Publicly Verifiable Software Watermarking

Aloni Cohen[*] Justin Holmgren[†] Vinod Vaikuntanathan[‡]

April 22, 2015

Abstract

Software Watermarking is the process of transforming a program into a functionally equivalent "marked" program in such a way that it is computationally hard to remove the mark without destroying functionality. Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang (CRYPTO 2001) defined software watermarking and showed that the existence of indistinguishability obfuscation implies that software watermarking is impossible. Given the recent candidate constructions of indistinguishability obfuscation, this result paints a bleak picture for the possibility of meaningful watermarking.

We show that slightly relaxing the functionality requirement gives us strong positive results for watermarking. Namely, instead of requiring the marked program to agree with the original unmarked program on *all inputs*, we require only that they agree on a large fraction of inputs. With this relaxation in mind, our contributions are as follows.

1. We define publicly verifiable watermarking where marking a program requires a secret key, but anyone can verify that a program is marked. The handful of existing watermarking schemes are secretly verifiable, and moreover, satisfy only a weak definition where the adversary is restricted in the type of unmarked programs it is allowed to produce (Naccache, Shamir and Stern, PKC 1999; Nishimaki, EUROCRYPT 2013). Moreover, our definition requires security against chosen program attacks, where an adversary has access to an oracle that marks programs of her choice.

2. We construct a publicly verifiable watermarking scheme for any family of puncturable pseudo-random functions (PPRF), assuming indistinguishability obfuscation and injective one-way functions.

We also give an indication of the limits of watermarking by showing that the existence of robust totally unobfuscatable families of functions rules out a general watermarking scheme for cryptographic functionalities such as signatures and MACs.

1 Introduction

Software watermarking is the process of embedding a "mark" in a program so that the marked program preserves functionality, and furthermore, it is impossible to remove the mark without destroying functionality. Despite its numerous applications in digital rights management and copy

[*]E-mail: aloni@mit.edu. MIT.

[†]E-mail: holmgren@mit.edu. MIT.

[‡]E-mail: vinodv@mit.edu. MIT.

protection, rigorous definitions of software watermarking and mathematically sound constructions have been few and far between.

Software watermarking was first formally defined in the seminal work of Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang [BGI+12]. They showed that the existence of indistinguishability obfuscation (IO) implies that software watermarking *cannot* exist (for any non-trivial class of programs). Given the recent candidate constructions of IO [GGH+13, BR14, PST14, GLSW14], this result paints a rather dismal picture for the possibility of meaningful watermarking. Fortunately, though, this impossibility result crucially relies on the fact that the marked version of a program *computes the same function* as the original program. Indeed, they suggested that an *approximate functionality-preserving* relaxation of the definition wherein the programs agree on a large fraction (but not all) inputs might lend itself to positive results.

In this work, we pursue the notion of approximate functionality, and show constructions of watermarking for a general class of functions, namely any family of puncturable pseudo-random functions [BW13, BGI14, KPTZ13], under a strong definition of security. Our construction is *publicly verifiable*, meaning that while marking a program requires a secret (marking) key, verification is public. Moreover, our construction is secure against *chosen program attacks*, where the adversary gets access to an oracle that marks any program of its choice. Curiously enough, our construction relies on the existence of indistinguishability obfuscators. We describe our results in more detail.

1.1 Our Results

Our first contribution is to define the notion of public-key watermarking, building on the beautiful work of Hopper, Molnar and Wagner [HMW07] who introduced a secret-key definition.[1] Roughly speaking, in a watermarking scheme, there is a marking algorithm Mark that takes as input a program P and uses the (secret) marking key MK to produce a marked program $\#P$[2] while the verification algorithm Verify uses the (public) verification key VK to recognize legally marked programs. A watermarking scheme should satisfy three properties:

- *Approximately Functionality Preserving:* The program $\#P$ should agree with P in at least $1 - \rho(n)$ fraction of inputs, where ρ is the approximate functionality parameter.

- *Unremovability:* We say that a watermark remover $\mathcal{R}$ succeeds given $\#P$ if she produces as program $\hat{P}$ that is approximately (functionally) equivalent to $\#P$ and yet, Verify fails to recognize $\hat{P}$ as a marked program. Unremovability says that this should be hard for polynomial-time removers $\mathcal{R}$.

- *Unforgeability:* The other side of the coin is unforgeability which requires that a forger $\mathcal{F}$ cannot succeed in producing a new marked program, given only $\#P$ and the verification key VK.

Moreover, we require security against *chosen program attacks*, namely that unremovability and unforgeability hold even given access to a Mark oracle that produces marks programs for the adversary. (In this case, the forger needs to produce a marked program that is sufficiently different from all his queries to the Mark oracle).

[1]While the work of [HMW07] targeted the watermarking of perceptual objects such as text, audio and video, the definition remains essentially unchanged for watermarking programs or circuits.

[2]Marked programs are always denoted as $\#P$ or $\#C$ (in the case of circuits) in this paper.

Armed with this definition, our main result is the construction of a publicly verifiable watermarking scheme for any family of puncturable pseudo-random functions. Puncturable pseudo-random functions (PPRFs) [BW13, BGI14, KPTZ13] are pseudorandom functions wherein the owner of the key K can produce a punctured key K_x that allows computation of the PRF on all inputs $y \neq x$. Moreover, given the punctured key, $\mathsf{PRF}_K(x)$ is pseudorandom. PPRFs have seen a large number of uses recently in applications of indistinguishability obfuscation. We show:

Theorem 1.1 (Informal)**.** *Assuming indistinguishability obfuscation and injective one-way functions, there is a watermarking scheme for any family of puncturable pseudo-random functions.*

A natural question that arises from this result is: Why stop with PPRFs? Can we watermark expressive cryptographic functionalities such as (general) pseudo-random functions, digital signatures, and encryption algorithms? We show limitations on general purpose watermarking schemes for natural cryptographic functionalities. Roughly speaking, we show that given a family of robust totally unobfuscatable functions (a generalization of robust unobfuscatable functions of [BP12] and totally unobfuscatable functions of [BGI$^+$12]) there are signature schemes that cannot be watermarked. We refer the reader to Section 7 for more details.

1.2 Our Techniques

We describe our publicly verifiable watermarking construction through a sequence of ideas. As a starting point, we construct a simple secret-key watermarking scheme for puncturable PRFs. Intuitively, the (secret) marking key specifies a random pair (x^*, y^*), and the watermarking of a PRF key K is simply the indistinguishability obfuscation of a program P_{K,x^*,y^*} that does the following:

On input x, output y^* if $x = x^*$, and $\mathsf{PRF}_K(x)$ otherwise.

(Secret-key) verification of a circuit C is simply checking whether $C(x^*) \overset{?}{=} y^*$. Intuitively, it should be hard to remove the mark since the obfuscation $\#P = \mathcal{O}(P_{K,x^*,y^*})$ "should hide" the location x^* where the functionality is altered. In other words, removing the mark necessitates altering the program at essentially all points. We formalize this intuition via a two-step proof.

First, we show a general *distinguishing-to-removing* reduction. Consider a watermark remover that takes the program $\#P$ and outputs an unmarked, yet approximately equivalent, program Q. We claim that the remover can be used to distinguish between the special input x^* and a uniformly random input x. This is because (a) we know that $\#P(x^*) \neq Q(x^*)$ (simply because the watermark remover succeeded), and yet: (b) for a uniformly random input x, $\#P(x) = Q(x)$ w.h.p. (because Q and $\#P$ agree on a large fraction of inputs). This idea can be formalized as a general distinguishing-to-removing reduction, as long as the watermark verifier uses the program $\#P$ as a black-box (which is true for all our constructions).

Secondly, we show that $\#P$ hides x^*, crucially relying on both the pseudorandomness of the family as well as its puncturability. First, one can indistinguishably replace $\#P$ by a program that uses the punctured PRF key K_{x^*} to compute the PRF on all inputs $x \neq x^*$; this is because of the IO security guarantee. Second, we can replace y^* by $\mathsf{PRF}_K(x^*)$ indistinguishably; this is because of pseudo-randomness. Finally, change the program to one that simply computes the PRF on all points, using the IO security guarantee again. At this point, the program contains no information about x^* whatsoever.

Unforgeability can be shown using similar ideas.

However, this construction falls to the following attack, against an adversary that obtains the marked version of even a single program of its choice. Consider an adversary that obtains a marked version $\#Q$ of an unmarked program Q that she chooses. She can build a distinguisher for x^* using these two programs. Indeed, if $Q(x) \neq \#Q(x)$, then x is likely x^*. In other words, she can build a program $\mathsf{Dist}_{Q,\#Q}$ that, on input x, predicts whether $x = x^*$. With this newfound ability, the adversary can easily remove marks from programs. Indeed, given a marked program $\#P$, it builds a wrapper around P that first checks if an input x is x^*. If yes, it outputs $\perp$, otherwise, it computes the program correctly.

Looking back at this attack (which we call the "majority attack"), we realize that the main source of difficulty is that we reuse the trigger point x^* across all programs. Our main idea to circumvent this attack is to make the trigger point *program-dependent*. In particular, the marking algorithm probes the program on a set of pre-determined inputs to obtain x^* (That is, x^* is a function of the *values* of the program on these inputs). Then, it goes ahead and changes the output of the program on x^*. This allows us to construct a watermarking scheme secure against *lunch-time chosen program attacks*. That is, the adversary can query the Mark oracle before it obtains the challenge program, but not after.

Finally, we augment the construction to be publicly verifiable. Our security proof sketched above relied crucially on the fact that the trigger points cannot be distinguished from uniformly random points, and yet verification seems to require knowledge of these points. We resolve this apparently conundrum by first embedding an exponentially large number (yet an exponentially small fraction) of trigger points, and observing that while verification requires a program that generates a random trigger point, security only requires that a uniformly random trigger point is pseudo-random. These requirements can indeed co-exist, and in fact, we use a variant of the *hidden sparse trigger* machinery of Sahai and Waters [SW14] to achieve both effect simultaneously. We refer the reader to Section 6 for the technical details.

1.3 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 the security guarantees required of watermarking, and have resulted in a cat-and-mouse game of designing watermarking schemes that are broken fairly immediately.

There are a handful of works that propose rigorous notions of security for watermarking, with associated proofs of security based on complexity-theoretic assumptions. Hopper, Molnar and Wagner [HMW07] formalized strong notions of watermarking security with approximate functionality; our definitions are inspired by their work. Barak et al. [BGI$^+$12] proposed simulation-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 preserves functionality exactly. The starting point of our construction is their speculative idea that relaxing this to approximate functionality might result in positive results.

