Watermarking Public-key Cryptographic Functionalities and Implementations*

Foteini Baldimtsi[†1], Aggelos Kiayias[†2], and Katerina Samari[‡3]

[1]George Mason University, USA
[2]University of Edinburgh and IOHK, UK
[3]National and Kapodistrian University of Athens, Greece

Abstract

A watermarking scheme for a public-key cryptographic functionality enables the embedding of a mark in the instance of the secret-key algorithm such that the functionality of the original scheme is maintained, while it is infeasible for an adversary to remove the mark (unremovability) or mark a fresh object without the marking key (unforgeability). Cohen et al. [STOC'16] has provided constructions for watermarking arbitrary cryptographic functionalities; the resulting schemes rely on indistinguishability obfuscation (iO) and leave two important open questions: (i) the realization of both unremovability and unforgeability, and (ii) schemes the security of which reduces to simpler hardness assumptions than iO.

In this paper we provide a new definitional framework that distinguishes between watermarking cryptographic functionalities and implementations (think of ElGamal encryption being an implementation of the encryption functionality), while at the same time provides a meaningful relaxation of the watermarking model that enables both unremovability and unforgeability under minimal hardness assumptions. In this way we can answer questions regarding the ability to watermark a *given* implementation of a cryptographic functionality which is more refined compared to the question of whether a watermarked implementation of the functionality exists. Taking advantage of our new formulation we present the first constructions for watermarking public key encryption that achieve both unremovability and unforgeability under minimal hardness assumptions. Our first construction enables the watermarking of *any public-key* encryption *implementation* assuming only the existence of one-way functions for private key detection. Our second construction is at the functionality level and uses a stronger assumption (existence of identity-based encryption (IBE)) but supports public detection of the watermark.

*This is the full version of the paper [2].

†This research was partly performed while at the National Kapodistrian University of Athens and supported by the ERC project CODAMODA, #259152.

‡This research was supported by the ERC project CODAMODA, #259152.

1 Introduction

Watermarking digital objects like pictures, video or software is usually achieved by embedding a special piece of information, the *mark*, into the object so that it is difficult for an adversary to remove it without damaging the object itself, or to introduce a fresh and legible mark. At the same time, the embedding of the mark should not result to a significantly different object, or an object with different functionality. Watermarking in practice is particularly useful, and widely applied, in order to protect content creators against illegal use and distribution of copyrighted digital objects. A plethora of watermarking schemes exists in the literature [1, 13, 27, 28] (and references therein), most of them focusing on watermarking "static" objects while lacking a rigorous theoretical analysis and provable secure constructions.

The first formal security definitions for watermarking objects were given by Barak et al. [3, 4] and by Hopper et al. [17]. Barak et al. [3, 4] proposed definitions for software watermarking and showed impossibility relations between program obfuscation and watermarking, while Hopper et al. [17] defined watermarking of perceptual objects without providing any constructions. Nishimaki [23], inspired by the work of [17], extended their definitions to formalize watermarking of *cryptographic functions/ circuits* and defined the security properties to be: correctness, functionality-preserving, unremovability and unforgeability.

Watermarking cryptographic functions has various real-life applications. Consider for instance the case of VPN clients. An organization might wish to distribute VPN clients to its employees where every employee has a public/secret-key pair. Watermarking the VPN client restricts the employees from sharing their clients since, due to the unremovability and unforgeability property, given any client one could detect to whom does this client belongs to (assuming the ID of the user is embedded in the watermark).

Nishimaki [23] provided the first construction of a cryptographic watermarking. While proven secure, the scheme is still vulnerable to a general obfuscation attack described in [4]; Cohen at al. [10] (merged result of [11, 25]) gave a watermarking scheme for puncturable PRFs [19, 7, 8] which avoids the impossibility result of [4] by allowing statistical correctness, i.e. the marked PRF is allowed to behave differently in a negligible fraction of inputs, comparing to the initial one. The construction suggested in [10] is based on the assumption that indistinguishability obfuscation (iO) exists, allows for public key detection and it provably satisfies the unremovability property.

Our results. Our contributions are both in definitional and constructional level. We start by rethinking the definitional framework for watermarking public-key cryptographic functionalities. We approach cryptographic watermarking, by making a relaxation and refinement to the model considered in previous works, which we argue maintains all the relevant to practice features that the previous formulations enjoyed, and moreover can be very suitable for some real world scenarios due to its more refined nature. Previous approaches [23, 25, 11, 10] considered a watermarking definition where the marking algorithm would take as input a *specific* unmarked program/circuit and would output the marked version of it, i.e., a program that *preserves the functionality* of the one given as input to Mark. The origins of this thinking are in the work of [17], that dealt with the cryptographic formalization of watermarking in general.

An important observation that motivates our modeling is that limiting the interaction between the marking system and the recipient of the object in the above fashion is unnecessarily restrictive. In most, if not all, applications of public-key cryptography, the actual details of the decryption or signing program are not relevant to its user, only its functionality is (which encompasses its correctness and security properties). For instance, in the VPN scenario we described above, the organization (i.e. the marking system) is often the one to sample a key K_U for its client and provide it along with the VPN client. Thus we argue that in practice, any interaction between the marking system and the recipient that results in the sampling of a decryption or signing program would be sufficient for an application of watermarking. Following the above reasoning, we propose a new version of watermarking definitions where the Mark algorithm does not take a specific program as input[1] but instead it partitions[2] the exponential space of available secret-key program instances into *marked* and *unmarked* (taking advantage of the marking key) and whenever queried it samples and returns a program from the *marked* space. This extends to the case of embedding a watermark in the form of a message msg, in which setting, the space is partitioned further labeled by the different messages that may be embedded.

In our model, we define watermarking for public-key cryptographic *functionalities* as well as cryptographic *implementations*. Distinguishing between the two is a further refinement of the definitional framework and relevant from a real world point of view. Specifically, in all previous works the focus was in the watermarking of a cryptographic functionality, in the sense of constructing a *new* scheme (say, public-key encryption or digital signature) for which one can argue the basic properties of watermarking (unremovability, unforgeability, functionality preserving) or watermarking a circuit directly. In other words, the starting point was the cryptographic functionality and the solution was a specific construction realizing it or the starting point was a fixed program. While this is sensible as in the first case it permeates the way cryptographic primitives are proposed and realized in general and in the second it resembles the definition of obfuscation, for the case of watermarking it appears also important to be able to watermark a *specific cryptographic implementation* of a functionality, which is a probability distribution ensemble of programs (with each sample containing both code and keys). In plain words, a marking service may want to watermark, say, ElGamal public-key encryption because this particular implementation of public-key encryption is the one that is standardized, backwards compatible, or sufficiently efficient for the context within which the cryptographic system is used. This of course can be achieved by watermarking a circuit implementing ElGamal decryption but definitionally this can be relaxed and the objective, can be seen to lie in between the objectives of designing a watermarked public-key encryption and watermarking arbitrary circuits.

Following the above we formulate secure watermarking both for the case of functionalities and implementations, focusing on the public-key setting. We also validate our model by showing that watermarking a given implementation of a functionality is a stronger notion

[1] In [6] a similar relaxation of the marking algorithm is given, in the sense that the algorithm does not receive as input a specific circuit to be marked, but instead samples a key to be marked and returns it together with the marked circuit. However, their watermarking model is restricted to watermarking PRFs only.

[2] This partition of the space to marked and unmarked programs is the reason why the impossibility result of [4] does not apply in our setting – applying iO to a marked program in our model would not remove the marking.

than merely watermarking a functionality (i.e., producing a watermarked implementation of the functionality). Note that, existing work in formalizing watermarking is either done for circuit classes [10] or even more restricted, for pseudorandom functions [21]. Our definition is more general, encompasses any public-key cryptographic functionality and implementation and is consistent to existing work. Cohen et al. [10] attempted to provide *specific* definitions for watermarking public-key cryptographic primitives, i.e. "Watermarkable Public Key Encryption and "Watermarkable Signature Scheme". Our definitional framework is more general and encompasses any public-key cryptographic functionality and implementation and is consistent with theirs for these functionalities. Thus, any construction that is described in their model for watermarkable encryption and signatures will be syntactically compliant and secure in our more general model as well.

