*)$. For security parameter λ, α and β are uniformly random λ-bit strings, sk is a secret key for the decomposable encryption scheme described above, s_h is a key for an invoker-randomizable pseudorandom function, and s_1, s_2, s_e, s_b, and s^* are independent keys for a family of PRFs. We denote by F_s a PRF with key s.

The domain of the PRF will be of the form (i, q) for $i \in \{1, \ldots, 9\}$, and $q \in \{0,1\}^{\ell(n)}$, for some polynomial ℓ. The range is similarly bit strings of length polynomial in ℓ. The function will be defined in terms of 9 auxiliary functions, and the index i will select among them. We use a combination of ideas from [BGI+12] and [BP13] to construct a PRF family for which s^* can be recovered from any (negligibly-close) approximation to f_k, which will enable us to compute f_k restricted to $i = 9$. This allows us to recover a $1/9$-close approximation of f_k that is implementation independent (simply by returning 0 whenever $i \neq 9$). To achieve a ε-close approximation for any $\varepsilon = 1 - \frac{1}{\mathrm{poly}(\lambda)}$, we simply augment the index i with an additional $\log(1/(1-\varepsilon))$ bits: if all these bits are 0, then we index as before; otherwise, use index $i = 9$. Instead of recovering $1/9$th of the function, we now recover ε of the function. This establishes the theorem.[14]

We now define the auxiliary functionalities we will use in the construction.

- $\mathbb{R}_s$: The function $\mathbb{R}_s$ is parameterized by a PRF key s. It takes as input q and returns $\mathbb{R}_s(q) = F_s(q)$, the PRF evaluated at q. That is, $\mathbb{R}_s$ simply evaluates a PRF.
- $\mathbb{C}_{a,b,s}$: The function $\mathbb{C}_{a,b,s}$ is parameterized by two bit strings a and b, and a PRF key s. It takes as input q and returns $\mathbb{C}_{a,b,s}(q) = b \oplus F_s(q \oplus a)$, where F_s is the PRF given by key s. That is, $\mathbb{C}$ evaluates a PRF on a point related to the queried point, then uses the value to mask the bitstring b.

[14] Note that the result is a PRF family that depends on the choice of ε. The argument would fail if ε was a negligible function, because an approximation for could "erase" all the structure of the PRF family, thwarting learnability. Removing this dependence (ie: constructing a family that works for all inverse polynomial ε simultaneously) would be interesting.

- $\mathbb{E}_{sk,\alpha,s_e}$: The function $\mathbb{E}_{sk,\alpha,s_e}$ is parameterized by a secret key sk for the encryption scheme, a bitstring α, and a PRF key s_E. It takes as input q and returns $\mathbb{E}_{sk,\alpha,s_e}(q) = \mathsf{Enc}_{sk}(\alpha; r)$ with randomness $r = F_{s_e}(q)$. That is, $\mathbb{E}$ returns an encryption of α using randomness derived by evaluating the PRF on the query.

- $\mathbb{H}_{sk,s_h}$: The function $\mathbb{H}_{sk,s_h}$ is parameterized by a secret key sk for the encryption scheme, and a invoker-randomizable PRF key s_h. It takes as input two cipher-texts of bits c and d, the description of a two-bit gate $\odot$, and some additional input $\bar{q}$, and returns $\mathbb{H}_{sk,s_h}(c, d, \odot, \bar{q}) = \mathsf{Enc}_{sk}(\mathsf{Dec}_{sk}(c) \odot \mathsf{Dec}_{sk}(d); r)$ with randomness $r = F_{s_h}(c, d, \odot, \bar{q})$. That is, $\mathbb{H}$ implements a homomorphic evaluation of $\odot$ on the ciphertexts c and d by decrypting and re-encrypting, with randomness derived by applying a PRF to the whole input.

- $\mathbb{B}_{sk,\alpha,\beta,s_b}$: The function $\mathbb{B}_{sk,\alpha,\beta,s_b}$ is parameterized by a secret key sk for the symmetric-key encryption scheme, bitstrings α and β, and a PRF key s_b. It takes as input n ciphertexts $c_1, \ldots, c_\lambda$ and additional input $\bar{q}$, and returns

  $$\mathbb{B}_{sk,\alpha,\beta,s_b}(c_1, \ldots, c_\lambda, \bar{q}) = \alpha \oplus F_{s_b}(m_1 \oplus \beta_1, \ldots, m_\lambda \oplus \beta_\lambda, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_\lambda), \bar{q})$$

  where $m_i = \mathsf{Dec}_{sk}(c_i)$.