In another line of work, Naccache, Shamir and Stern [NSS99] showed how to watermark a specific hash function. Nishimaki [Nis14] recently showed a similar result along these lines, watermarking a specific construction of lossy trapdoor functions (LTDF) based on bilinear groups. These works achieve a rather weak notion of watermarking. First and most important, they define a successful adversary as one that outputs a program from the class $\mathcal{C}$. In particular, the watermark remover for the LTDF must output an LTDF key. In contrast, we permit the watermark remover to output any (polynomial-size) circuit. Secondly, that is they are only secure as long as a bounded number of marked programs is released to the adversary. Finally, they are both *privately verifiable.*

2 Preliminaries and Definitions

Notation. We will let λ denote a security parameter throughout this paper. We will let $\{\mathbb{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be the family of all circuits with domain $\{0,1\}^{n(\lambda)}$ and range $\{0,1\}$ for some polynomial n. We will let $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be a particular family of circuits; that is $\mathcal{C}_\lambda \subseteq \mathbb{C}_\lambda$ for all $\lambda \in \mathbb{N}$.

Let $\rho : \mathbb{N} \to [0,1]$ be a function. For circuits $C, C' \in \mathbb{C}_\lambda$, we say that $C \sim_\rho C'$ if

$$\Pr_{x\leftarrow\{0,1\}^n}[C(x) \neq C'(x)] \leq \rho(n)$$

We call such circuits "ρ-close". We refer to probabilistic polynomial time algorithms simply as "p.p.t. algorithms".

2.1 Watermarking Schemes

Our definitions will generalize and refine those of Hopper, Molnar and Wagner [HMW07]. A (public-key) watermarking scheme $\mathcal{W}$ for a family of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ is a tuple of p.p.t. algorithms $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Extract})$, where:

- $(\mathsf{mk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$ is a p.p.t. key generation algorithm takes as input a security parameter $\lambda \in \mathbb{N}$ and outputs a *marking key* mk and a *verification key* vk.
- $C_\# \leftarrow \mathsf{Mark}(\mathsf{mk}, C)$ is a p.p.t. marking algorithm that takes as input the marking key mk and a circuit $C \in \mathbb{C}_\lambda$ and outputs a circuit $C_\#$.
- $b \leftarrow \mathsf{Verify}(\mathsf{vk}, C_\#)$ is a p.p.t. algorithm which takes as input the (public) verification key vk and a (possibly marked) circuit $C_\# \in \mathbb{C}$, and outputs either `accept` (1) or `reject` (0).

Although Mark and Verify can take any circuit as input, we only require correctness and security properties when the input circuit is drawn from the family $\mathcal{C}$.

Correctness Properties. Having defined the syntax of a watermarking scheme, we now define the desired correctness properties. First, it is functionality preserving, namely marking a circuit C does not change its functionality too much. We formalize this by requiring that the marked and unmarked circuits agree on some fraction $\rho(n)$ of the domain (where n is the length of the input to the circuit). Secondly, we require that the verification algorithm always accepts a marked circuit.

Definition 2.1 (ρ-Functionality Preserving)**.** We say that a watermarking scheme $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify})$ is ρ-functionality preserving if for all $C \in \mathcal{C}$:

$$\mathsf{Mark}(\mathsf{mk}, C) \sim_\rho C$$

Definition 2.2 (Completeness)**.** A watermarking scheme $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify})$ is said to be *complete* if

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, \mathsf{Mark}(\mathsf{mk}, C)) = 1 \middle| \begin{array}{l} \mathsf{mk}, \mathsf{vk} \leftarrow \mathsf{Setup}(1^\lambda) \\ C \leftarrow \mathcal{C} \end{array}\right] \geq 1 - \operatorname{negl}(\lambda)$$

Security Properties. We turn to the desired security properties of the watermarking scheme. We define a scheme's "uremovability" and "unforgeability" with respect to the following security game.

The watermarking security game is defined with two helper sets $\mathfrak{C}$ and $\mathfrak{M}$: $\mathfrak{C}$ is the set of marked challenge programs given to a p.p.t. adversary $\mathcal{A}$; $\mathfrak{M}$ is the set of circuits given to the adversary as a response to a $\mathsf{Mark}(\mathsf{mk}, \cdot)$ query. We only require that the adversary cannot "unmark" a challenge program, and that the adversary cannot "forge" a mark on a program for which he has never seen a mark.

Game 1 (Watermarking Security)*.* First, the challenger generates $(\mathsf{mk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and helper sets $\mathfrak{C}$ and $\mathfrak{M}$ initialized to $\varnothing$. The adversary is presented with vk and access to the following two oracles.

- A marking oracle $\mathcal{O}_M$ which takes a circuit C and returns $\mathsf{Mark}(\mathsf{mk}, C)$. $\mathcal{O}_M$ also adds C to the set $\mathfrak{M}$.
- A challenge oracle $\mathcal{O}_C$ which takes no input, but samples a circuit C^* uniformly from $\mathcal{C}$ and returns $\#C^* = \mathsf{Mark}(\mathsf{mk}, C^*)$. $\#C^*$ is then added to $\mathfrak{C}$.

Finally, $\mathcal{A}$ outputs a circuit $\hat{C}$.

Our ideal notion of unremovability is that no adversary can – with better than negligible probability – output a program that is δ-close to the challenge program, but on which Verify returns zero with any noticeable probability. Along the way, we will need to consider a relaxed notion: (p, δ)-unremovable. For this, we require that no adversary can – with better than negligible probability – output a program that is δ-close to the challenge program, but on which Verify returns 0 with *probability significantly greater than p.* If $p = 0$, then this coincides with the previous notion of unremovability, which we simply call δ-unremovability.

Definition 2.3 ((p, δ)-Unremovable)**.** In the security game, we say that the adversary (p, δ)-*removes* if $\Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 0] > p + \frac{1}{\operatorname{poly}(\lambda)}$ and $\hat{C} \sim_\delta C'$ for some $C' \in \mathfrak{C}$. $\mathcal{A}$'s (p, δ)*-removing advantage* is the probability that $\mathcal{A}$ (p, δ)-removes. The scheme is (p, δ)*-unremovable* if all p.p.t. $\mathcal{A}$ have negligible (p, δ)-removing advantage. The scheme is δ*-unremovable* if it is (p, δ)-unremovable for all $p > \operatorname{negl}(\lambda)$.

Formally, the scheme is (p, δ)-unremovable if for all p.p.t. algorithms $\mathcal{A}$ and all polynomials $\operatorname{poly}(\lambda)$:

$$\Pr\left[\begin{array}{rl} & \left(\Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 0] > p + \frac{1}{\operatorname{poly}(\lambda)}\right) \\ \wedge & \left(\exists C' \in \mathfrak{C} : C' \sim_\delta \hat{C}\right) \end{array} \middle| \begin{array}{l} \mathsf{mk}, \mathsf{vk} \leftarrow \mathsf{Setup}(1^\lambda), \\ \hat{C} \leftarrow \mathcal{A}^{\mathcal{O}_M, \mathcal{O}_C}(1^\lambda, \mathsf{vk}) \end{array}\right] \leq \operatorname{negl}(\lambda)$$

Likewise, we say a scheme is (q, γ)-unforgeable if no adversary can – with better than negligible probability – output a program that is γ-far from all marked programs previously received from the challenger, but on which Verify returns 1 with *probability significantly greater than* q. If $q = 0$, then this coincides with a stronger notion we call γ-unforgeability.[3]

Definition 2.4 ((q, γ)-Unforgeable)**.** In the security game, we say that the adversary (q, γ)*-forges* if $\Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1] > q + \frac{1}{\mathrm{poly}(\lambda)}$ and for all $C' \in \mathfrak{M} \cup \mathfrak{C}$, $\hat{C} \not\sim_\gamma C'$. $\mathcal{A}$'s (q, γ)*-forging advantage* is the probability that $\mathcal{A}$ (q, γ)-forges. The scheme is (q, γ)*-unforgeable* if all p.p.t. $\mathcal{A}$ have negligible (q, γ)-forging advantage. The scheme is γ*-unforgeable* if it is (q, δ)-unforgeable for all $q > \mathrm{negl}(\lambda)$.

Formally, the scheme is (q, γ)-unforgeable if for all p.p.t algorithms $\mathcal{A}$ for all polynomials $\mathrm{poly}(\lambda)$:

$$\Pr\left[\begin{array}{c} \left(\Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1] > q + \frac{1}{\mathrm{poly}(\lambda)}\right) \\ \wedge \quad \left(\forall C' \in \mathfrak{M} \cup \mathfrak{C} : C' \not\sim_\gamma \hat{C}\right) \end{array} \middle| \begin{array}{l} \mathsf{mk}, \mathsf{vk} \leftarrow \mathsf{Setup}(1^\lambda), \\ \hat{C} \leftarrow \mathcal{A}^{\mathcal{O}_M, \mathcal{O}_C}(1^\lambda, \mathsf{vk}) \end{array}\right] \leq \mathrm{negl}(\lambda)$$

2.1.1 Weakened Definitions

In our main construction, we achieve a weaker version of δ-unremovability, which we call δ*-lunchtime unremovability.*[4] In this weaker definition, the adversary only has access to the marking oracle before receiving the challenge, and receives only one challenge program.

Definition 2.5 ((p, δ)-Lunchtime Unremovability)**.** δ*-lunchtime unremovability* is defined as above, except that we modify the security game: the adversary can query $\mathcal{O}_C$ at most once, after which the adversary can no longer query $\mathcal{O}_M$.

In our main construction, we also achieve a weaker notion of γ-unforgeability. In particular, we only show that a "strong" type of γ-forgery is impossible, in effect establishing only a relaxed form of unforgeability.

In order to say that $\mathcal{A}$ γ*-strong-forges*, we require an additional property on $\mathcal{A}$'s output $\hat{C}$. Instead of requiring that for all marked programs $C' \in \mathfrak{M} \cup \mathfrak{C}$ received by the adversary, there is a γ fraction of the domain on which C' differs from $\hat{C}$, we switch the order of the quantifiers. That is, there is a γ fraction of the domain on which for all $C' \in \mathfrak{M} \cup \mathfrak{C}$, C' differs from $\hat{C}$.