Once we set our new definitional model we present two constructions. In Section 5 we propose a scheme for watermarking cryptographic implementations, precisely a watermarking scheme for watermarking *any* public key encryption implementation. This construction works for private detection of watermarked programs. It assumes a shared state of logarithmic size in the security parameter between the Mark and Detect algorithms while the running time of the detection algorithm depends on the number of marked programs so far. We stress that these relaxations to the notion of watermarking do not appear to hurt the applicability of the scheme in a real world setting, where e.g., an organization wishes to issue watermarked versions of cryptographic algorithms (embedded in VPN clients). In such scenarios private detection is the default requirement and given that detection of malicious clients happens with much lower frequency compared to marking, a detection process with linear running time to the number of clients can be reasonable. Countering these downsides, our construction enjoys security against *both* unremovability and unforgeability attacks, actually achieving *unconditional* unremovability for any public-key encryption implementation. Moreover, the only assumption needed for unforgeability is the existence of one-way functions (that we utilize as a facilitator for a PRF function). This suggests that the security of watermarking comes essentially "for free" since the security of the underlying public-key encryption would imply the existence of one-way functions already.

Our second construction achieves watermarking for the public key encryption *functionality*. It is based in identity-based encryption (IBE) [5], also assumes a shared state of logarithmic size in the security parameter between the Mark and Detect but, as opposed to our first construction, it allows for public key detection of the watermark. This is the *first* construction in the literature for watermarking a cryptographic functionality with public-key detection when only based on standard assumptions (i.e. without using iO).

In a high level, both our constructions exploit the notion of a PRF [16], to create a compact "dictionary" of marked objects that is subsequently scanned and compared with the adversarial implementation. Our proposed constructions are simple and use well-known building blocks and are secure under minimal standard assumptions. Despite their simplicity, our schemes require a very careful analysis in order to comply with the complex security properties of watermarking. Finally, we would like to note that we view the simplicity of our constructions as an advantage, and a testament to the fact that rethinking and performing small relaxations to the model of watermarking public-key functionalities can allow for quite substantial improvements, both in terms of efficiency and security assumptions, that remain relevant to practice.

Related work and comparison to our model. One of the earliest works related to software watermarking is due to Naccache et al. [22] that considered the problem of "copyrighting" public-key encryption schemes in a setting that is akin to traitor tracing [9]; implementations are fingerprinted and the detection mechanism should be collusion resilient. Note that this type of fingerprinting an object is distinct from the one we consider here. Indeed, watermarking is about establishing the ownership of a certain object whereas fingerprinting is about controlling its distribution. A number of heuristic methods for software watermarking were later presented in [12]. Another related notion is leakage-deterring public key cryptography as defined in [20]. The idea there is that some personal information is embedded to the public key of a user such that, if she decides to share her secret key (or a partial working implementation of her decryption function) the recipient can extract the private information embedded in the public key. This notion is different from watermarking since it focuses on *private* information embedding in a cryptosystem that remains hidden unless the secret key is shared. Privacy is not an issue in watermarking thus construction techniques are technically and conceptually different. Finally, leakage deterring schemes require the embedded information to be of high entropy while in watermarking it is meaningful, depending on the application, to embed arbitrary messages or even not include a message at all.

In [3, 4], Barak et al. study theoretically obfuscation of programs and their main result is an impossibility result showing that virtual black box obfuscation is impossible. They also put forth weaker notions of obfuscation called indistinguishability obfuscation (iO) [3]. This work is also the first that gives a formal definition for software watermarking and explores its relation with iO. The authors provide an impossibility result showing that if a marked circuit has exactly the same functionality as the original one, then under the assumption of indistinguishability obfuscation (iO), watermarking is impossible. Note that the definition of watermarking is not included in original version [3] and is only added in the more recent full version [4].

Nishimaki, [23] (cf. also [24]), inspired by the definitions of watermarking given in [17] (for static objects), suggests a new model for watermarking cryptographic functions modeling both notions unremovability and unforgeability and proposes a watermarking scheme for Lossy Trapdoor functions [26]. The construction is vulnerable, in light of the impossibility result of [4], to an obfuscation attack, i.e., the application of iO to a marked circuit which would effectively remove the mark. It should be noted that [23] circumvents the impossibility result by considering more restricted adversaries whose outputs in the security games should preserve the format of the original functions but, naturally, this leaves open the question of considering general adversaries.

More recently, Cohen et al. [10] motivated by the fact that the iO impossibility result does not hold if a marked circuit is *approximately* close to the original unmarked one (they formulate this as statistical correctness), they propose a watermarking scheme for any puncturable PRF family. This scheme relies on iO, features public key detection and satisfies unremovability without placing any restriction to the adversarial strategy. Based on this scheme and the constructions given by Sahai and Waters [29] for public key encryption and signatures, Cohen et al. [10] describe how to construct "Watermarkable Public-key

[3] A first candidate construction was given in [15].

Encryption" and "Watermarkable Signatures". Both constructions rely on iO. Furthermore, the definitions for these primitives do not consider the notion of unforgeability, however there are some preliminary results related to this notion in [11] (but they are not conclusive).

Boneh et al. [6] provide a watermarking construction for a class of PRFs, called private programmable PRFs, as an application of private constrained PRFs. Their construction achieves unremovability and unforgeability in the private key setting (i.e. private key detection), but relies on iO.

Concurrently to our work, Kim and Wu [21] suggest a watermarking scheme for a family of PRFs based on standard lattice assumptions. In particular, they first introduce a new primitive called private translucent PRFs for which they give a lattice-based construction. Based on that, they provide a construction for a watermarkable family of PRFs that allows private key detection.

Apart from the differences in our definitional models, we also highlight the following differences between [10, 21] and our work. We achieve watermarking of both public key encryption implementations and functionalities instead of only constructing watermarkable instances of public-key cryptographic functionalities (as done by [10]). Our first construction takes advantage of a small shared state while at the same time being very efficient during marking; in fact it is as efficient as the underlying public-key cryptographic implementation and does not require any additional intractability assumptions. Our second construction for watermarking PKE functionalities is the first to achieve both unforgeability and unremovability with public detection which is an open problem in the setting of both [10, 21].

Paper Outline. The rest of the paper is organized as follows. In Section 2 we set notation, overview some basic background and give formal definitions of the notions of cryptographic functionalities and implementations. In Section 3 we present our watermarking security model and the desired security properties. The definitions in Section 3 are focused on watermarking cryptographic functionalities while in the next Section (Sec. 4) we explain how our definitions apply to the setting of watermarking cryptographic implementations. In Section 5 we present our first construction: a watermarking scheme for implementations of public key encryption and prove it secure under the existence of pseudorandom functions. Finally, in Section 6 we present our second construction that is watermarking the public key encryption functionality assuming IBE.

2 Preliminaries

Notation. We first set the notation to be used throughout the paper. By $\lambda \in \mathbb{N}$ we denote the security parameter and by $\mathsf{negl}(\cdot)$ a function negligible in some parameter. The left arrow notation, $x \leftarrow \mathcal{D}$, denotes that x is chosen at random from a distribution $\mathcal{D}$. PPT stands for probabilistic polynomial time. C will always denote an unmarked algorithm/circuit and $\widetilde{C}$ a watermarked one.

Chernoff bounds. Let X be a random variable such that $X = \sum_{i=1}^{n} X_i$, where all X_i are discrete, independent, random variables and $X_i = 0$ with probability p_i and $X_i = 1$

with probability $1-p_i$. Then $\mu = E[X] = \sum_{i=1}^{n} p_i$. Then, it holds that

1. $\mathsf{Pr}[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu}$, for all $0 < \delta < 1$.
2. $\mathsf{Pr}[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}$, for all $0 < \delta < 1$.

Pseudorandom functions [16]. A function $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^\ell$ is a pseudorandom function if the following conditions hold:

1. Given a key $k \in \{0,1\}^k$ and $x \in \{0,1\}^n$, $F(k,x)$ is computed in polynomial time.
2. For any p.p.t. algorithm $\mathcal{A}$, $\left|\mathsf{Pr}_{k\leftarrow\{0,1\}^k}[\mathcal{A}^{F(k,\cdot)} = 1] - \mathsf{Pr}_{f\leftarrow\mathcal{F}}[\mathcal{A}^{f(\cdot)} = 1]\right| \leq \mathsf{negl}(\lambda)$, where $\mathcal{F} = \{f | f : \{0,1\}^n \to \{0,1\}^\ell\}$.

Relations between circuits. In this paragraph we define some notions of "closeness" between circuits which are crucial in defining properties of a watermarking scheme like unforgeability and unremovability as we will see later. These notions are defined with respect to a distribution $\mathcal{D}$ over an input space X.

Definition 2.1 (ρ-closeness)**.** We say that two circuits C_1, C_2 are ρ-close with respect to distribution $\mathcal{D}$ over a space X if they agree on *at least* ρ-fraction of the inputs chosen according to $\mathcal{D}$. Namely,

$$\mathsf{Pr}_{x\leftarrow\mathcal{D}}[C_1(x) = C_2(x)] \geq \rho.$$

We denote ρ-closeness by $C_1 \sim_{\rho,\mathcal{D}} C_2$.