Having defined the auxiliary functions, our pseudorandom function f_k for $k = (\alpha, \beta, sk, s_1, s_2, s_e, s_h, s_b, s^*)$ is a combination of these functions. The argument (i, q) selects which function is evaluated, and q is parsed appropriately by each of the functionalities. For example, $\mathbb{B}$ parses q as λ ciphertexts $c_1, \ldots, c_\lambda$, and all remaining bits as $\bar{q}$.

$$f_k(i, q) = \begin{cases} \mathbb{C}_1(q) := \mathbb{C}_{\alpha,\beta,s_1}(q) & \text{if } i = 1 \\ \mathbb{C}_2(q) := \mathbb{C}_{\alpha,s^*,s_2}(q) & \text{if } i = 2 \\ \mathbb{E}(q) := \mathbb{E}_{sk,\alpha,s_e}(q) & \text{if } i = 3 \\ \mathbb{H}(q) := \mathbb{H}_{sk,s_h}(q) & \text{if } i = 4 \\ \mathbb{B}(q) := \mathbb{B}_{sk,\alpha,\beta,s_b}(q) & \text{if } i = 5 \\ \mathbb{R}_1 := \mathbb{R}_{s_1}(q) & \text{if } i = 6 \\ \mathbb{R}_2 := \mathbb{R}_{s_2}(q) & \text{if } i = 7 \\ \mathbb{R}_b := \mathbb{R}_{s_b}(q) & \text{if } i = 8 \\ \mathbb{R}^* := \mathbb{R}_{s^*}(q) & \text{if } i = 9 \end{cases}$$

While this construction may appear daunting, each subfunction serves a very concrete purpose in the argument; understanding the proof ideas will help clarify the construction. We must now argue two properties of this family: learnability as in Theorem 9.13, and pseudorandomness.

B.3 Learnability

We must show that $F_\lambda = \{f_k\}$ is robustly, $\frac{1}{9}$-approximately learnable by an implementation-independent algorithm, L from arbitrary approximate implementation.[15] It suffices to show that, given any ρ-implementation g of f_k for random key k, s^* can be recovered, because $\mathbb{R}^* = \mathbb{R}_{s^*}$ comprises $1/9$th of the functionality.

To begin, consider the case the when the implementation is perfect: $g \equiv f_k$. In this case, recovery of s^* is straightforward. Given α, $\mathbb{C}_1$, and $\mathbb{R}_1$ it is easy to find β: for any q, $\beta = \mathbb{C}_1(q) \oplus \mathbb{R}_1(q \oplus \alpha)$. That is, it is easy to construct a circuit that, on input α, outputs β (by fixing some uniformly random q in the above). [16] But we don't know α, only encryptions of α (coming from $\mathbb{E}$), so how might we recover β?

[15] As discussed earlier, it suffices to prove learnability for $\varepsilon = 1/9$. We may then change the how the subfunctions are indexed to achieve any inverse polynomial.

[16] This ability is what enables the learnability; the black-box learner cannot construct such a circuit and thus cannot continue with the homomorphic evaluation in the next step.

Using $\mathbb{H}$, it is easy to homomorphically evaluate the circuit on such an encryption, yielding an encryption $c = (c_1, \ldots, c_n)$ of $\beta = (\beta_1, \ldots, \beta_n)$. For any $\bar{q}$, evaluating $\mathbb{B}(c, \bar{q})$ will yield $\alpha \oplus F_{s_b}(\mathbf{0}, c, \bar{q})$. Evaluating $\mathbb{R}_b(\mathbf{0}, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_n), \bar{q})$ immediately yields α in the clear. Now we can directly recover $s^* = \mathbb{C}(q) \oplus \mathbb{R}_2(q \oplus \alpha)$, for any q.