Definition 2.6 ((q, γ)-Relaxed Unforgeability)**.** We say that the adversary (q, γ)*-strong forges* if $\Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1] > q + \frac{1}{\mathrm{poly}(\lambda)}$ and for all $C' \in \mathfrak{M} \cup \mathfrak{C}$,

$$\Pr[\exists C' \in \mathfrak{M} \cup \mathfrak{C} \text{ s.t. } C'(x) = \hat{C}(x) | x \leftarrow \{0,1\}^n] \leq \gamma$$

The scheme is (q, γ)*-relaxed unforgeable* if all p.p.t. $\mathcal{A}$ have negligible (q, γ)-strong forging advantage.

2.2 Remarks on the Definition

Relation to definition in "From Weak to Strong Watermarking" As discussed in the introduction, [HMW07] provide a similar definition for watermarking to that provided above. One major difference is the notion of public verification – to the best of our knowledge, we are the first to put forth this notion and provide a construction in any cryptographic setting.

[3]This implies a sort of "meaningfulness" property a la [BGI+12]: that for a random circuit $C \leftarrow \mathcal{C}$, $\mathsf{Verify}(\mathsf{vk}, C) = 0$ with high probability.

[4]This name is in analogy to "lunchtime" chosen ciphertext attacks on encryption schemes, in which access to a decryption oracle is given only before the challenge ciphertexts.

Another important difference is in the definitions of ρ-functionality preserving, δ-unremovability, and γ-unforgeability. The earlier definition does not consider these parameters separately: all three are conflated and represented by δ. As observed in that work, the (im)possibility of watermarking in a particular setting depend intimately on these parameters. In this work, we separate them. In doing so, we are able to achieve constant δ for negligible ρ: that is an adversary cannot remove a mark that only alters the behavior of the original negligible fraction of the domain, even by changing the functionality of the program on $1/4$ of the domain! Additionally, we show that if a watermarking scheme is both $(0, \delta)$-unremovable and $(0, \gamma)$-unforgeable, then $\gamma \geq \delta + \frac{1}{\mathrm{poly}(n)}$ for some polynomial poly. (see subsubsection 2.2.1). We also provide an Amplification Lemma (see section 3) which depends crucially on δ and γ being distinct.

Extractable Watermarking We consider a setting in which a program is either "marked" or it is "unmarked." A natural extension is to allow a program to be "marked with a string." That is, Mark would take a string M as an additional argument and output a marked program C_M. A corresponding detection algorithm $\mathsf{Extract}$ would then extract the embedded mark. Indeed, similar definitions are considered in [BGI+12] and [HMW07]. The two notions coincide when the space of possible marks to embed is just a singleton set $\{\top\}$. The extractable setting presents a number of additional challenges, and is an interesting direction for future work.

Perfectly Functionality Preserving. As observed in Barak et al. [BGI+12], if indistinguishability obfuscation exists for $\mathbb{C}$, then no watermarking scheme can exist that perfectly preserves functionality. To see this, choose $C \leftarrow \mathbb{C}_\lambda$ and mark it to get $C_\#$. Obfuscate both circuits (appropriately padded) – yielding $\tilde{C}$ and $\tilde{C}_\#$. By unforgeability, $\mathsf{Verify}(\mathsf{vk}, \tilde{C}) = \perp$; by unremovability $\mathsf{Verify}(\mathsf{vk}, \tilde{C}_\#) = m$. In this way, Verify gives a distinguisher for the two obfuscated circuits. Because Mark perfectly preserves functionality, this violates the security of the obfuscation.

Even more basically, if we restrict our attention to black-box verification (as discussed in section 4), then a watermarking scheme must clearly change the functionality in some way.

2.2.1 Relationship between γ and δ

Below we illustrate some simple requirements on δ and γ in the unremovability and unforgeability definitions that are necessary if both are to be satisfied simultaneously.

Theorem 2.1. *If a watermarking scheme is both $(0, \delta)$-unremovable and $(0, \gamma)$-unforgeable, then $\gamma \geq \delta$. If the watermarking scheme additionally has black-box verification and if one-way functions exist, then there is a polynomial* poly *such that $\gamma \geq \delta + \frac{1}{\mathrm{poly}(n)}$.*

Proof. Recall that an adversary δ-removes for a challenge $C^*_\#$ if it outputs a program $\hat{C} \approx_\delta C^*_\#$ such that $\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 0$ with non-negligible probility. That is, it must remove the mark without changing the challenge program on more than δ-fraction of inputs. An adversary wins the γ-forging game if it outputs a program $\hat{C}$ such that $\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1$ with non-negligible probability and additionally, for all marked programs $C_{i\#}$ seen by the adversary, $\hat{C} \not\approx_\gamma C_{i\#}$. That is, its output is only considered a forgery if it is at least γ-far from all marked programs.

Consider the following "attack": Given a random marked program $C_\# : \{0,1\}^n \to \{0,1\}^m$, consider the following program, parameterized by $c \in [2^n]$:

$$C_c(x) = \begin{cases} C_\#(x) \oplus 1 & \text{if } x \leq c \\ C_\#(x) & \text{if } x > c \end{cases}$$

Consider $b \leftarrow \mathsf{Verify}(\mathsf{vk}, C_c)$. At least one of $\Pr[b = 0]$ or $\Pr[b = 1]$ is at least $1/2$. In the former case, this construction violates unremovability unless $\delta \leq \frac{c}{2^n}$; in the latter case, this construction violates unforgeability unless $\gamma \geq \frac{c}{2^n}$. That is, if a watermarking scheme is both unremovable and unforgeable for parameters δ and γ, then for all $c \in [2^n]$, either $\gamma \geq \frac{c}{2^n}$ or $\delta \leq \frac{c}{2^n}$. Therefore, $\gamma \geq \delta$.

Furthermore, in the setting when Verify is black-box with respect to C_c, then $\gamma \geq \delta + \frac{1}{\mathrm{poly}(n)}$, for some polynomial. We alter the above attack as follows. For $c \in [2^n]$ and a pseudo-random permutation $\pi \leftarrow \mathcal{PRP}$:

$$P_{c,\pi}(x) = \begin{cases} C_\#(x) \oplus 1 & \text{if } \pi(x) \leq c \\ C_\#(x) & \text{if } \pi(x) > c \end{cases}$$

Suppose there is some negligible function $\mathrm{negl}(n)$ such that $\gamma = \delta + \mathrm{negl}(n)$. For $c = \delta 2^n$ and randomly chosen $\pi \leftarrow \mathcal{PRP}$, $\mathsf{Verify}(\mathsf{vk}, C_{\delta 2^n,\pi}) = 1$ with all but negligible probability; otherwise $C_{\delta 2^n,\pi}$ violates unremovability. For $c = \gamma 2^n$, $\mathsf{Verify}(\mathsf{vk}, C_{\gamma 2^n,\pi}) = 0$ with all but negligible probability; otherwise $C_{\gamma 2^n,\pi}$ violates unforgeability.

The programs $C_{\delta 2^n,\pi}$ and $C_{\gamma 2^n,\pi}$ disagree on a negligible fraction of the domain. The set on which they disagree is pseudo-random, by the security of π. Because Verify is black-box, we can use it to distinguish black-box access to these two functions, which is impossible. $\square$

3 Amplifying Unremovability and Unforgeability

Ideally, we would like to construct a watermarking scheme that that is δ-unremovable and γ-unforgeable; that is, for all PPT algorithms $\mathcal{A}$, the probability that $\mathcal{A}$ can remove or forge is negligible. In this section, we show that it suffices to prove something weaker. Informally, we show that given a watermarking scheme $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify})$ that is (p, δ)-unremovable and (q, γ)-unforgeable for some parameters $(1 - p) \geq q + \frac{1}{\mathrm{poly}(\lambda)}$, we construct a watermarking $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify}')$ that is δ-unremovable and γ-unforgeable.

To achieve this, we amplify both security guarantees by repeating Verify a polynomial number of times, and choosing an appropriate verification threshold $\tau = \frac{q+(1-p)}{2}$: if $\mathsf{Verify} = 1$ on more than τ trials, we return 1; if $\mathsf{Verify} = 1$ on fewer than τ trials, we return 0. More formally, we prove the following lemma.

Lemma 3.1 (Amplification Lemma). *Let* $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify})$ *be a* (p, δ)*-unremovable and* (q, γ)*-unforgeable watermarking scheme, where the run-time of* Verify *is* t_{Verify}*. Suppose that* $(1-p) = q+\alpha$ *for some non-negligible* α*. Then there is a watermarking scheme* $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify}_{p,q})$ *that is* $(0, \delta)$*-unremovable and* $(0, \gamma)$*-unforgeable, and the run-time of* $\mathsf{Verify}_{p,q}$ *is* $O(\frac{\lambda}{\alpha^2}) \cdot t_{\mathsf{Verify}}$*.*

Remark 1. The proof is essentially an application of a Hoeffding bound, and is given in detail in Appendix B. The argument also holds for the weaker definitions of unremovability and unforgeability that we consider in this work. Specifically, in the setting of Theorem 6.1, we use the Amplification Lemma to construct a δ-lunchtime unremovable, γ-relaxed unforgeable watermarking scheme from a $(2\delta, \delta)$-lunchtime unremovable, $(1 - \gamma, \gamma)$-relaxed unforgeable scheme.

4 A Distinguishing to Removing Reduction

The way in which we prove unremovability is by showing that no adversary can distinguish points queried by Verify from random. This technique is clearly restricted to schemes in which there is a notion of "points queried by Verify". We will therefore restrict our attention in this work to watermarking schemes with black-box verification.

Definition 4.1 (Watermarking scheme with black-box Verify)**.** We say a watermarking scheme has a **black-box verification** if $\mathsf{Verify}(\mathsf{vk}, P)$ can be efficiently evaluated with oracle access to P.

The distinguishing-to-removing reduction we now illustrate applies to any game in which the adversary's goal is to unmark a random marked program, but for concreteness we only show a reduction to Game 1.

We now give sufficient conditions for a watermarking scheme to be (p, δ)-unremovable. In particular, the condition is that no p.p.t. algorithm $\mathcal{A}$ has non-negligible advantage in the following game.

Game 2 (Distinguishing). First, the challenger samples $(\mathsf{mk}, \mathsf{vk})$ from $\mathsf{Setup}(1^\lambda)$. The adversary is then given vk and access to the following two oracles.

- A marking oracle $\mathcal{O}_M$ which takes a circuit C and returns $\mathsf{Mark}(\mathsf{mk}, C)$.
- A challenge oracle $\mathcal{O}_C$ which when queried, samples a circuit C^* uniformly from $\mathcal{C}$ and computes $C^*_\# = \mathsf{Mark}(\mathsf{mk}, C^*)$. The adversary must query $\mathcal{O}_C$ exactly once.