Definition 2.2 (γ-farness)**.** We say that two circuits C_1, C_2 are γ-far with respect to a distribution $\mathcal{D}$ over a space X, if they agree on *at most* $(1-\gamma)$-fraction of the inputs chosen according to $\mathcal{D}$. Namely,

$$\mathsf{Pr}_{x\leftarrow\mathcal{D}}[C_1(x) = C_2(x)] \leq 1-\gamma.$$

We denote γ-farness by $C_1 \nsim_{\gamma,\mathcal{D}} C_2$.

2.1 Defining cryptographic objects

We now define the notions of cryptographic functionalities and implementations. The goal of the cryptographic functionality definition is to capture cryptographic objects (such as: an encryption scheme, a pseudorandom function, etc.) in an abstract ideal way, focusing on the properties it should satisfy (one could think of this as the ideal functionality of a cryptographic scheme). On the other hand, the notion of a cryptographic implementation is used to describe a *specific* implementation of a cryptographic functionality (i.e. the ElGamal encryption scheme [14] is an implementation of the encryption functionality).

Definition 2.3 (Cryptographic functionality)**.** A cryptographic functionality $\mathcal{C}_\mathcal{F}$ consisting of m algorithms[4], $(C_1, \ldots, C_m)$, is defined by a set of n properties and their corresponding probabilities $(G_\mathcal{A}^{\mathsf{prop}_i}, \pi_{\mathsf{prop}_i})_{i=1}^n$. Each property $G_\mathcal{A}^{\mathsf{prop}_i}$ is described in a game fashion: it

[4]We consider protocols to also be described as a set of algorithms.

receives as input m algorithms (that constitute an instance of a candidate implementation of the functionality) and interacts with any PPT adversary $\mathcal{A}$ that attempts to "break" the desired property.

Remark 1. In a more complex definition we could associate each property with a parameter $t_i(\lambda)$ which would define the running time of the adversary. For simplicity, in Definition 2.3, we opt to define all the properties with respect to PPT adversaries. However, some properties may also hold for super-polynomial adversaries (e.g. correctness-related properties).

Example. Consider the public key encryption functionality as an example, which can be defined as a pair of algorithms $\langle \mathsf{Enc}, \mathsf{Dec} \rangle$ that should satisfy the properties of *correctness* and *IND-CPA security*. Correctness can be defined as a security game where an adversary is challenged to provide an encryption of a message M which is decrypted to a message different than M. The IND-CPA security property is defined in the standard way. In Appendix B, we give a concrete definition of both properties for the public key encryption functionality in a game fashion. Corresponding to our definitions, the games will receive as input the encryption/decryption algorithms for a specific key pair. Given that the definition of a cryptographic functionality describes the "ideal" scenario, correctness would always hold with probability 0 (perfect correctness) while in IND-CPA property the adversary would have probability of success of exactly 1/2.

Definition 2.4 (Cryptographic Implementation)**.** Let $\mathcal{C}_{\mathcal{F}}$ be a cryptographic functionality with m algorithms and n properties $(G_{\mathcal{A}}^{\mathsf{prop}_i}, \pi_{\mathsf{prop}_i})_{i=1}^{n}$. An implementation of the cryptographic functionality $\mathcal{C}_{\mathcal{F}}$ consists of an $(m+1)$-tuple of algorithms/protocols $(\mathsf{Gen}, C_1, \ldots, C_m)$ such that, for every security parameter λ and each property prop_i for $i \in \{1, \ldots, n\}$ and for any corresponding PPT adversary $\mathcal{A}$, it holds that :

$$\mathsf{Pr}\left[\begin{array}{l}(k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\lambda) : \\ G_{\mathcal{A}}^{\mathsf{prop}_i}(C_1(k_1, \cdot), \ldots, C_m(k_m, \cdot)) = 1\end{array}\right] \leq \pi_{\mathsf{prop}_i} + \mathsf{negl}(\lambda).$$

In Definition 2.4 we consider *single-instance* properties. This means that the input of the property game is a specific instance of the implementation's algorithms under a fixed key. One could also define *multi-instance* properties, where the corresponding game would receive as inputs multiple versions of the algorithms all under different keys.

3 Watermarking Cryptographic Functionalities

We now define the notion of watermarking cryptographic functionalities. The main idea of our definition follows [4, 10, 6, 21] however notice that: (1) we define watermarking of a *functionality* rather than a circuit class, (2) our marking algorithm is not given a *specific* algorithm/circuit to mark but selects and outputs only an instance of the functionality being marked (i.e. the tuple of the corresponding algorithms), and last (3) our definition allows for a shared public state between the Mark and Detect algorithms. In this section we will refer to the algorithms of a cryptographic functionality as circuits.

3.1 Syntax of a watermarking scheme

Let $\mathcal{C}_{\mathcal{F}}$ be a cryptographic functionality with m algorithms/circuits and n properties $(G_{\mathcal{A}}^{\mathsf{prop_i}}, \pi_{\mathsf{prop_i}})_{i=1}^{n}$ and let $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ denote the message space (of the messages to be embedded on the watermarked scheme), where λ is a security parameter. The entities that are involved in a watermarking scheme are a set of clients, and a "marking service", MarkService.

Definition 3.1 (Watermarking Scheme)**.** A stateful watermarking scheme for a cryptographic functionality $\mathcal{C}_{\mathcal{F}}$, consists of three probabilistic polynomial time algorithms $\langle$WGen, Mark, Detect$\rangle$ whose input/output behavior has as follows:

- WGen : On input 1^λ, it outputs public parameters $param$ and a pair of keys (mk, dk), where mk is the marking key and dk is the detection key. It also initializes a public variable state which can be accessed by all the parties.
- Mark : On input mk, $param$, a message $msg \in \mathcal{M}_\lambda$ (which is sent by a client to the MarkService) and current state, the marking algorithm outputs a tuple of circuits $(\widetilde{C}_1, C_2, \ldots, C_m)$, an efficiently sampleable and representable distribution $\mathcal{D}$ on the inputs of the circuit $\widetilde{C}_1$ [5], and the updated state state$'$.
- Detect : On input dk, $param$, state and a circuit C_1', it outputs a message msg' or unmarked.

Despite the fact that the marking service outputs a tuple of circuits (as many as the algorithms of $\mathcal{C}_{\mathcal{F}}$), only one circuit among them is considered marked. By convention, this would be the first circuit in a tuple produced by the Mark algorithm. It is trivial to extend this definition for the case where more than one circuits are considered marked. The Detect algorithm, as in previous definitions, will run on input any circuit C_1'. A *stateless* watermarking scheme can be described in the same way, by either assuming that the variable state is an empty string or by modifying Definition 3.1 so that state doesn't exist.

Remark 2. Notice that a new feature of our definition of watermarking is that the Mark algorithm outputs a distribution $\mathcal{D}$ on the inputs of marked circuit. This distribution is relevant to our definitions of closeness and farness between circuits (cf. Definitions 2.1, 2.2) and essentially defines on which inputs we expect that circuits are similar or not.

3.2 Security Model

For our security model, we define oracles Challenge, Detect and Corrupt in Figure 1. The Challenge oracle calls the Mark algorithm, and returns to the client a tuple of all output circuits except the one that is considered marked (i.e. the first one) along with an index i that shows how many times the Mark algorithm is invoked so far. The Corrupt oracle outputs the whole tuple of circuits generated by the Mark algorithm for a specific i and works for queries the indices of which were previously returned the Challenge oracle. The Detect oracle runs the Detect algorithm with input a given circuit. Finally, given that state is public, we assume that all oracles have access to it.

[5] The marking algorithm, Mark, can output the distribution $\mathcal{D}$ in the form of an algorithm that samples inputs for the circuit $\widetilde{C}_1$.