How does this argument change when g and f_k may disagree on an (arbitrary) ρ-fraction of the domain for some negligible function $\rho(n)$? The first observation is that in the above algorithm, each of $\mathbb{C}_1$, $\mathbb{C}_2$, $\mathbb{E}$, $\mathbb{R}_1$, and $\mathbb{R}_2$, can each evaluated (homomorphically in the case of $\mathbb{C}_1$) at a single point that is distributed uniformly at random. With high probability, g will agree with f_k on these inputs.

It remains to consider robustness to error in $\mathbb{H}$, $\mathbb{B}$, and $\mathbb{R}_b$. The same idea does not immediately work, because the queries to these circuits are not uniform.

For $\mathbb{H}$, we leverage the invoker-randomizability of the PRF F_{s_h}, using the argument presented in [BGI+12, Proof of Theorem 4.3]. In every query to $\mathbb{H}(c, d, \odot, \bar{q})$, the input $\bar{q}$ only effects the randomness used in the final encrypted output. For each such query, pick $\bar{q}$ uniformly and independently at random. Now $\mathbb{H}$ returns a uniformly random encryption of $\mathsf{Dec}_{sk}(c) \odot \mathsf{Dec}_{sk}(d)$. This is because the randomness used for the encryption is now uniformly sampled by F_{s_h}. The distribution over the output induced by the random choice of $\bar{q}$ depends only on $(\mathsf{Dec}_{sk}(c), \mathsf{Dec}_{sk}(d), \odot) \in \{0,1\}^2 \times \{0,1\}^2 \times \{0,1\}^4$. As in [BGI+12], the probability of returning an incorrect answer on such a query is at most 64ρ, which is still negligible.

For $\mathbb{B}$ and $\mathbb{R}_b$, we leverage the properties of the decomposable symmetric-key encryption scheme, using the argument presented in [BP13, Proof of Claim 3.8]. We modify the procedure of using $\mathbb{B}$ and $\mathbb{R}_b$ to recover α given an encryption c of β. Instead of querying $\mathbb{B}$ on $(c, \bar{q})$, sample a fresh random m, and using $\mathbb{H}$, compute an encryption c' of $\beta \oplus m$. Note that c' is a uniformly random encryption (by invoker-pseudorandomness) of the uniformly random string $\beta \oplus m$, and is thus a uniformly-distributed string of the appropriate length. Independently sample a random $\bar{q}$ and query $\alpha' := \mathbb{B}(c', \bar{q})$. This query to $\mathbb{B}$ is now distributed uniformly, and will therefore be answered correctly with high probability.

To recover α, we evaluate $\alpha = \alpha' \oplus \mathbb{R}_b(m, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_\lambda), \bar{q})$. This query to $\mathbb{R}_b$ is also distributed uniformly at random (for random $\bar{q}$), and will therefore be answered correctly with high probability.

B.4 Pseudorandomness

Our proof that the family $\{f_k\}$ is pseudorandom follows that of [BP13]; the main technical change comes from the fact that $\mathbb{B}$ depends on α. We consider a polynomial-time adversary $\mathcal{A}$ with oracle access to f_k. For simplicity, we ignore the indexing of the subfunctions of f_k and assume that $\mathcal{A}$ has direct oracle access to each of the constituent functions, showing that they are simultaneously pseudorandom.

Let E_1 be the the event that $\mathcal{A}$ produces *distinct* queries $q = (c, \bar{q})$, $q' = (c', \bar{q}')$ such that:

$$(m \oplus \beta, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_\lambda), \bar{q}) = (m' \oplus \beta, \mathsf{pub}(c'_1), \ldots, \mathsf{pub}(c'_\lambda), \bar{q}')$$

where $m, m' \in \{0,1\}^\lambda$ are the decryptions under sk of c and c' respectively.

Claim B.4 $\Pr_{k,\mathcal{A}}[E_1] = 0$

Proof. Recall that for any ciphertext c, $\mathsf{pub}(c)$ and the plaintext m uniquely determine the ciphertext. If $m \oplus \beta = m' \oplus \beta$, and $\mathsf{pub}(c_i) = \mathsf{pub}(c_i)'$ for all i, then $c = c'$. Therefore $q = q'$. □

We consider two "bad" events, and argue that if $\mathcal{A}$ is to distinguish f_k from a random function, (at least) one of the events must occur.