At the end of the game, the challenger executes $\mathsf{Verify}(\mathsf{vk}, C^*_\#)$. From the points at which Verify queried $C^*_\#$, the challenger picks x_0 uniformly at random. The challenger also chooses x_1 as a random point in the domain of $C^*_\#$. The challenger picks a random bit b and sends x_b to the adversary.

The adversary then outputs a bit b' and wins if $b' = b$.

Lemma 4.1 (Distinguishing to Removing Reduction)**.** *If all p.p.t. algorithms $\mathcal{A}$ have negligible advantage in Game 2, and if* Verify *queries at most L points, then* $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify})$ *is $(L \cdot \delta, \delta)$-unremovable for all δ.*

We prove the lemma by assuming the existence of an $(L \cdot \delta, \delta)$-remover, and constructing a distinguisher for the above game. Consider a removing adversary restricted to changing at most δ fraction of the marked challenge program. If he can remove the mark with too high of a probability (significantly greater than $L \cdot \delta$), then we must show that he can distinguish points queried by Verify from random points with non-negligible advantage. This follows because Verify is black-box; intuitively, the unmarked program $\hat{C}$ will disagree with the challenge $C^*_\#$ at x_0 with significantly greater probability than on a uniformly random point x_1.

Remark 2. It is natural to wonder whether a converse of the above lemma holds. That is, if there exists an algorithm $\mathcal{A}$ with non-negligible advangatge in Game 2, does there exist an efficent (p, δ)-remover for some δ and p? A weak converse may be shown: if $\mathcal{A}$ distinguishes x_0 from x_1 with probability $1 - \text{negl}(\lambda)$, then a $(1 - \text{negl}(\lambda), \delta)$-remover can be constructed for some δ that depends on the specifics of the watermarking scheme.

Proof. Suppose that a removing adversary outputs a circuit $\hat{C}$ such that $\Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 0] = L \cdot \delta + \mathsf{Adv}$ for some non-negligible Adv. Let q_i be a random variable denoting the i^{th} point at which Verify queries $C^*_\#$.

$$\begin{aligned}
L \cdot \delta + \mathsf{Adv} &= \Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 0] \\
&\leq \Pr[\exists i : \hat{C}(q_i) \neq C^*_\#(q_i)] + \mathrm{negl}(\lambda) \qquad (1) \\
&\leq \sum_i \Pr[\hat{C}(q_i) \neq C^*_\#(q_i)] + \mathrm{negl}(\lambda) \qquad (2) \\
&= L \cdot \Pr_{i \leftarrow \{1,\ldots,L\}} \left[\hat{C}(q_i) \neq C^*_\#(q_i)\right] + \mathrm{negl}(\lambda) \\
&= L \cdot \Pr_{x_0} \left[\hat{C}(x_0) \neq C^*_\#(x_0)\right] + \mathrm{negl}(\lambda)
\end{aligned}$$

Inequality (1) is by the black-box property of Verify. With high probability, $\mathsf{Verify}(\mathsf{vk}, C^*_\#) = 1$, by completeness. If $\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 0$, then (with high probability) there must be some queried point q_i for which $\hat{C}(q_i) \neq C^*_\#(q_i)$. Inequality (2) is by a union bound. As a result $\Pr_{x_0}\left[\hat{C}(x_0) \neq C^*_\#(x_0)\right] \geq \delta + \mathsf{Adv}'$ for some non-negligible $\mathsf{Adv}' \approx \frac{\mathsf{Adv}}{L}$.

If the adversary outputs a circuit $\hat{C}$ such that $\hat{C} \sim_\delta C^*_\#$, then since x_1 is chosen uniformly at random, $\Pr_{x_1}\left[\hat{C}(x_1) \neq C^*_\#(x_1)\right] \leq \delta$. Thus there is a p.p.t. algorithm distinguishing x_0 from x_1 with non-negligible advantage (specifically, $\mathsf{Adv}'/2$). □

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 two 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 $\{c_0, c_1\}$ is functionally equivalent to the nonpunctured decryption key, except possibly at c_0 and c_1.

In this section we define the puncturable encryption abstraction that we use in section 6. We instantiate this definition in section 7.

Syntactically, a puncturable encryption scheme $\mathcal{PE}$ 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 $\lambda = \ell$.

- $\mathsf{Gen}(1^\lambda) \to EK, DK$: Gen takes the security parameter in unary, and outputs an encryption key EK and a decryption key DK.
- $\mathsf{Puncture}(DK, \{c_0, c_1\}) \to DK\{c_0, c_1\}$: $\mathsf{Puncture}$ takes a decryption key DK, and a set $\{c_0, c_1\} \subset \{0,1\}^n$.[5] $\mathsf{Puncture}$ outputs a "punctured" decryption key $DK\{c_0, c_1\}$.

[5]We can assume that c_0 and c_1 are given to $\mathsf{Puncture}$ in sorted order, so $\mathsf{Puncture}(DK, \{c_0, c_1\})$ is identical to $\mathsf{Puncture}(DK, \{c_1, c_0\})$.

- $\mathsf{Enc}(EK, m) \to c$: Enc takes an encryption key EK and a message $m \in \{0,1\}^\ell$, and outputs a ciphertext c in $\{0,1\}^n$.
- $\mathsf{Dec}(DK, c) \to m$ or $\perp$: Dec takes a possibly punctured decryption key DK and a string $c \in \{0,1\}^n$. It outputs a message m or $\perp$.

5.1 Required Properties

Correctness We require that for all messages m,

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

Punctured Correctness We also require the same to hold for keys which are punctured: for all messages m,

$$\Pr\left[\begin{array}{l} \exists c \notin \{c_0, c_1\} \text{ s. t.} \\ \mathsf{Dec}(DK, c) \neq \mathsf{Dec}(DK', c) \end{array} \middle| \begin{array}{l} (EK, DK) \leftarrow \mathsf{Gen}(1^\lambda), \\ c_0 \leftarrow \mathsf{Enc}(EK, m), c_1 \leftarrow \{0,1\}^n, \\ DK' \leftarrow \mathsf{Puncture}(DK, \{c_0, c_1\}) \end{array} \right] \leq \operatorname{negl}(\lambda)$$

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

Game 3 (Ciphertext Pseudorandomness).

1. $\mathcal{A}$ sends a message m^* to the challenger.
2. The challenger samples $(EK, DK) \leftarrow \mathsf{Gen}(1^\lambda)$ and sends EK to $\mathcal{A}$.
3. The challenger computes $c_0 \leftarrow \mathsf{Enc}(EK, m^*)$ and $c_1 \leftarrow \{0,1\}^n$.
4. The challenger generates $DK' \leftarrow \mathsf{Puncture}(DK, c_0, c_1)$.
5. The challenger samples $b \leftarrow \{0,1\}$ and sends c_b, c_{1-b}, EK, DK' to $\mathcal{A}$.
6. The adversary outputs b' and wins if $b = b'$.

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

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

One of our contributions is the following theorem.

Theorem 5.1. *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 and A.2.

6 Main Construction

Theorem 6.1. *There is a watermarking scheme which is both δ-lunchtime unremovable and γ-relaxed unforgeable, for any choice of δ, γ satisfying $\gamma > 2\delta + \frac{1}{\mathrm{poly}(\lambda)}$ for some polynomial* poly.

Proof. In Theorem 6.2, we show that Construction 1 is $(2\delta, \delta)$-lunchtime unremovable for every δ. In Theorem 6.3, we show that Construction 1 is $(1-\gamma, \gamma)$-relaxed unforgeable for every γ.

Theorem 3.1 then implies that given any δ, γ satisfying $\gamma > 2\delta + \frac{1}{\mathrm{poly}(\lambda)}$, Construction 1 can be amplified into a scheme which is simultaneously $(0, \delta)$-lunchtime unremovable and $(0, \gamma)$-relaxed unforgeable. □

Remark 3. We compare the result of Theorem 6.1 to the impossibility discussed in Section 2.2.1. The impossibility states that γ must be at least $\delta + \frac{1}{\mathrm{poly}(\lambda)}$. Theorem 6.1 gives a construction for any γ larger than $2\delta + \frac{1}{\mathrm{poly}(\lambda)}$.

Remark 4. In Appendix C, we describe how to extend this construction to pseudorandom function families with arbitrary output bit-length, with a small loss in parameters.

Our construction marks any PRF family $\{\mathcal{P}_\lambda : \{0,1\}^n \to \{0,1\}^m\}_{\lambda \in \mathbb{N}}$ that is puncturable at a single point, if $n = n(\lambda)$ and $m = m(\lambda)$ are $\Omega(\lambda^\epsilon)$ for some $\epsilon > 0$. Our construction uses the following building blocks.

- Pseudorandom function families $\{\mathcal{F}_\lambda : \{0,1\}^n \to \{0,1\}^m\}_{\lambda \in \mathbb{N}}$ and $\{\mathcal{G}_\lambda : \{0,1\}^\ell \to \{0,1\}^m\}_{\lambda \in \mathbb{N}}$ which are selectively puncturable on any interval. We refer the reader to [BW13] for definitions and constructions of selectively secure puncturable PRF families.
- A puncturable encryption system $\mathcal{PE}$ with plaintexts in $\{0,1\}^\ell$ and ciphertexts in $\{0,1\}^n$, where $\ell = \ell(\lambda)$ is $\Omega(\lambda^\epsilon)$. We explicitly denote the randomness used in $\mathcal{PE}.\mathsf{Enc}$ as r. We construct such a $\mathcal{PE}$ in section 5.
- A collision resistant hash function $\{h_\lambda : \{0,1\}^m \to \{0,1\}^{\ell/2}\}_{\lambda \in \mathbb{N}}$.
- A pseudorandom generator $PRG_1 : \{0,1\}^{\ell/4} \to \{0,1\}^{\ell/2}$, as well as another pseudorandom generator $PRG_2 : \{0,1\}^{\ell/2} \to \{0,1\}^n$.

Construction 1 (Unamplified).