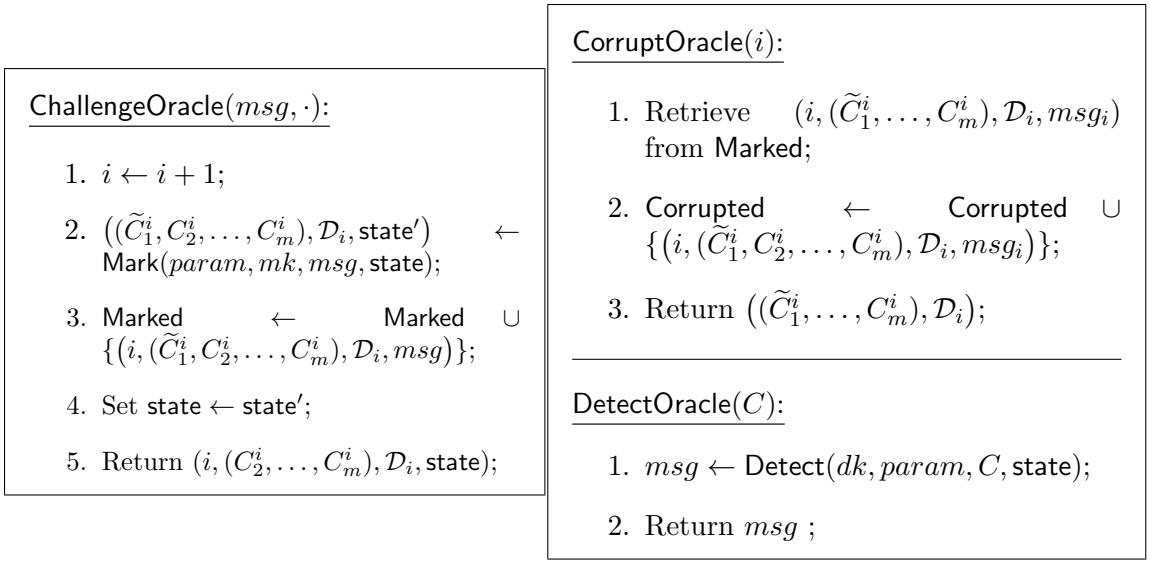

Figure 1: The Challenge, and Detect and Corrupt oracles.

Remark. Notice that for marked (but not corrupted tuples) the adversary does not have access to the marked circuit $\widetilde{C_1}$. This might be restrictive for certain schemes and properties. Consider for instance the case of CCA security for a public key encryption scheme. Then, the marked algorithm would be the decryption one. Although the adversary should not receive Dec_{sk}, he should still be able to query it on ciphertexts of his choice. Thus, we could define one more oracle name QueryOracle that would take as input an index i and an input x and would return the output of the i-th watermarked circuit produced by ChallengeOracle.

Comparing our security model with previous work. In the security model of [10], [21] (note that [21] is specific to PRFs), the adversary has access to both marking and challenge oracles. Their marking oracle receives as input an unmarked circuit and returns the corresponding marked one, while the challenge oracle samples a circuit and returns it marked without revealing the sampled, unmarked one. In the security model of [6], the marking oracle receives a message as input, and returns an unmarked PRF key and a marked circuit embedded with this message. Note that [6, 21] give only definitions for PRFs and not circuit classes in general. Although the security model of [6] seems closer to our model, the existence of a marking oracle, as this is defined in [10, 21] and [6], does not comply with our model. The Mark algorithm in our case neither takes as input an unmarked circuit nor returns an unmarked circuit together with the marked one as output. Another difference with the model of [6, 10, 21] is that our challenge oracle does not return the marked circuit of the tuple, i.e. the first one by convention. The corrupt oracle is the one that returns the marked cicruit for a previously sampled marked instance of a functionality or implementation. Notice that our challenge oracle does not play any important role for functionalities with a single algorithm like a PRF but it is crucial for multi-algorithm functionalities. For example, in the public-key encryption functionality if the decryption function, i.e. a secret key, is the one which is marked, it is reasonable that the adversary should be given the corresponding public key.

3.3 Security Properties

Next, we define the properties that should be satisfied by a watermarking scheme.

We start by detection correctness which informally states that a valid watermarked circuit should be detected as such with a non-negligible probability. Our definition guarantees that any update on the state, after each execution of Mark, does not affect the detection correctness of previously marked circuits.

Definition 3.2 (Detection Correctness)**.** We say that a watermarking scheme satisfies detection correctness if for any PPT advrersary $\mathcal{A}$ against the security game described in Figure 2, it holds that:

$$\mathsf{Pr}[G^{\mathsf{det-corr}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

The next property we define is ρ-unremovability. Informally, an adversary after querying the Challenge and Corrupt oracles, should not be able to output a circuit that is ρ-close to any of the queried ones, and at the same time is unmarked or is marked under a different (than the original) mark. In Figure 3 we first describe the unremovability security game and then we provide the definition below.

Definition 3.3 (ρ-Unremovability)**.** We say that a watermarking scheme satisfies the ρ-unremovability property if for any PPT adversary $\mathcal{A}$ against the security game described in Figure 3, it holds that

$$\mathsf{Pr}[G^{\mathsf{unrmv}}_{\mathcal{A}}(1^\lambda, \rho) = 1] \leq \mathsf{negl}(\lambda).$$

We then define γ-unforgeability which informally states that an adversary, after receiving marked circuits through oracle queries, should not be able to output a marked circuit that is γ-far from the received, marked ones. Note that $\mathcal{A}$ only receives marked circuits through the Corrupt oracle, thus if he manages to forge a circuit that is close to a marked (but not corrupted one) he should still win the game. The unforgeability security game is described in Figure 4.

$\underline{G^{\mathsf{det-corr}}_{\mathcal{A}}(1^\lambda)}$:

1. The Challenger runs $\mathsf{WGen}(1^\lambda)$ which outputs $(param, (mk, dk), \mathsf{state})$. It gives $param$ to the adversary $\mathcal{A}$. $\mathcal{A}$ has also access to the public variable state. If detection is public, $\mathcal{A}$ also receives dk from the Challenger. The Challenger initializes the sets Marked and $\mathsf{Corrupted}$ as empty and $i \leftarrow 0$.
2. $\mathcal{A}$ makes queries to $\mathsf{DetectOracle}$, $\mathsf{ChallengeOracle}$ and $\mathsf{CorruptOracle}$.
3. $\mathcal{A}$ outputs an index j.
4. Output 1 iff $\left(j, (\widetilde{C}^j_1, \cdot), \mathcal{D}_j, msg\right) \in \mathsf{Marked}$ and $\mathsf{Detect}(dk, param, \widetilde{C}^j_1, \mathsf{state}) \neq msg$.

Figure 2: The Detection-Correctness game

Definition 3.4 (γ-Unforgeability)**.** We say that a watermarking scheme satisfies γ-unforgeability if for any PPT adversary $\mathcal{A}$ against the security game defined in Figure 4 it holds that

$$\mathsf{Pr}[G^{\mathsf{unforge}}_{\mathcal{A}}(1^\lambda, \gamma) = 1] \leq \mathsf{negl}(\lambda).$$

$G_{\mathcal{A}}^{\mathsf{unrmv}}(1^\lambda, \rho)$:

1. The Challenger runs $\mathsf{WGen}(1^\lambda)$ which outputs $(param, (mk, dk), \mathsf{state})$. It gives $param$ to the adversary $\mathcal{A}$. $\mathcal{A}$ has also access to the public variable state. If detection is public, $\mathcal{A}$ also receives dk from the Challenger. The Challenger initializes the sets Marked and $\mathsf{Corrupted}$ as empty and $i \leftarrow 0$.
2. $\mathcal{A}$ makes queries to $\mathsf{DetectOracle}$, $\mathsf{ChallengeOracle}$ and $\mathsf{CorruptOracle}$.
3. $\mathcal{A}$ outputs a circuit C^*.
4. The game outputs 1 iff there exists $\left(j, (\widetilde{C}_1^j, \cdot), \mathcal{D}_j, msg\right) \in \mathsf{Marked}$ such that $C^* \sim_{\rho, \mathcal{D}_j} \widetilde{C}_1^j$ **and** $\mathsf{Detect}(dk, param, C^*, \mathsf{state}) \neq msg$.

Figure 3: The ρ-Unremovability game

$G_{\mathcal{A}}^{\mathsf{unforge}}(1^\lambda, \gamma)$:

1. The Challenger runs $\mathsf{WGen}(1^\lambda)$ which outputs $(param, (mk, dk), \mathsf{state})$. It gives $param$ to the adversary $\mathcal{A}$. $\mathcal{A}$ has also access to the public variable state. If detection is public, $\mathcal{A}$ also receives dk from the Challenger. The Challenger initializes the sets Marked and $\mathsf{Corrupted}$ as empty and $i \leftarrow 0$.
2. $\mathcal{A}$ makes queries to $\mathsf{DetectOracle}$, $\mathsf{ChallengeOracle}$ and $\mathsf{CorruptOracle}$.
3. $\mathcal{A}$ outputs a circuit C^*.
4. The game outputs 1 iff
   (a) For all $\widetilde{C}_1^j$ such that $\left(j, (\widetilde{C}_1^j, \cdot), \mathcal{D}_j, msg\right) \in \mathsf{Corrupted}$ it holds that $C^* \approx_{\gamma, \mathcal{D}_j} \widetilde{C}_1^j$.
   (b) $\mathsf{Detect}\big(dk, param, C^*, \mathsf{state}\big) \neq \mathsf{unmarked}$.