- Let E_α be the event that $\mathcal{A}$ produces queries q and q' such that $q \oplus \alpha = q'$.
- Let E_β be the event that $\mathcal{A}$ produces queries $q = (c, \bar{q})$ and q' such that $q' = (m \oplus \beta, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_\lambda), \bar{q})$, where $m \in \{0,1\}^\lambda$ is the decryption under sk of c.

Claim B.5 *If* $\Pr_{k,\mathcal{A}}[E_\alpha] \leq \mathsf{negl}(\lambda)$ *and* $\Pr_{k,\mathcal{A}}[E_\beta] \leq \mathsf{negl}(n)$*, then* $\mathcal{A}$ *cannot distinguish between* f_k *and a random function.*

Proof. Because f_k depends on the PRF keys s_1, s_2, s_e, s_h, and s_b (but not s^*) only by black-box application of the respective PRFs, we can indistinguishably replace all applications of these PRFs by (independent) truly random functions. If E_α never occurs, than the responses from $\mathbb{C}_1$ and $\mathbb{R}_1$ (respectively $\mathbb{C}_2$ and $\mathbb{R}_2$) are uncorrelated; thus we can indistinguishably replace $\mathbb{C}_1$ (respectively, $\mathbb{C}_2$) by a independent random function. At this point, $\mathcal{A}$'s oracle only depends on s^* through calls to the PRF F^*_s; we can now replace $\mathbb{R}^*$ with a independent random function. By similar reasoning, if E_β never occurs, then the responses from $\mathbb{B}$ and $\mathbb{R}_b$ are uncorrelated; thus we can indistinguishably replace $\mathbb{B}$ with another independent random function. The above holds with high probability, conditioning on $\neg E_\alpha$ and $\neg E_\beta$.

Now $\mathcal{A}$ is left with oracles of $\mathbb{E}$ and $\mathbb{H}$ in which the PRFs F_{s_e} and F_{s_h} have been replaced by random (along with 7 additional independent random functions). The ciphertexts of the encyption scheme we use are pseudorandom. Thus, access to these two oracles may be replaced with random without noticeably affecting the output distribution of $\mathcal{A}$. □

All that remains is to bound the probabilities of E_α and E_β. We consider two cases separately: when E_α occurs before E_β and vice-versa, arguing that the probability of either event occurring first is negligible. Let $E_{\alpha,i}$ (respectively, $E_{\beta,i}$) be the event that E_α (respectively E_β) occurs in the first i queries.

Claim B.6 *For all* i*,* $\Pr_{k,\mathcal{A}}[E_{\beta,i}|\neg E_{\alpha,i-1}] \leq \mathsf{negl}(\lambda)$

Proof. It suffices to show that for all i:

$$\Pr_{k,\mathcal{A}}[E_{\beta,i}|\neg E_{\alpha,i-1}, \neg E_{\beta,i-1}] \leq \mathsf{negl}(\lambda).$$

Furthermore, because the events are efficiently testable given only α, β, and sk, it is enough to prove the claim when all the underlying PRFs (corresponding to s_1, s_2, s_e, s_h, s_b, and s^* are replaced by (independent) truly random functions.

As in Claim B.5, if E_α doesn't occur in the first $i-1$ queries, than the responses from $\mathbb{C}_1$ and $\mathbb{R}_1$ (respectively $\mathbb{C}_2$ and $\mathbb{R}_2$) are uncorrelated on these queries; thus we can indistinguishably replace $\mathbb{C}_1$ (respectively, $\mathbb{C}_2$) by a independent random function. By similar reasoning, if E_β doesn't occur in the first $i-1$ queries, then the responses from $\mathbb{B}$ and $\mathbb{R}_b$ are uncorrelated on these queries; thus we can indistinguishably replace $\mathbb{B}$ with another independent random function. The above holds with high probability, conditioning on $\neg E_{\alpha,i-1}$ and $\neg E_{\beta,i-1}$.