- $\mathsf{Setup}(1^\lambda)$: Setup samples $(DK, EK) \leftarrow \mathcal{PE}.\mathsf{Gen}(1^\lambda)$ and puncturable PRFs $F \leftarrow \mathcal{F}_\lambda$ and $G \leftarrow \mathcal{G}_\lambda$. Setup then outputs $(\mathsf{mk}, \mathsf{vk})$, where $\mathsf{mk} = (DK, F, G)$ and vk is the iO-obfuscation of the program in Figure 3.
- $\mathsf{Mark}(\mathsf{mk}, C)$: Mark outputs the iO obfuscation of the circuit $C_\#$, which is described in Figure 1, but padded to be as big as the largest of the circuits in any of the hybrids.
- $\mathsf{Verify}(\mathsf{vk}, C)$: Verify samples uniformly random bit strings $a \in \{0,1\}^{\ell/4}$ and randomly samples r. Verify then computes $b = h(C(PRG_2(PRG_1(a))))$. Verify evaluates vk (an obfuscated program) on $(a\|b, r)$ to obtain a pair (x, y), and returns 1 if $C(x) = y$; otherwise, it returns 0.

As Verify is an algorithm which runs an obfuscated program vk as a subroutine, we provide the "unrolled" verification algorithm in Figure 3.

We now state the main theorems of this section.

Theorem 6.2. *Construction 1 is $(2\delta, \delta)$-lunchtime unremovable for every δ.*

Theorem 6.3. *Construction 1 is $(1-\gamma, \gamma)$-relaxed unforgeable for every γ.*

```
                         $C_{\#}$
**Constants**: Decoding key $DK$, Puncturable PRFs $F$ and $G$, and a circuit $C$
**Inputs:** $c \in \{0,1\}^n$
  1. Try $t\|b \leftarrow \mathsf{Dec}(DK, x)$, where $t \in \{0,1\}^{\ell/2}$ and $b \in \{0,1\}^{\ell/2}$.
  2. If $t\|b \neq \perp$ and $h(C(PRG_2(t))) = b$, output $F(c) \oplus G(t\|b)$.
  3. Otherwise, output $C(c)$.
```

Figure 1: Program $C_{\#}$

```
                         Verify
**Inputs:** Verification key vk and circuit $C$.
  1. Sample $a \leftarrow \{0,1\}^{\ell/4}$
  2. $b = h(C(PRG_2(PRG_1(a))))$
  3. $r \leftarrow \$$
  4. $(x, y) = \mathsf{vk}(a, b, r)$ :
       ┌─────────────────────────────────────────────┐
                          vk
       **Constants**: Encoding key $EK$, Puncturable PRFs $F$ and $G$
       **Inputs:** $a \in \{0,1\}^{\ell/4}, b \in \{0,1\}^{\ell/2}$, randomness $r$
           Generate $c \leftarrow \mathsf{Enc}(EK, PRG_1(a)\|b; r)$.
           Output $(c, F(c) \oplus G(PRG_1(a)\|b))$.
       └─────────────────────────────────────────────┘
  5. If $C(x) = y$ output 1. Otherwise output 0.
```

Figure 2: Unrolled $\mathsf{Verify}(\mathsf{vk}, C)$. Line 4 expands the execution of the program vk, which is itself an obfuscated program.

```
Constants: Encoding key $EK$, Puncturable PRFs $F$ and $G$
Inputs: $a \in \{0,1\}^{\ell/4}, b \in \{0,1\}^{\ell/2}$, randomness $r$

  1. Generate $x \leftarrow \mathsf{Enc}(EK, PRG_1(a)\|b;\ r)$.

  2. Output $(x, F(x) \oplus G(PRG_1(a)\|b))$.
```

Figure 3: Verification key vk (pre-obfuscated)

6.1 Proof of Lunchtime Unremovability

To prove Theorem 6.2, we first give a distinguishing game and show that no p.p.t. algorithm $\mathcal{A}$ can win this game with non-negligible probability. Because Verify is black-box as in Definition 4.1 and queries exactly 2 points of an argument program, a variant of Theorem 4.1 implies the desired result.[6]

Game 4 (Lunchtime Distinguishing). First, the challenger generates $(\mathsf{mk}, \mathsf{vk})$ by $\mathsf{Setup}(1^\lambda)$. The adversary is presented with vk and access to the following two oracles.

- A marking oracle $\mathcal{O}_M$ which takes a circuit C and returns $\mathsf{Mark}(\mathsf{mk}, C)$.
- A challenge oracle $\mathcal{O}_C$ which takes no input, but samples a circuit C^* uniformly from $\mathcal{C}$ and returns $C^*_\# = \mathsf{Mark}(\mathsf{mk}, C^*)$. The adversary must query $\mathcal{O}_C$ exactly once, after which he may no longer query either oracle.

At the end of the game, the challenger executes $\mathsf{Verify}(\mathsf{vk}, C^*_\#)$. From the points at which Verify queried $C^*_\#$, the challenger picks x_0 uniformly at random. The challenger also chooses x_1 as a random point in the domain of $C^*_\#$. The challenger picks a random bit b and sends x_b to the adversary.

The adversary then outputs a bit b' and wins if $b' = b$. The adversary's advantage is $\left|\Pr[b' = b] - \frac{1}{2}\right|$.

Lemma 6.4. *Every p.p.t. algorithm $\mathcal{A}$ has negligible advantage in Game 4 for Construction 1.*

We will show that security in the last hybrid reduces to the security of the puncturable encryption $\mathcal{PE}$. We will construct an algorithm $\mathcal{B}$ distinguishing $(EK, DK\{x_0, x_1\}, x_0, x_1)$ from $(EK, DK\{x_0, x_1\}, x_1, x_0)$ with non-negligible advantage. To do so, we will have to answer the queries of the watermarking adversary in Game 4 using the $\mathcal{PE}$ challenge. The main challenge therefore is to use only the punctured decryption key and to treat x_0 and x_1 symmetrically.

Proof. We must show that in the game, no p.p.t. algorithm can distinguish x_0 from a random point x_1 with non-negligible advantage. Recall that x_0 is chosen from the points at which Verify queries $C^*_\#$. In Construction 1, there are two such points, one of which is $PRG_2(PRG_1(a^*))$ for randomly chosen $a^* \leftarrow \{0,1\}^{\ell/4}$, and thus pseudorandom. In the rest of the proof, we focus on the other case: x_0 is computed as $\mathsf{Enc}(EK, PRG_1(a^*)\|h^*)$, where $h^* = h(C^*_\#(PRG_2(PRG_1(a^*))))$. We must show that this distribution of x_0 is indistinguishable from a uniform x_1.

[6]As noted in the discussion of Theorem 4.1, we prove that lemma for specific removing and distinguishing games. The proof techniques and result apply directly in this setting as well.

As $\mathcal{A}$'s runtime is bounded by some polynomial q, we define a sequence of hybrid games H_0 through H_{q+4}. Hybrid H_0 is simply Game 4, except that the challenger picks C^*, generates $C^*_{\#}$, and samples x_0 and x_1 appropriately before presenting the adversary with vk. This is possible because these are all chosen independently of the adversary's actions.

In hybrid $\mathbf{H_i}$, the adversary's first i queries $C_1, \dots, C_i$ to $\mathcal{O}_m$ are answered in a modified way (as in Figure 4). These queries are answered using only a *punctured* decryption key $DK' = DK\{x_0, x_1\}$ and *punctured* PRF key $F' = F\{x_0, x_1\}$. In hybrid H_q, we answer all queries to $\mathcal{O}_M$ in this way.[7]

Constants: <u>Punctured</u> decoding key $DK' = DK\{x_0, x_1\}$, <u>Punctured</u> PRF $F\{x_0, x_1\}$, PRF G, and a circuit C_j
Inputs: $x \in \{0,1\}^n$

1. <u>If $x \in \{x_0, x_1\}$, output $C_j(x)$.</u>
2. Try $t\|b \leftarrow \mathsf{Dec}(DK', x)$.
3. If $t\|b \neq \perp$ and $h(C_j(PRG_2(t))) = b$, output $F\{x_0, x_1\}(x) \oplus G(t\|b)$.
4. Otherwise, output $C_j(x)$.

Figure 4: Modified marked program $C_{j\#}$ for $j \leq i$ in hybrid H_i.

Claim 6.4.1. *In H_i, $h(C_{i+1}(PRG_2(PRG_1(a^*)))) = h^*$ with negligible probability.*

Proof. First we note that because $h : \{0,1\}^m \to \{0,1\}^{\ell/2}$ is a compressing collision-resistant hash function, h is also a one-way function. Suppose that in hybrid H_i, $h(C_{i+1}(PRG_2(PRG_1(a^*)))) = h^*$ with non-negligible probability. Then we provide an algorithm Inv that inverts h, or violates the pseudorandomness of C^*. Throughout this proof, let PRG denote $PRG_2 \circ PRG_1$.

Inv takes as input $y = h(r)$ for uniformly sampled $r \leftarrow \{0,1\}^m$. Inv samples random $a^* \leftarrow \{0,1\}^{\ell/2}$, $EK, DK \leftarrow \mathsf{Gen}(1^\lambda)$, $x_1 \leftarrow \{0,1\}^n$, $F \leftarrow \mathcal{F}_\lambda$, and $G \leftarrow \mathcal{G}_\lambda$. Inv samples an encryption of $a^*\|y$, namely $x_0 \leftarrow \mathsf{Enc}(a^*\|y)$. Lastly, Inv punctures both the decryption key and PRF key at $\{x_0, x_1\}$, obtaining DK' and F'.

Using the above, Inv runs $\mathcal{A}$ and answers queries to $\mathcal{O}_M$ as in hybrid H_i. The view of $\mathcal{A}$ in this simulation is indistinguishable from the view in the real hybrid H_i; the only difference is that in the real hybrid, $y = h(r)$ for $r = C^*(PRG(a^*))$ for random $C^* \leftarrow \mathcal{C}$ (instead of uniformly random r). By the pseudorandomness of the family $\mathcal{C}$, the views are indistinguishable.

Finally, $\mathcal{A}$ outputs a circuit C_{i+1}. By our hypothesis, $h(C_{i+1}(PRG(a^*))) = h^*$ with non-negligible probability. Thus, with non-negligible probability, Inv has found a pre-image of y under h, namely $C_{i+1}(PRG(a^*))$ □

The proof of the above claim crucially relies on the fact that when $\mathcal{A}$ queries the circuit C_i, he has no information about C^*. This is why this proof only applies to *lunchtime* unremovability. Using the above claim, we now prove the following:

[7]The reason for puncturing F here is so that later we may make a change to the challenge program (see hybrid H_{q+3}).

Claim 6.4.2. *For $0 \leq i < q$, $H_i \approx H_{i+1}$.*

Proof. The only difference between H_i and H_{i+1} is in the way that the $i+1^{th}$ query is answered. We now show that the two programs returned are functionally equivalent with high probability, and so by the security of iO, the two hybrids are indistinguishable.