Figure 4: The γ-Unforgeability game

Finally, we define the functionality property-preserving notion. Informally, this notion captures the requirement that a watermarked cryptographic functionality $\mathcal{C}_{\mathcal{F}}$ should preserve the properties of the original (non-marked) functionality. In other words, the probability that an adversary $\mathcal{A}$ breaks a property prop_i of a watermarked functionality should be less or equal to the probability that an adversary breaks the same property for the non-watermarked functionality (plus a negligible factor). We define functionality property-preserving with the aid of the game in Figure 5. In that game, the adversary $\mathcal{A}$ decides the instance of the algorithms for which they will play the security property game $G_{\mathcal{A}}^{\mathsf{prop}_j}$, choosing among the watermarked ones he received by the $\mathsf{ChallengeOracle}$. Note that $\mathcal{A}$ cannot pick an instance that has previously corrupted. If the selected instance was a corrupted one, then the security property prop_j could have been trivially broken by $\mathcal{A}$.

Definition 3.5 (Functionality Property-preserving)**.** A watermarking scheme is property-preserving for a cryptographic functionality $\mathcal{C}_{\mathcal{F}}$ with m algorithms and n properties ($G_{\mathcal{A}}^{\mathsf{prop_j}}$,

$\underline{G_{\mathcal{A}}^{\mathsf{wm-prop}_j}(1^\lambda):}$

1. The Challenger runs $\mathsf{WGen}(1^\lambda)$ which outputs $(param, (mk, dk), \mathsf{state})$. It gives $param$ to the adversary $\mathcal{A}$. $\mathcal{A}$ has also access to the public variable state. If detection is public, $\mathcal{A}$ also receives dk from the Challenger. The Challenger initializes the sets Marked and $\mathsf{Corrupted}$ as empty and $i \leftarrow 0$.
2. $\mathcal{A}$ can make queries to $\mathsf{DetectOracle}$, the $\mathsf{ChallengeOracle}$ and the $\mathsf{CorruptOracle}$.
3. $\mathcal{A}$ chooses i such that $(i, (\widetilde{C}_1^i, C_2^i, \ldots, C_m^i), \mathcal{D}_i, msg) \in \mathsf{Marked} \setminus \mathsf{Corrupted}$ and sends i to the Challenger.
4. Then, the Challenger runs the game $G_{\mathcal{A}}^{\mathsf{prop}_j}$ with $\mathcal{A}$ but on input $(\widetilde{C}_1^i, C_2^i, \ldots, C_m^i)$ (notice that only challenger knows $\widetilde{C}_1^i$).
5. The game $G_{\mathcal{A}}^{\mathsf{wm-prop}_j}(1^\lambda)$ outputs whatever $G_{\mathcal{A}}^{\mathsf{prop}_j}$ outputs.

Figure 5: The Functionality property-preserving game for a property prop_j.

$\pi_{\mathsf{prop_j}})_{j=1}^n$ if for any PPT adversary $\mathcal{A}$ against the security game defined in Figure 5, and for any property $\mathsf{prop_j}$, it holds that

$$\mathsf{Pr}[G_{\mathcal{A}}^{\mathsf{wm-prop}_j}(1^\lambda) = 1] \leq \pi_{\mathsf{prop}_j} + \mathsf{negl}(\lambda).$$

Note. There may be property games where the adversary is not given all the circuits $C_2^i, \ldots, C_m^i$ but only a subset of them. We could give an alternative definition capturing such cases, however we omit it for simplicity reasons. We also described property-preserving for the scenario when $\mathcal{A}$ is not given the marking key mk. One could also consider an alternative, stronger definition, where $\mathcal{A}$ has mk, marks objects by himself and then for a state of his choice, runs the security game for the particular property using the algorithms returned by Mark in the chosen state.

Note (Functionality-preserving). The notion of functionality-preserving was defined in the literature but with a different meaning of what functionality means. In particular in [23] (revised eprint version [24]) it was used to capture that for any input x the outputs of the unmarked and the corresponding marked circuits should remain the same (i.e. $C(x) = \tilde{C}(x)$). The notion was also used in a similar way in [11]. This notion is implied by our property-preserving property which more generically captures the fact that the *properties* of the watermarked scheme should be preserved, in the sense that it should be almost equally (but a negligible factor) difficult for an adversary to break them.

Note (Meaningfulness). In the literature, the *meaningfulness* of a watermarking scheme (i.e., the vast majority of circuits is unmarked) is also considered as a required security property. As observed in [11] though, it is captured by unforgeability: if an adversary can sample a circuit that is marked with good probability, then it can directly forge a circuit without making any oracle queries.

$\underline{G_{\mathcal{A}}^{\mathsf{mi-prop}_j}(1^\lambda):}$

1. Set $i \leftarrow 0$.
2. $\mathcal{A}$ can make queries to $\mathsf{MultiInstanceOracle}$ and the $\mathsf{CorruptOracle}$.
3. $\mathcal{A}$ chooses i such that $(i, (C_1^i, C_2^i, \ldots, C_m^i), \mathcal{D}_i, msg) \in \mathsf{Instances} \setminus \mathsf{Corrupted}$ and sends i to the Challenger.
4. Then, $\mathcal{A}$ runs with the Challenger the game $G_{\mathcal{A}}^{\mathsf{prop}_j}$ but on input $(C_1^i, C_2^i, \ldots, C_m^i)$.
5. The game $G_{\mathcal{A}}^{\mathsf{mi-prop}_j}(1^\lambda)$ outputs whatever $G_{\mathcal{A}}^{\mathsf{prop}_j}$ outputs.

Figure 6: The multi-instance security game for a property prop_j.

4 Watermarking Cryptographic Implementations

Let $(\mathsf{Gen}, C_1, \ldots, C_m)$ be an implementation of a cryptographic functionality $\mathcal{C}_{\mathcal{F}}$. The syntax of a watermarking scheme for cryptographic implementations is exactly the same with the syntax for cryptographic functionalities. The reason is that in practice the Mark algorithm of the watermarking scheme acts as (replaces in a sense) the Gen algorithm of a cryptographic implementation and outputs an instance of the implementation algorithms under a specific key. What differentiates these two definitions is only the property-preserving notion. The rest of the security properties (detection correctness, ρ-unremovability, γ-unforgeability) remain the same as in Section 3.

In order for a watermarked implementation to be property-preserving it needs to hold that the watermarked implementation preserves the properties of the non-watermaked one, which in turn preserves the properties of the corresponding cryptographic functionality it implements. Notice that, when we watermark a cryptographic implementation we naturally want to achieve multi-instance security for the properties of the implementation (multi-instance versions of security definitions are encountered in the literature for various types of cryptographic functionalities, i.e. [18]). This arises by the fact that the $\mathsf{ChallengeOracle}$ is called multiple times by the adversary, who thus receives multiple instances of implementations and then chooses for which one he will attempt to break the property of the implementation. Therefore we first define the multi-instance version of the security game for a property prop_i in Figure 6. The $\mathsf{MultiInstanceOracle}$ called in the game is identical to the $\mathsf{ChallengeOracle}$ but instead of calling the Mark algorithm it calls the key generation algorithm Gen of the implementation and stores all the created instances of generated algorithms to a set $\mathsf{Instances}$. The security game $G_S^{\mathsf{prop}_j}$ is defined as in the previous definition.

Definition 4.1 (Implementation Property-preserving)**.** We say that a watermarking scheme satisfies implementation property-preserving with error ε for a cryptographic implementation $(\mathsf{Gen}, C_1, \ldots, C_m)$ if for any p.p.t. adversary $\mathcal{S}$ there is a PPT adversary $\mathcal{A}$ such that

$$\left|\mathsf{Pr}[G_{\mathcal{S}}^{\mathsf{wm-prop}_j}(1^\lambda) = 1] - \mathsf{Pr}[G_{\mathcal{A}}^{\mathsf{mi-prop}_j}(1^\lambda) = 1]\right| \leq \varepsilon.$$

Proposition 4.1. *If a watermarking scheme is implementation property-preserving, it is*

also functionality property-preserving, i.e. Definition 4.1 implies Definition 3.5, when ε is negligible to the security parameter.

5 A watermarking scheme for implementations of PKE

We describe a construction of an efficient watermarking scheme for a cryptographic implementation of a public key encryption scheme. One could view our construction as a compiler that takes as input an existing public key encryption scheme and converts it into a watermarked public key encryption scheme.

Public key detection via linear size state vs secret-key detection via logarithmic size state. Given that our definition of a watermarking scheme (Def. 3.1) allows for a public state one could design a watermarking scheme for an implementation of a public key encryption scheme by assuming a state with size linear to the number of markings. Specifically, assume that the shared state is represented as a public table which can be accessed by both the Marking Service and any party that runs Detect algorithm. For any marking request, Mark generates a fresh pair of keys (pk, sk) using the key generation algorithm of the public key encryption scheme that is being watermarked. Then, it stores the generated public key pk to the state table and outputs $(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$. Thus, state will hold all the public keys generated by Mark so far. Now, how does Detect work given the public state? When Detect receives as input a (decryption) algorithm/circuit C, it will check for any public key stored in the public table state, whether the circuit can decrypt correctly a number of ciphertexts which is above a certain threshold.