The view of $\mathcal{A}$ after the first $i-1$ queries is now independent of β. Now E_β amounts to outputting a ciphertext c and string q such that $\mathsf{Dec}_{sk}(c) \oplus q = \beta$, for $\beta \leftarrow \{0,1\}^\lambda$ drawn independently of the view of the adversary. This occurs with vanishingly small probability. □

Claim B.7 $\Pr_{k,\mathcal{A}}[E_{\alpha,i}|\neg E_{\beta,i-1}] \leq \mathsf{negl}(\lambda)$

Proof. It suffices to show that for all i:

$$\Pr_{k,\mathcal{A}}[E_{\alpha,i}|\neg E_{\beta,i-1}, \neg E_{\alpha,i-1}] \leq \mathsf{negl}(\lambda).$$

Again, because the events are efficiently testable given only α, β, and sk, it is enough to prove the claim when all the underlying PRFs (corresponding to s_1, s_2, s_e, s_h, s_b, and s^* are replaced by (independent) truly random functions. As in the previous claim, we may indistinguishably replace the first $i-$ responses of $\mathbb{C}_1$, $\mathbb{C}_2$, $\mathbb{B}$, $\mathbb{R}_b$, $\mathbb{R}_1$, and $\mathbb{R}_2$ by independent random functions. The above holds with high probability, conditioning on $\neg E_{\alpha,i-1}$ and $\neg E_{\beta,i-1}$.

The view of the adversary is depends on α only by way of $\mathbb{E}$, the circuit that outputs random encryptions of α. Furthermore, besides the oracles $\mathbb{E}$ and $\mathbb{H}$, all of the oracle responses $\mathcal{A}$ receives are uniformly random (and independent of α). But just as in [BGI+12, Claim 3.6.1] and [BP13, Claim 3.3], with only these two oracles, any CCA-1 encryption scheme is semantically secure. Thus we can indistinguishably replace $\mathbb{E}_{sk,\alpha,s_e}$ with $\mathbb{E}_{sk,\alpha,s_e}$ – returning only encryptions of 0. Finally, the view of $\mathcal{A}$ is information theoretically independent of α; as before, we conclude that $E_{\alpha,i}$ occurs with vanishingly small probability. □

C Key-Injective pPRF from LWE or DDH

A key-injective puncturable PRF can be constructed with a modification of the GGM pPRF by using an ensemble of left- and right-injective PRGs $\mathsf{PRG}^{(1)}, \ldots, \mathsf{PRG}^{(n)}$. When we say that $\mathsf{PRG}^{(i)}$ is left- and right-injective, we mean that if $\mathsf{PRG}^{(i)}$ is writen as $\mathsf{PRG}_0^{(i)} \| \mathsf{PRG}_1^{(i)}$, then both $\mathsf{PRG}_0^{(i)}$ and $\mathsf{PRG}_1^{(i)}$ are injective.

We also require the $\mathsf{PRG}^{(i)}$'s to have additive stretch. That is, there exists a polynomial p such that for each i, $\mathsf{PRG}^{(i)}$ maps $\{0,1\}^{\lambda+(i-1)\cdot p(\lambda)} \to \{0,1\}^{\lambda+i\cdot p(\lambda)}$. This ensures that, in the GGM construction, the size of the PRF output is bounded by $n \cdot \mathrm{poly}(\lambda)$. Such PRGs can be constructed from standard assumptions such as DDH or LWE.

Key-Injective pPRFs from LWE. For example, using the learning with errors (LWE) assumption, we define $\mathsf{PRG}_{\boldsymbol{A}} : \mathbb{Z}_q^n \to \mathbb{Z}_p^m$ as $\mathsf{PRG}_{\boldsymbol{A}}(\boldsymbol{x}) := \left\lfloor \boldsymbol{A}^{\mathsf{T}} \cdot \boldsymbol{x} \right\rceil_p$ 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 $\mathsf{PRG}_b(x) = \left\lfloor \boldsymbol{A}_b^{\mathsf{T}} \boldsymbol{x} \right\rceil$. In this case, each $\mathsf{PRG}_b(x)$ 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.

Key-Injective pPRFs from DDH. Alternatively, it may seem that using DDH, we can set $\mathsf{PRG}_{g_1,g_2}(x) = g_1^x, g_2^x$ where g_1, g_2 are generators of some group $\mathbb{G}$ of prime 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 $\mathsf{PRG}_{g_1,g_2,g_3,h_0,h_1}(x) = h_0(g_1^x, g_2^x, g_3^x), h_1(g_1^x, g_2^x, g_3^x)$ 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".