By the correctness of the punctured $\mathcal{PE}$ decryption key DK' and the correctness of the punctured PRF F' on non-punctured points, there are only two possible inputs on which $C_{i+1\#}$ may differ in H_i and H_{i+1}: namely, x_0 and x_1.

By the sparseness of ciphertexts in $\mathcal{PE}$, $\mathsf{Dec}(DK, x_1) = \perp$ with high probability over the choice of x_1. Thus in hybrid H_i, $C_{i+1\#}(x_1) = C_{i+1}(x_1)$. This is also true in hybrid H_{i+1}.

On the other hand, x_0 decrypts to $a^* \| h^*$. By the previous claim, the check that $h(C_{i+1}(PRG(a^*))) = h^*$ fails with high probability. Thus in hybrid H_i, $C_{i+1\#}(x_0) = C_{i+1}(x_0)$. This is also true in hybrid H_{i+1}. □ □

From $\mathbf{H_q}$ to $\mathbf{H_{q+4}}$, every query to $\mathcal{O}_M$ is answered as Figure 4. We proceed to modify only $C^*_\#$, with two simultaneous goals in mind. We want x_0 and x_1 to be treated symmetrically in the *challenge* marked program $C^*_\#$. We also want to generate $C^*_\#$ using only the *punctured* decryption key $DK'\{x_0, x_1\}$.

H_{q+1}**:** In hybrid H_{q+1}, we modify $C^*_\#$ as in Figure 5. We puncture C^* at $\{x_1\}$ and F at $\{x_0\}$, and hard-code a mapping $x_0 \mapsto y_0$ and $x_1 \mapsto y_1$. y_0 and y_1 are defined as $y_0 = F(x_0) \oplus G(PRG_1(a^*) \| h^*)$ and $y_1 = C^*(x_1)$. We also puncture the decryption key DK at $\{x_0, x_1\}$. These changes preserve functionality with high probability. F is never evaluated on x_0 because of the hard-coded check in line 1. Similarly, $C^*\{x_1\}$ is not evaluated at x_1 on line 4. Furthermore, with high probability, x_1 is not in the image of PRG. Thus on line 3, $C^*\{x_1\}$ is never evaluated at x_1.

Constants: A $\underline{\text{punctured}}$ decoding key $DK' = DK\{x_0, x_1\}$, $\underline{\text{punctured}}$ PRF $F\{x_0\}$, PRF G, and a $\underline{\text{punctured}}$ circuit $C^*\{x_1\}$, $\underline{\text{values } x_0,\ x_1,\ y_0 = F(x_0) \oplus G(PRG_1(a^*) \| h^*),\ y_1 = C^*(x_1)}$
Inputs: $x \in \{0,1\}^n$

1. $\underline{\text{If } x = x_i \text{ for } i \in \{0,1\}, \text{ output } y_i.}$
2. Try $a \| b \leftarrow \mathsf{Dec}(DK', x)$.
3. If $a \| b \neq \perp$ and $h(C^*\{x_1\}(PRG_2(PRG_1(a)))) = b$, output $F\{x_0\}(x) \oplus G(PRG_1(a) \| b)$.
4. Otherwise, output $C^*\{x_1\}(x)$.

Figure 5: Modified $C^*_\#$ in H_{q+1}

H_{q+2}**:** In hybrid H_{q+2}, we change y_1 to be random. This is an indistinguishable change by the punctured pseudorandomness of C^* at x_1.

H_{q+3}**:** In hybrid H_{q+3}, we change y_0 to be random. This is an indistinguishable change by the punctured pseudorandomness of F at x_0.

H_{q+4}: In hybrid H_{q+4}, we no longer puncture C^* and F. This preserves the functionality of $C^*_{\#}$ by the same reasoning as in hybrid H_{q+1}, hence is an indistinguishable change by the security of iO.

Final Security Argument We now reduce to the security of $\mathcal{PE}$. Given an adversary $\mathcal{A}$ with non-negligible advantage in hybrid H_{q+4}, we construct an adversary $\mathcal{B}$ violating the punctured ciphertext indistinguishability of $\mathcal{PE}$. That is, in Game 3, $\mathcal{B}$ distinguishes $(EK, DK\{x_0, x_1\}, x_0, x_1)$ from $(EK, DK\{x_0, x_1\}, x_1, x_0)$ with the same advantage. $\mathcal{B}$ executes the following steps:

1. $\mathcal{B}$ samples $C^* \leftarrow \mathcal{C}$, picks $a^* \leftarrow \{0,1\}^{\ell/4}$, computes $h^* = h(C^*(PRG_2(PRG_1(a^*))))$, and sends $m = a^* \| h^*$ to the $\mathcal{PE}$ challenger.
2. $\mathcal{B}$ receives $(EK, DK\{x_0, x_1\}, x_b, x_{1-b})$ as input, where $x_0 \leftarrow \mathsf{Enc}(m)$, $x_1 \leftarrow \{0,1\}^n$ for some unknown $b \in \{0,1\}$.
3. $\mathcal{B}$ samples $F \leftarrow \mathcal{F}$ and $G \leftarrow \mathcal{G}$ to construct the verification vk as in Setup, and runs $\mathcal{A}$ on vk. $\mathcal{B}$ answers $\mathcal{A}$'s queries to $\mathcal{O}_M$ and $\mathcal{O}_C$ as in H_{q+4}.
4. At the end of the game, $\mathcal{B}$ sends x_b to $\mathcal{A}$. $\mathcal{A}$ outputs a bit b', and $\mathcal{B}$ also outputs b'.

In this execution, $\mathcal{A}$'s view is exactly the same as in H_{q+4}. The b in Game 4 is the same as the b in Game 3. □

6.2 Proof of Relaxed Unforgeability

For clarity, we recall the security game of (q,γ)-relaxed unforgeability. In the theorem statement, $q = 1 - \gamma$.

Game 5 ((q,γ)-relaxed Unforgeability).
First, the challenger generates $(\mathsf{mk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$. The adversary is presented with vk and access to a marking oracle $\mathcal{O}_M$, which takes a circuit C_i and returns $C_{i\#} = \mathsf{Mark}(\mathsf{mk}, C_i)$.

At the end of the game, the adversary outputs a circuit $\hat{C}$. The adversary wins if the following conditions hold:

1. $\Pr\left[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1\right] \geq q + \frac{1}{\mathrm{poly}(\lambda)}$
2. $\Pr\left[\forall i, \hat{C}(x) \neq C_i(x) \middle| x \leftarrow \{0,1\}^n\right] \geq \gamma$.

Remark 5. As in the proof of Theorem 4.1, we will analyze $\mathcal{A}$'s winning probability conditioned on $\mathcal{A}$ outputting a $\hat{C}$ that satisfies condition 2. This isn't an efficiently testable event because the threshold of γ is too sharp.[8] However, as discussed in Theorem 4.1, we can *relax* this condition in a way which only increases $\mathcal{A}$'s advantage. Even in this game, the adversary has a negligible chance of winning. For clarity, we omit these technical details and just assume that $\mathcal{A}$ always outputs such a $\hat{C}$.

Proof. We will construct a series of games H_0 through H_3 between a challenger and an adversary. Each game defines some random variables. In particular, each game defines the following variables:

[8]in fact, $\#P$-complete!

- The queries $C_1, \ldots, C_T$ made by the adversary (T is a bound on the adversary's running time)
- The candidate forgery $\hat{C}$ output by the adversary at the end of the game
- A randomly chosen t^*. This, together with $C^*, C_1, \ldots, C_T$ defines $h_1, \ldots, h_T$ and h^*. Specifically, we define $h_i = h(C_i(PRG_2(t^*)))$, and $h^* = h(\hat{C}(PRG_2(t^*)))$.
- x^*

In our security proof, we consider two events:

1. The event that $\hat{C}(x^*) = F(x^*) \oplus G(t^* \| h^*)$. In H_0, the probability of this event is exactly the probability that $\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1$ in Game 5.
2. The event that for some $i \in \{1, \ldots, T\}$, $h_i = h^*$. We will upper bound the probability of this "bad" event to upper bound the probability that $\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1$.

H_0**:** Our first hybrid H_0, is defined as follows: the challenger runs Game 5 with Construction 1, as well as sampling a few extra variables. The extra variables are $a^* \leftarrow \{0,1\}^{\ell/4}$ and $t^* = PRG_1(a^*)$.

That is, the challenger first samples $(EK, DK) \leftarrow \mathcal{PE}.\mathsf{Gen}(1^\lambda)$, and samples $F \leftarrow \mathcal{F}$ and $G \leftarrow \mathcal{G}$. These are used to define marking keys $(\mathsf{mk}, \mathsf{vk})$ as in $\mathsf{Setup}(1^\lambda)$. The challenger sends vk to $\mathcal{A}$.

When $\mathcal{A}$ makes a query C_i, the challenger responds with $C_{i\#} = \mathsf{Mark}(\mathsf{mk}, C_i)$. If the adversary produces a candidate forgery $\hat{C}$, x^* is defined as as $\mathsf{Enc}(EK, t^* \| h^*)$, where $h^* = h(\hat{C}(PRG_2(t^*)))$.

Claim 6.4.3. *In H_0, event 2 happens with probability at most $1 - \gamma + \mathrm{negl}(\lambda)$.*

Proof. Here we just use the collision resistance of h, the pseudorandomness of PRG_2, and the fact that in Game 5, the adversary is restricted to producing $\hat{C}$ such that $\hat{C}$ differs from *every* C_i on at least a γ fraction of the domain. □

Claim 6.4.4. *In H_0, event 1 happens with probability at most $1 - \gamma + \mathrm{negl}(\lambda)$*

Proof. We give indistinguishable hybrids H_1 through H_3, such that in hybrid H_3, the probability of event 1 is bounded by the probability of event 2, which by the previous claim is $1 - \gamma + \mathrm{negl}(\lambda)$.

H_1: In hybrid H_1, $\mathcal{B}$ samples $t^* \leftarrow \{0,1\}^{\ell/2}$ instead of generating $t^* = PRG_1(a^*)$. This is indistinguishable by the pseudorandomness of PRG_1.

H_2: In hybrid H_2, $\mathcal{B}$ sends a modified vk to the adversary. In this vk, the puncturable PRF G is punctured at the set of strings beginning with t^*. This is functionally equivalent because with high probability t^* is not in the image of PRG_1. Indistinguishability thus follows from the security of iO.