Such a construction could be proven to be a secure watermarking scheme for public key encryption however the use of a state that grows linearly to the number of markings is not very appealing in practice especially for implementations where the public keys are large. We overcome this problem by focusing on private detection watermarking. In Figure 7, we suggest a watermarking scheme with logarithmic state and private key detection where the same key is being used for both marking and detection.

Overview of our construction. Our proposed construction is given in Figure 7 and assumes a state of logarithmic size (in the security parameter). We use a PRF function F with a random key K and set marking and detection keys equal to K and state to be a counter of the number of markings so far. Whenever, Mark is run it will compute (pk, sk) by running $F(K, \mathsf{state}+1)$, set $\mathsf{state} = \mathsf{state}+1$ and output $\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk}$. In order for the detection algorithm to correctly identify whether a decryption circuit C is marked or not, it will first re-generate all possible key pairs by running $F(K, i)$ for every $i \leq \mathsf{state}$. Then, for each produced pk_i it will check whether an encryption of a random plaintext under it, can be correctly decrypted with C. As it turns out by our security analysis it is not enough to check for a single plaintext, in fact, it will check the decryptions of λ/ρ randomly selected plaintexts.

Notice that in the above watermarking scheme of Figure 7 if the number of circuits to be marked was upper bounded by N (where N is polynomial to the security parameter), then our construction works without state. The only difference is in the Detect algorithm:

Detect on input a circuit C will start testing all possible pairs (pk_i, sk_i) for $i \in [1, \ldots, N]$. If for a key pair it detects marked then it stops. In the worst case (where C is not marked) Detect will have to test all N possible keys (which is still polynomial running time).

A note about state. Note that the state information in our construction is public and it should be immutable for the system to work in practice. A potential solution for storing the state would be by using a public bulletin board or a blockchain system. For example, every time the state is updated, the marking service signs it and posts a new transaction in the blockchain with the new state and the signature. Even though storing information in the blockchain is an expensive operation, our scheme, with its logarithmic size state, is suitable for a blockchain deployment. We leave a detailed analysis under a formal blockchain security model for future work.

Security analysis of our construction. In our analysis we consider key-generation algorithms which create their random tape by choosing keys uniformly at random. This aligns with the key generation algorithms of all the well-known encryption schemes. We provide below the security theorem for our construction.

Theorem 5.1. *Let* $\langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$ *be an implementation of the Public Key Encryption functionality that has plaintext space of exponential size (in the security parameter) and satisfies (multi-instance) perfect correctness*[6] *and (multi-instance) IND-CPA security. Let* $F : \mathcal{K} \times \{0,1\}^n \leftarrow \{0,1\}^\ell$ *be a pseudorandom function, where* $\mathcal{K}$ *is the key space. Then, the scheme in Figure 7 is a watermarking scheme for the implementation* $\langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$*. Namely, it satisfies Detection Correctness, Implementation property-preserving with error* ε_{prf}*,* ρ*-Unremovability and* $(1 - \rho/3)$*-Unforgeability, where* ε_{prf} *is the security of the PRF and* ρ *is a parameter with* $\rho \geq \frac{1}{\mathsf{poly}(\lambda)}$*.*

We prove Theorem 5.1 by proving separately each property of a watermarking scheme in a sequence of Lemmas, i.e. Lemmas 1, 2, 3, 4.

Lemma 1 (Detection Correctness)**.** *The scheme in Figure 7 satisfies Detection Correctness as this is defined in Definition 3.2.*

Proof. Proving Detection Correctness is trivial since Detect algorithm in Figure 7 reconstructs exactly the same circuits already produced by Mark. Namely, for any j chosen by an adversary at step 3 of the game of Figure 2, in our construction we have that $j \leq \mathsf{state}$. Therefore, Detect algorithm will just reconstruct $\widetilde{C}_j$ at the j-th iteration and will decrypt correctly all λ/ρ ciphertexts. □

Lemma 2 (Implementation property-preserving)**.** *If the public key encryption scheme* $\langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$ *satisfies (multi-instance) IND-CPA security, perfect correctness, and* $F : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^\ell$ *is a pseudorandom function, then the watermarking scheme of Figure 7 satisfies Definition 4.1.*

[6] Our proofs could also be extended for implementations which have a negligible decryption error.

- WGen: On input 1^λ, it chooses uniformly at random a key K for a pseudorandom function $F : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^\ell$. It outputs $mk = dk = K$ and initializes the public variable $\mathsf{state} \leftarrow 0$.
- Mark: On input K, state, marked, compute $i = \mathsf{state}+1$ and run $\mathsf{Gen}(1^\lambda)$ with randomness $F(K, i)$. The output is a public-secret key pair (pk_i, sk_i) and the algorithm returns a pair of circuits $(\mathsf{Enc}_{pk_i}, \mathsf{Dec}_{sk_i})$. Set as $\mathcal{D}_i$ the distribution of the ciphertexts that correspond to plaintexts chosen uniformly from the plaintext space. Then, set $\mathsf{state} \leftarrow \mathsf{state} + 1$.
- Detect: On input K, a circuit C and state, for $i = 1$ to state:
  - Run $\mathsf{Gen}(1^\lambda)$ with randomness $F(K, i)$ (as the Mark algorithm does) in order to obtain (pk_i, sk_i).
  - Choose $k = \lambda/\rho$ plaintexts uniformly at random and encrypt them under pk_i, i.e. compute the ciphertexts $c_1, \ldots, c_k$.
  - For $j = 1$ to k check whether $C(c_j) = m_j$. If this is true for at least $\lambda/2$ ciphertexts, return marked.

  Otherwise, return $\mathsf{unmarked}$.

Figure 7: Watermarked Public Key Encryption Implementation

Proof. We prove this lemma in the following two claims.
Claim 2.1:(IND-CPA security) Assuming that F is a pseudorandom function then, for any PPT adversary $\mathcal{S}$, there is a PPT adversary $\mathcal{A}$ such that

$$\left|\mathsf{Pr}[G_{\mathcal{S}}^{\mathsf{wm-IND-CPA}}(1^\lambda) = 1] - \mathsf{Pr}[G_{\mathcal{A}}^{\mathsf{mi-IND-CPA}}(1^\lambda) = 1]\right| \leq \varepsilon_{prf},$$

where ε_{prf} is the security of the PRF.

Sketch. This claim is proven by using a sequence of games. We first define as $G_0 = G_{\mathcal{S}}^{\mathsf{wm-IND-CPA}}(1^\lambda)$ (for a fixed $\mathcal{S}$). Then we define G_1 by substituting any call to the PRF function F for a key K to generate tuples $(\mathsf{Dec}_{sk_i}, \mathsf{Enc}_{pk_i})$ by a call to a random function $f : \{0,1\}^n \to \{0,1\}^\ell$. Observe that G_1 matches the multi-instance IND-CPA security game and therefore the claim holds due to the security of the PRF.
Claim 2.2:(Correctness) For any PPT adversary $\mathcal{S}$, there is a PPT adversary $\mathcal{A}$ such that $\left|\mathsf{Pr}[G_{\mathcal{S}}^{\mathsf{wm-COR}}(1^\lambda) = 1] - \mathsf{Pr}[G_{\mathcal{A}}^{\mathsf{mi-COR}}(1^\lambda) = 1]\right| = 0$, under the assumption that the encryption scheme $\langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$ is perfectly correct.

Sketch. By assumption, we have that $\mathsf{Pr}[G_{\mathcal{A}}^{\mathsf{mi-COR}}(1^\lambda) = 1] = 0$. We have to show that $\mathsf{Pr}[G_{\mathcal{S}}^{\mathsf{wm-COR}}(1^\lambda) = 1] = 0$. This means that for any pair of public-secret keys generated by Gen, the adversary wins the multi-instance game with probability 0 by a counting argument. This argument holds if we replace Gen with another algorithm Gen' with a different sampling distribution but under the restriction the support set of Gen' is a subset of the support set of Gen. This holds for the support set of the PRF.

□

Lemma 3 (ρ-Unremovability)**.** *The scheme in Figure 7 satisfies ρ-Unremovability according to Definition 3.3 under the assumption that the underlying public-key encryption scheme*

$\langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$ *satisfies perfect correctness. (without introducing any negligible decryption error.)*

Proof. We prove this lemma using counting arguments which are independent of the adversary's strategy. In particular, we fix $(i, (\mathsf{Dec}_{sk_i}, \mathsf{Enc}_{pk_i}), \mathcal{D}_i) \in \mathsf{Marked}$ and we assume that C^* is ρ-close to the function Dec_{sk_i} with respect to $\mathcal{D}_i$. Then, we prove that Detect returns $\mathsf{unmarked}$ only with negligible probability.