H_3: In hybrid H_3, $\mathcal{B}$ answers the queries to $\mathcal{O}_M$ in a modified way. In particular, each marked response $C_{i\#}$ will have G punctured on the set of strings beginning with t^*. $C_{i\#}$ will also have the values t^*, h_i, and $G(t^* \| h_i)$ hard-coded where $h_i = h(C_i(PRG_2(t^*)))$. We modify $C_{i\#}$ by the correct hard-coded value on the only punctured point $(t^* \| h_i)$ on which G will ever be evaluated. Indistinguishability thus follows from the security of iO.

When event 2 does not happen, the probability that $\hat{C}(x^*) = F(x^*) \oplus G(t^* \| h^*)$ is negligible. □

This concludes the proof of Theorem 6.3. □

7 The Limits of Watermarking

In this section, we show a barrier to the construction of a general watermarking result for several cryptographic objects. In particular, we show that assuming the existence of robust totally unobfuscatable functions (defined below), there are signature schemes and private-key encryption algorithms that cannot be watermarked, even if they are only required to satisfy the unremovability property.

A totally unobfuscatable function family $\mathcal{F} = \{f_K\}_{K \in \{0,1\}^n}$, defined by Barak et al. [BGI+12], is one where oracle access to a random function f_K from the family does not help an adversary learn the function key K, yet given any circuit C that computes the function f_K (exactly), it is possible to extract K in polynomial time. A *robust* totally unobfuscatable function family is one where extraction succeeds even given a circuit C that approximates f_K well, namely C computes f_K on all but a $\rho(n)$ fraction of inputs. Barak et al. construct a totally unobfuscatable function family assuming one-way functions. Bitansky and Paneth [BP12] define robust unobfuscatable functions, a weaker variant of the definition above where the extractor is only required to output a hard-to-learn predicate $\pi(K)$ of the function key K, and construct it from one-way functions.

To the best of out knowledge, it is open how to construct robust and totally unobfuscatable function families. We show that such families give rise to *unmarkable* signature schemes and private-key encryption algorithms. Our proof closely resembles that of Barak et al. [BGI+12] who construct unobfuscatable signature and private-key encryption algorithms.

Definition 7.1 (Robust Totally Unobfuscatable Functions)**.** A function ensemble $\mathcal{F} = \{\mathcal{F}_n\}_{n \in \mathbb{N}}$ where each family $\mathcal{F}_n = \{f_K\}_{K \in \{0,1\}^n}$ is a *robust totally unobfuscatable* family if it is:

- *Black-box Non-learnable:* There is a function $\pi : \{0,1\}^n \to \{0,1\}^\lambda$ such that for any poly-size family of circuits $L = \{L_n\}_{n \in \mathbb{N}}$, and all large enough $n \in \mathbb{N}$,

$$\Big| \Pr_{K \leftarrow \{0,1\}^n}[L_n^{f_K}(\pi(K)) = 1] - \Pr_{K \leftarrow \{0,1\}^n, z \leftarrow \{0,1\}^\lambda}[L_n^{f_K}(z) = 1] \Big| \le \mathrm{negl}(n)$$

- *(ρ-Robustly) Non-black-box Learnable:* There exists an efficient extractor E such that for all large enough $n \in \mathbb{N}$ and any $K \in \{0,1\}^n$, E extracts K from any circuit C such that $C \sim_{\rho(n)} f_K$:

$$\Pr_E[K \leftarrow E(C, 1^n)] \ge 1 - \mathrm{negl}(n) \cdot \mathsf{poly}(|C|)$$

  where poly is a fixed polynomial that depends only on E.

Theorem 7.1. *Assuming the existence of one-way functions and robust totally unobfuscatable functions, there is a signature scheme $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ that is existentially unforgeable against chosen message attacks for which there is no watermarking scheme satisfying Definition 2.3.*

Proof. (Sketch.) Assume the existence of a robust totally unobfuscatable family of functions $\mathcal{F} = \{f_K\}_{K \in \{0,1\}^n}$ as in Definition 7.1. Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme that is existentially unforgeable against a chosen-message attack. We construct a signature scheme $\mathcal{S}' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ as follows.

- $\mathsf{Gen}'(1^\lambda)$ generates keys $(SK, VK) \leftarrow \mathsf{Gen}(1^\lambda)$. It also samples a uniformly random function $f_K \leftarrow \mathcal{F}$.

  Output the secret key $SK' = (SK, K)$ and the verification key $VK' = (VK, \pi(K) \oplus SK)$ where π is the function from the unlearnability condition in Definition 7.1.

- $\mathsf{Sign}'(SK', m)$ computes $\sigma \leftarrow \mathsf{Sign}(SK, m)$ and outputs $(\sigma, f_K(m))$.
- $\mathsf{Verify}'(VK', (\sigma_1, \sigma_2))$ ignores σ_2 and outputs $\mathsf{Verify}(VK, \sigma_1)$.

We first claim that $\mathcal{S}'$ is a signature scheme that is existentially unforgeable against a chosen-message attack. First, oracle access to Sign' gives the adversary oracle access to f_K, meaning that $\pi(K)$ is pseudo-random and so is $\pi(K) \oplus SK$ in the verification key. Once this is done, security of $\mathcal{S}$ implies that of $\mathcal{S}'$.

Secondly, this is an unmarkable signature scheme. Indeed, given any marked program that agrees with Sign' on more than an $1 - \rho(n)$ fraction of messages, we have a program that computes f_K on more than an $1 - \rho(n)$ fraction of inputs. By robust total extraction, we get K from such a program and therefore SK. □

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.

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

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 501–519, 2014.

[BP12] Nir Bitansky and Omer Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 223–232. IEEE Computer Society, 2012.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, volume 8349 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2014.

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

[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. IEEE Computer Society, 2013.

[GLSW14] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[HMW07] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 362–382. Springer, 2007.

[KPTZ13] 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.

[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.

[Nis14] Ryo Nishimaki. How to watermark cryptographic functions. *IACR Cryptology ePrint Archive*, 2014:472, 2014.

[NSS99] David Naccache, Adi Shamir, and Julien P. Stern. How to copyright a function? In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings*, volume 1560 of *Lecture Notes in Computer Science*, pages 188–196. Springer, 1999.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517. Springer, 2014.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484. ACM, 2014.

A Puncturable Encryption Construction

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 $PRG : \{0,1\}^{\ell} \to \{0,1\}^{2\ell}$
- A puncturable, injective family of PRFs $\{\mathcal{F}_\lambda : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}\}$. We require $\mathcal{F}_\lambda$ to be selectively puncturable on any pair of intervals.[9]
- A puncturable family of PRFs $\{G_\lambda : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}\}$. We require $\mathcal{G}_\lambda$ to be selectively puncturable on any pair of points.
- A injective bit commitment Commit 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 2 (Puncturable Encryption Scheme)*.*

- $\mathsf{Gen}(1^\lambda)$ samples a function $F \leftarrow \mathcal{F}$ and $G \leftarrow \mathcal{G}$, and generates EK as the iO-obfuscation of the circuit in Figure 6. Gen returns (EK, DK), where DK is the (un-obfuscated) program in Figure 7.
- $\mathsf{Puncture}(VK, c_0, c_1)$ outputs DK', where DK' is the iO-obfuscation of the program described in Figure 8.
- $\mathsf{Enc}(EK, m)$ takes $m \in \{0,1\}^{\ell}$ and outputs $EK(m) \in \{0,1\}^{12\ell}$.
- $\mathsf{Dec}(DK, c)$ takes $c \in \{0,1\}^{12\ell}$ and returns $DK(c)$.

Remark 6. We note that in all of our obfuscated programs (including the hybrids), whenever α_0 and α_1 or β_0 and β_1 or γ_0 and γ_1 are treated symmetrically, then we can and do store them in lexicographical order. A random ordering would also suffice for security.

```
Pre-obfuscated $EK$
Constants: Puncturable injective PRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, Puncturable PRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$
Inputs: $m \in \{0,1\}^{\ell}, r \in \{0,1\}^{\ell}$
    1. Compute $\alpha = PRG(r)$.
    2. Compute $\beta = F(\alpha \| m)$.
    3. Compute $\gamma = G(\beta) \oplus m$.
    4. Output $(\alpha, \beta, \gamma)$.
```

Figure 6: Program describing how to encode

[9]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.

```
Decoding key $DK$
Constants: Puncturable injective PRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, Puncturable PRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$
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 7: Program describing how to decode

```
Pre-obfuscated $DK'$ punctured at $c_0$ and $c_1$
Constants: $c_0$, $c_1$, a puncturable injective PRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, and a puncturable PRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$
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 8: Program describing how to decode

Correctness Correctness is obvious, observing in the punctured case that DK' is defined to be functionally equivalent to DK except on inputs c_0 and c_1.

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

A.2 Ciphertext Pseudorandomness

We give a sequence of hybrids H_0 through H_{14}. The goal of the hybrids to reach a game in which c_0 and c_1 are treated symmetrically in EK and DK', 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.

H_0 Hybrid H_0 is defined as the real security game:

1. $\mathcal{A}$ sends a message m^* to the challenger.
2. The challenger samples an injective PRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$ which is selectively puncturable on an arbitrary prefix, and a selectively puncturable PRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$.

3. The challenger generates EK as the iO-obfuscation of Figure 6 and sends EK to $\mathcal{A}$.

4. The challenger samples $r \leftarrow \{0,1\}^\ell$, computes $\alpha_0 = PRG(r) \in \{0,1\}^{2\ell}$, $\beta_0 = F(\alpha \| m^*)$, and $\gamma_0 = G(\beta) \oplus m^*$. Define c_0 as $\alpha_0 \| \beta_0 \| \gamma_0$.

5. The challenger samples $c_1 \leftarrow \{0,1\}^{12\ell}$. Parse c_1 as $\alpha_1 \| \beta_1 \| \gamma_1$.

6. The challenger generates DK' as the iO-obfuscation of Figure 8.

7. The challenger samples $b \leftarrow \{0,1\}$ and sends (c_b, c_{1-b}, EK, DK') to $\mathcal{A}$.

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

H_1: In hybrid H_1, we sample α_0 uniformly at random from $\{0,1\}^{2\ell}$.

H_2: In hybrid H_2, we puncture F in EK on all strings of the form $\alpha_0 \| \star$ or $\alpha_1 \| \star$. This is functionally equivalent because F is never evaluated on strings of these forms because α_0 and α_1 are with high probability not in the image of PRG.

H_3: In hybrid H_3, we add the following line in the beginning of DK': "If $c = \alpha_1 \| \beta^* \| \gamma^*$, output m^*", where we hard-code $\beta^* = F(\alpha_1 \| m^*)$ and $\gamma^* = G(\beta^*) \oplus m^*$. This change is functionally equivalent.