We define the random variable $X_{i,j}$ as follows:

$$X_{i,j} = \begin{cases} 1, & \text{if } \ C^*(c_j) = \mathsf{Dec}_{sk_i}(c_j) \text{ where } c_j \leftarrow \mathcal{D}_i \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

We have that $\mathsf{Pr}[X_{i,j} = 1] \geq \rho$. Then we define the random variable $X_i = \sum_{j=1}^{\lambda/\rho} X_{i,j}$, where the random variables $X_{i,j}$ for $j = 1, \ldots, \lambda/\rho$ are independent. Thus, we have that $\mu = E[X_i] = \sum_{i=1}^{\lambda/\rho} E[X_{i,j}] \geq \sum_{i=1}^{\lambda/\rho} \rho = \lambda$.

By Claim 2.2, we have that the watermarking scheme preserves perfect correctness of the encryption scheme $\langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$ (without introducing any decryption error). This means that the random variable X_i counts the number of ciphertexts from $\mathcal{D}_i$ (out of $k = \lambda/\rho$) which are correctly decrypted by C^* under sk_i. We will compute an upper bound on $\mathsf{Pr}[X_i < \lambda/2]$ using the following Chernoff bound:

$$\mathsf{Pr}[X_i \leq (1-\delta)\mu] \leq e^{-\mu \cdot \frac{\delta^2}{2}}, \text{ for all } 0 < \delta < 1. \tag{2}$$

First, we have that $\mathsf{Pr}[X < \lambda/2] \leq \mathsf{Pr}[X_i \leq \lambda/2] \leq \mathsf{Pr}[X_i \leq \mu/2]$. If we set $\delta = 1/2$ in (2) we have that $\mathsf{Pr}[X_i \leq \mu/2] \leq e^{-\mu/8} \leq e^{-\lambda/8}$, which is negligible in λ. Consequently, $\mathsf{Pr}[X_i \geq \lambda/2] \geq 1 - e^{-\lambda/8}$ which means that Detect will return marked with probability $1 - \mathsf{negl}(\lambda)$.

□

Lemma 4 ($(1-\rho/3)$-Unforgeability)**.** *If the function $F : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^\ell$ satisfies PRF security and the public key encryption scheme $\langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$ satisfies IND-CPA security and perfect correctness, then the scheme in Figure 7 satisfies $(1-\rho/3)$-unforgeability.*

Proof. Assuming that there is a PPT adversary $\mathcal{A}$ that breaks $(1-\rho/3)$-unforgeability property of our watermarking scheme with non-negligible probability α, we will construct a PPT adversary $\mathcal{B}$ which breaks IND-CPA security of the encryption scheme $\langle \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec} \rangle$. At first, observe that by the security of the PRF function F, $\mathcal{A}$ breaks $(1-\rho/3)$-unforgeability property with non-negligible probability for a modified watermarking scheme, where any call to the pseudorandom function $F(K, i)$ (for a random key K), is substituted by a uniformly chosen string in $\{0,1\}^\ell$ (i.e. the range of the PRF). Therefore, $\mathcal{B}$ will simulate a $(1-\rho/3)$-unforgeability experiment with $\mathcal{A}$ by choosing uniformly at random a string from $\{0,1\}^\ell$, whenever $\mathsf{Gen}(1^\lambda)$ is invoked for some index $i \in \{0,1\}^n$. This string will remain the same in any invocation of the algorithm for the same index i throughout the simulation.

In this proof, we will use the notion of IND-CPA security for multiple messages as this is defined in Appendix B.1. Specifically, the challenge will be two tuples of length λ/ρ (which is polynomial in λ). Without loss of generality, we assume that $\mathcal{A}$ makes m_1 queries to the

Description of $\mathcal{B}$:

1. $\mathcal{B}$ receives pk from the Challenger of the IND-CPA security game.

2. $\mathcal{B}$ chooses $i^* \in \{1, \ldots, m_1\}$ uniformly at random and initializes $\mathsf{state} \leftarrow 0$, $i \leftarrow 0$ and $\mathsf{Marked} \leftarrow \emptyset$. From now on, we will refer to pk as pk_{i^*}.

3. $\mathcal{B}$ gives $param$ to $\mathcal{A}$ and simulates the oracle queries as follows:

   ChallengeOracle queries: When $\mathcal{A}$ makes a query to the $\mathsf{ChallengeOracle}$, $\mathcal{B}$ sets $i \leftarrow i + 1$ and $\mathsf{state} \leftarrow \mathsf{state} + 1$. If $i \neq i^*$, $\mathcal{B}$ chooses uniformly a string r_i from $\mathcal{S}$, runs $\mathsf{Gen}(1^\lambda)$ with randomness r_i and generates a pair (pk_i, sk_i). Then, $\mathcal{B}$ stores $\left(i, (\mathsf{Dec}_{sk_i}, \mathsf{Enc}_{pk_i}), \mathcal{D}_i\right)$ to the table Marked and sends $\left(i, \mathsf{Enc}_{pk_i}, \mathcal{D}_i\right)$ to $\mathcal{A}$. If $i = i^*$, then $\mathcal{B}$ returns $(i^*, \mathsf{Enc}_{pk_{i^*}}, \mathcal{D}_{i^*})$ to $\mathcal{A}$.

   CorruptOracle queries: When $\mathcal{A}$ makes a query $i \leq m_1$ to the Corrupt Oracle, $\mathcal{B}$ checks if $i = i^*$. If this holds, $\mathcal{B}$ outputs a random bit $b \in \{0, 1\}$ and the game stops. This happens because $\mathcal{B}$ cannot simulate correctly this query since it does not hold the secret key that corresponds to the public key pk_{i^*}. If $i \neq i^*$, $\mathcal{B}$ sends $\left(i, (\mathsf{Dec}_{sk_i}, \mathsf{Enc}_{pk_i}), \mathcal{D}_i\right)$ to $\mathcal{A}$.

   DetectOracle queries: When $\mathcal{A}$ makes a query C, $\mathcal{B}$ runs the algorithm Detect by utilizing the encryption functions stored at the table Marked.

4. $\mathcal{A}$ outputs C^*.

5. $\mathcal{B}$ chooses two tuples of messages $(M_{0,1}, \ldots, M_{0,\lambda/\rho})$, $(M_{1,1}, \ldots, M_{1,\lambda/\rho})$ and sends them to the Challenger. Each message of the tuple is chosen by $\mathcal{B}$ uniformly at random from the message space.

6. Upon receiving $(c_{b,1}, \ldots, c_{b,\lambda/\rho})$ from the Challenger, where $c_{b,j} = \mathsf{Enc}(pk, M_{b,j})$, $\mathcal{B}$ initializes three counters $\mathsf{count}_0, \mathsf{count}_1, \mathsf{count}_\perp$ to 0. Then $\mathcal{B}$ runs Detect on input C^* in a way that at round i^*, it uses the ciphertexts $(c_{b,1}, \ldots, c_{b,\lambda/\rho})$ given by the Challenger. Then, for $j = 1, \ldots, \lambda/\rho$, $\mathcal{B}$ does the following:

   - If $C^*(c_{b,j}) = M_{0,j}$, it sets $\mathsf{count}_0 \leftarrow \mathsf{count}_0 + 1$
   - If $C^*(c_{b,j}) = M_{1,j}$, it sets $\mathsf{count}_1 \leftarrow \mathsf{count}_1 + 1$
   - Otherwise it sets $\mathsf{count}_\perp \leftarrow \mathsf{count}_\perp + 1$

   If $\mathsf{count}_0 \geq \lambda/2$ or $\mathsf{count}_1 \geq \lambda/2$, then $\mathcal{B}$ outputs 0 or 1 respectively. Otherwise, it outputs a random bit.

Figure 8: Unforgeability to IND-CPA reduction

$\mathsf{ChallengeOracle}$, m_2 queries to the $\mathsf{CorruptOracle}$ and m_3 queries to the $\mathsf{DetectOracle}$. The reduction is described in Figure 8. Then, we provide a description of the analysis of our reduction.

Remark 3. Notice that in step 5 of the reduction of Figure 8 there is a probability for messages in the same positions of the two tuples to be equal, i.e. $M_{0,i} = M_{1,i}$. In that case the reduction would fail. However, the probability of this event is very small since the plaintext space has exponential size. For simplicity, we ignore this event in the probability analysis below assuming that all probabilities are also conditioned on the event that the reduction does not fail.