H_4: In hybrid H_4, we add the following line (after the line added in Hybrid H_3) to DK': "If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$." This is functionally equivalent by two cases:

1. When $(\alpha, m) = (\alpha_0, m^*)$, then either $c = c_0$, in which case DK' already would output $\perp$, or $c \neq c_0$, in which case DK' rejects c as an invalid ciphertext.

2. When $(\alpha, m) = (\alpha_1, m^*)$, we only reach this line if $c \neq \alpha_1 \| \beta^* \| \gamma^*$ (by the check introduced in Hybrid H_3). In this case, DK' rejects c as an invalid ciphertext.

For reference, we describe DK' from Hybrid H_4 in Figure 9.

Constants: c_0, c_1, a puncturable injective PRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, and a puncturable PRF $G : \{0,1\}^{9\ell} \to \{0,1\}^\ell$. Also the values α_0, α_1, β^*, γ^*, 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 $\beta = \beta^*$ and $\gamma = \gamma^*$, output $m^*$.
2. If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
3. If $c \in \{c_0, c_1\}$, output $\perp$.
4. Compute $m = G(\beta) \oplus \gamma$.
5. If $\beta = F(\alpha \| m)$, output $m$.
6. Else output $\perp$.
```

Figure 9: DK' as in Hybrid H_4, pre-obfuscation

H_5**:** In hybrid H_5, we puncture F at $\{\alpha_0 \| m^*, \alpha_1 \| m^*\}$ in DK'. This is functionally equivalent because by the checks added in the previous hybrid, F will never be evaluated on such inputs.

H_6**:** In hybrid H_6, we sample β^* uniformly at random from $\{0,1\}^{9\ell}$. This is indistinguishable by the pseudorandomness of F at punctured points.

H_7**:** In hybrid H_7, we replace the check "$\beta = \beta^*$" (in line 1 of Figure 9) with the check "$\mathsf{Commit}(0; \beta) = z^*$", where z^* is hard-coded as $\mathsf{Commit}(0; \beta^*)$. This is functionally equivalent by the injectivity of Commit.

H_8**:** In hybrid H_8, z^* is hard-coded as "$\mathsf{Commit}(1; \beta^*)$. Indistinguishability is by the computational hiding of Commit.

H_9**:** In hybrid H_9, we replace the expression "$\mathsf{Commit}(0; \beta) = z^*$" with FALSE. This is functionally equivalent with high probability because of the perfect binding of Commit (which follows from injectivity). In fact, we remove the entire line 1, which also preserves functionality.

For reference, we describe DK' from Hybrid H_9 in Figure 11.

Constants: c_0, c_1, a punctured injective PRF $F' : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$ punctured at $\{\alpha_0 \| m^*, \alpha_1 \| m^*\}$, and a puncturable PRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$. Also the values α_0, α_1, 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 $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
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 10: DK' as in Hybrid H_4, pre-obfuscation

H_{10}**:** In hybrid H_{10}, we sample β_0 uniformly at random from $\{0,1\}^{9\ell}$. This is indistinguishable by the pseudorandomness of F at the (selectively) punctured points.

H_{11}**:** In hybrid H_{11}, we puncture G in EK on $\{\beta_0, \beta_1\}$. This is functionally equivalent by the sparsity of F: β_0 and β_1 are now chosen at random, thus with high probability they are not in the image of F.

H_{12}**:** In hybrid H_{12}, the line "If $c \in \{c_0, c_1\}$: output $\perp$" (in line 2 of DK' in Figure 11) is replaced by "If $\beta \in \{\beta_0, \beta_1\}$: output $\perp$". To see that this change is functionally equivalent, we observe that with high probability, neither of these lines has any effect.

Since β_0 and β_1 are with high probability not in the image of F, if $\beta \in \{\beta_0, \beta_1\}$ – which is the case when $c \in \{c_0, c_1\}$ – then $DK'(c) = \perp$ with high probability, even without the extra check.

The obvious question, then, is why 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.

H_{13}: In hybrid H_{13}, we puncture G at $\{\beta_0, \beta_1\}$ in DK'. This change is functionally equivalent because of the ostensibly useless checks in the previous hybrid.

H_{14}: In hybrid H_{14}, we sample γ_0 uniformly at random from $\{0,1\}^{\ell}$. This change is indistinguishable by the selective indistinguishability of G at the punctured set.

For reference, we describe DK' from Hybrid H_9 in Figure 11. In this hybrid, $c_0 = \alpha_0 \| \beta_0 \| \gamma_0$ and $c_1 = \alpha_1 \| \beta_1 \| \gamma_1$ are now treated symmetrically in both EK and DK'. Furthermore, they are both sampled uniformly and independently at random from $\{0,1\}^{12\ell}$. So no adversary has any advantage in this hybrid.

Constants: c_0, c_1, a punctured injective PRF $F' : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$ punctured at $\{\alpha_0 \| m^*, \alpha_1 \| m^*\}$, and a punctured PRF $G' : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$ punctured at $\{\beta_0, \beta_1\}$. Also the values α_0, α_1, 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 $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
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 11: DK' as in Hybrid H_4, pre-obfuscation

B Proof of Theorem 3.1

Proof. Let $\tau = \frac{q+(1-p)}{2}$. We define $\mathsf{Verify}_{p,q}(\mathsf{vk}, C)$ the following algorithm that repeatedly evaluates $\mathsf{Verify}(\mathsf{vk}, C)$ a total of $T = \frac{8\lambda}{\alpha^2}$ times, with independent randomness on each trial.

$$\mathsf{Verify}_{p,q}(\mathsf{vk}, C) \begin{cases} 1 & \text{if } \frac{1}{T}\sum_{i=1}^{T} \mathsf{Verify}(\mathsf{vk}, C) > \tau \\ 0 & \text{if } \frac{1}{T}\sum_{i=1}^{T} \mathsf{Verify}(\mathsf{vk}, C) \leq \tau \end{cases}$$

As Setup and Mark are unchanged, the new scheme preserves functionality to the same extent as the original.

To show completeness, we observe that for all $C \in \mathcal{C}$ and $C_{\#} \leftarrow \mathsf{Mark}(\mathsf{mk}, C)$, $\mathsf{Verify}_{p,q}(\mathsf{vk}, C_{\#}) = 0$ if at least a $(1-\tau)$ fraction of the independent executions of $\mathsf{Verify}(\mathsf{vk}, C_{\#})$ return 0. By the completeness of the original scheme, $\Pr[\mathsf{Verify}(\mathsf{vk}, C_{\#}) = 0] \leq \mathrm{negl}(\lambda)$ in each independent run. Observing that $1-\tau$ is significantly larger than $\frac{\alpha}{4}$, and applying a standard Hoeffding bound yields $\Pr[\mathsf{Verify}_{p,q}(\mathsf{vk}, C_{\#}) = 0] \leq e^{-\lambda}$.

Next, we must show that $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify}_{p,q})$ is $(0, \delta)$-unremovable. That is, for all p.p.t. $\mathcal{A}$ playing the watermarking security game and outputting $\hat{C}$, if $\hat{C}$ is δ-close to a challenge program $C' \in \mathfrak{C}$, then $\Pr[\mathsf{Verify}_{p,q}(\mathsf{vk}, \hat{C}) = 0]$ is negligible. Equivalently, we must show that

$$\Pr[\sum_{i=1}^{T} \mathsf{Verify}(\mathsf{vk}, \hat{C}) \leq \tau \cdot T]$$

is negligible. By the (p, δ)-unremovability of the watermarking scheme, we know that $\Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1] > 1 - p - \mathrm{negl}(\lambda)$, lower-bounding the one-shot probability of verification. Combined with the definition of τ, this implies that $\Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1] - \tau$ is significantly larger than $\frac{\alpha}{4}$. Applying a standard Hoeffding bound yields $\Pr[\mathsf{Verify}_{p,q}(\mathsf{vk}, \hat{C}) = 0] \leq e^{-\lambda}$.

Lastly, we must show that $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify}_{p,q})$ is $(0, \gamma)$-unforgeable. That is, for all p.p.t. $\mathcal{A}$ playing the watermarking security game and outputting $\hat{C}$, if $\hat{C}$ is γ-far from all marked programs in $\mathfrak{M} \cup \mathfrak{C}$, then the probability $\Pr[\mathsf{Verify}_{p,q}(\mathsf{vk}, \hat{C}) = 1]$ is negligible. Equivalently, we must show that the probability $\Pr[\sum_{i=1}^{T} \mathsf{Verify}(\mathsf{vk}, \hat{C}) > \tau \cdot T]$ is negligible. By the (q, γ)-unforgeability of the watermarking scheme, we know that $\Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1] < q + \mathrm{negl}(\lambda)$, upper-bounding the one-shot probability of verification. Combined with the definition of τ, this implies that $\tau - \Pr[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1]$ is significantly larger than than $\frac{\alpha}{4}$. Aplying a standard Hoeffding bound yields $\Pr[\mathsf{Verify}_{p,q}(\mathsf{vk}, \hat{C}) = 1] \leq e^{-\lambda}$. $\square$

C Multi-bit Equivalence

In our main theorem, we state that there exist watermarking schemes for puncturable pseudorandom function families with long output lengths (specifically, $\Omega(\lambda^{\epsilon})$ for some constant $\epsilon > 0$.

However, we note that there is a simple reduction, allowing us to watermark puncturable pseudorandom function families with arbitrary output lengths, suffering only a small loss in parameters. The idea is simple: any pseudorandom function with multi-bit outputs can be viewed as a PRF with single-bit outputs, with a slightly expanded input space.

Concretely, if we have a (p, δ)-unremovable watermarking scheme for a PRF family $\mathcal{F}$ with m-bit outputs, then this is easily seen to be a $(p, \delta/m)$-unremovable watermarking scheme for the $\mathcal{F}$ interpreted as a PRF family with single-bit outputs. If the scheme for $\mathcal{F}$ is (q, γ)-unforgeable, then it is also (q, γ)-unforgeable for $\mathcal{F}$ when interpreted as a family with single-bit outputs.

Dually, given a scheme for a family with single-bit outputs, we can analyze its parameters when construed as a scheme for an m-bit PRF family. A (p, δ)-unremovable scheme for a single-bit family is also (p, δ)-unremovable for the same family construed as an m-bit family. A (q, γ)-unforgeable scheme for the single-bit family becomes a $(q, m\gamma)$-unforgeable scheme.