Outline of the analysis. In order to compute the probability that $\mathcal{B}$ wins, we have to compute the probabilities $\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins]$ and $\mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins]$. We start by the case where $\mathcal{A}$ **wins**. If $\mathcal{A}$ wins, this means that there is j^* such that C^* is $(1-\rho/3)$-far from Dec_{sk_j} but Detect returns marked. First, using counting arguments, we essentially rule out all the decryption functions that belong to the $\mathsf{Corrupted}$ set (i.e. returned by $\mathsf{CorruptOracle}$ upon request). Then, given that j^* does not belong the $\mathsf{Corrupted}$ set, we distinguish two different cases depending on whether $\mathcal{B}$ guesses j^* correctly, or not. $\mathcal{B}$ guesses correctly j^* with probability $1/m$, since the best it can do is choosing in the beginning uniformly at random $i^* \in \{1,\ldots,m\}$ and plug into the i^*-th $\mathsf{ChallengeOracle}$ query for this index the public key given by the IND-CPA Challenger. If $\mathcal{B}$ guesses correctly j^* (i.e. $j^* = i^*$), since pk_{j^*} is the key given to $\mathcal{B}$ by the IND-CPA Challenger, $\mathcal{B}$ will win as well. If $\mathcal{B}$ does not guess correctly j^*, this means that $\mathcal{A}$ produces a "forgery" but for a decryption function which is already known to $\mathcal{B}$. The question is whether in such case $\mathcal{B}$ could benefit in guessing the correct bit at the end or not. Therefore, we should analyze how C^* behaves on input the tuple of ciphertexts $(c_{b,1},\ldots,c_{b,\lambda/\rho})$ returned by the Challenger.

There are three cases. First, if C^* decrypts at least $\lambda/\rho - \lambda/2$ ciphertexts of the tuple (which are encrypted under pk_{j^*}) to messages completely irrelevant to the corresponding plaintexts of both tuples challenged in step 5 (i.e. $C^*(c_{b,i}) \neq M_{0,i}$ and $C^*(c_{b,i}) \neq M_{1,i}$), then $\mathcal{B}$ outputs a random bit and wins with probability $1/2$. In the second case, if C^* decrypts correctly at least $\lambda/2$ ciphertexts of the tuple $(c_{b,1},\ldots,c_{b,\lambda/\rho})$, then $\mathcal{B}$ guesses the correct bit and wins. In other words, in this case C^* decrypts correctly a portion of ciphertexts encrypted under two different keys. Although it would be interesting to explore under which conditions this may happen or not (and with what probability), it does not affect our probability analysis as we can just set the probability that $\mathcal{B}$ guesses the correct bit in this case to be at least $1/2$. The last case is having C^* decrypt at least $\lambda/2$ out of λ/ρ ciphertexts in corresponding plaintexts of the opposite tuple (the one **not** selected by the Challenger). To make this more clear, assume that the Challenger chooses $b = 0$, and therefore encrypts the plaintexts $(M_{0,1},\ldots,M_{\lambda/\rho})$ under pk_{j^*}. In this case, C^* decrypts $c_{0,j}$ to $M_{1,j}$ (for at least $\lambda/2$ values of j). This scenario may happen only with negligible probability since the plaintexts in the two tuples are chosen uniformly at random from a plaintext space of exponential size. To sum up, for the case where $\mathcal{B}$ does not guess correctly j^*, $\mathcal{B}$ wins only with probability at least $1/2$.

At this point we move to the case where $\mathcal{A}$ **does not win** $(1-\rho/3)$-unforgeability game. One possible case is that $\mathcal{A}$ makes a $\mathsf{CorruptOracle}$ query for the index i^* and since $\mathcal{B}$ cannot provide the corresponding secret key, it aborts and outputs a random bit. Supposing that $\mathcal{B}$ does not just abort, two possible scenarios can take place if we consider the $(1-\rho/3)$-unforgeability definition: (1) Either Detect returns unmarked **or** (2) there is a decryption function Dec_{sk_i} such that C^* is *not* $(1-\rho/3)$-far from Dec_{sk_i}. In the former case $\mathcal{B}$ will

- $\mathsf{WGen}(1^\lambda)$: Run $\mathsf{IBE.Setup}(1^\lambda)$ which outputs $(msk, \mathsf{IBE.param})$. Set $mk = msk$, $param = \mathsf{IBE.param}$, $dk = \mathsf{IBE.param}$, and initialize $\mathsf{state} \leftarrow 0$.
- Mark: On input $mk, param$, compute $i = \mathsf{state} + 1$ and set $id_i = i$. Run $\mathsf{IBE.Extract}(msk, f(param, id_i))$ which outputs a secret key sk_i for the identity id_i. Return to the Client $(\mathsf{Dec}_{sk_i}, \mathsf{Enc}_{pk_i})$. Set as $\mathcal{D}_i$ the distribution of the ciphertexts that correspond to plaintexts chosen uniformly from the plaintext space. Set $\mathsf{state} \leftarrow \mathsf{state} + 1$.
- Detect: On input dk and a circuit C, for $i = 1$ to state:
  - Compute $pk_i = f(param, id_i)$.
  - Choose $k = \lambda/\rho$ plaintexts uniformly at random from the the plaintext space and encrypt them under pk_i. We denote the corresponding ciphertexts as $c_1, \ldots, c_k$.
  - If for at least $\lambda/2$ plaintexts it holds that $C(c_i) = m_i$ then return marked

  Otherwise return $\mathsf{unmarked}$.

Figure 9: Watermarked Public Key Encryption Functionality from IBE

output a random bit. The latter case splits in two subcases, (a) there is Dec_{sk_i} such that C^* is *not* $(1-\rho/3)$-far from Dec_{sk_i} and Detect returns unmarked, and (b) there is Dec_{sk_i} such that C^* is *not* $(1-\rho/3)$-far from Dec_{sk_i} and Detect returns marked. In case (a), $\mathcal{B}$ will output a random bit. In case (b), we should examine how C^* behaves on input the tuple of ciphertexts $(c_{b,1}, \ldots, c_{b,\lambda/\rho})$ given by the Challenger. Using the same analysis with the case where $\mathcal{A}$ wins but $\mathcal{B}$ does not guess j^* correctly described in the previous paragraph, we show that $\mathcal{B}$ wins with probability at least $1/2$.

We provide the detailed probability analysis in Appendix C where we conclude that if $\mathcal{A}$ wins with non-negligible probability α, then, $\mathcal{B}$ breaks IND-CPA security with probability at least $\frac{1}{2} + \frac{\alpha}{2m} - \mathsf{negl}(\lambda)$.

6 Watermarking PKE funtionality from IBE

Finally, we present a watermarking scheme for the public key encryption functionality. Our construction relies on identity-based encryption (IBE) [5] and will allow for public detection of the watermark. The state, as before, will be of logarithmic size to the security parameter. As a reference, we provide the IBE definition and the security properties in the section D of the appendix.

Assuming an IBE scheme, one can construct a watermarking scheme for the public-key encryption functionality based on the following idea: The private marking key equals the master secret key of the IBE scheme. Then, the marking service (i.e., the Mark algorithm of the watermaking scheme), when invoked, sets $pk_i = f(param, id_i)$ for some deterministic function f[7] and then runs the private key generator of IBE, $\mathsf{IBE.Extract}(msk, f(param, id_i))$,

[7]In standard IBE the *id* of the user (i.e. email address or other unique identifier) serves as pk. Here,

to get the corresponding sk_i. The identities, id_i, are not given as input to Mark, instead, each identity is the next value of a counter that keeps the number of keys generated so far (which is stored at state). Detection works in a similar way to our construction in Section 5: try every possible public key (since by state you know the number of keys generated) and check if the given decryption circuit is watermarked by checking if for any of these public keys it correctly decrypts ciphertexts. We present our construction in Figure 9.

Theorem 6.1. *Let* $\langle \mathsf{IBE.Setup}, \mathsf{IBE.Extract}, \mathsf{IBE.Encrypt}, \mathsf{IBE.Decrypt} \rangle$ *be an IBE scheme with plaintext space of exponential size which satisfies correctness and IND-ID-CPA security. Then, the scheme of Figure 9 is a watermarking scheme for the public key encryption functionality. Namely, it satisfies Detection-correctness, is Functionality property-preserving, and achieves* ρ*-Unremovability and* $(1-\rho/3)$*-Unforgeability with* $\rho \geq \frac{1}{\mathsf{poly}(\lambda)}$.

The scheme of Figure 9 is based on the same idea as the scheme of Section 5, thus, the proofs for detection correctness, ρ-Unremovability and $(1-\rho/3)$-Unforgeability are very similar and thus omitted. However, this is a scheme showing how to watermark a cryptographic *functionality* and not an implementation and therefore we prove that it satisfies property-preserving as this is defined in Definition 3.5. We prove this property in Lemma 5 of Appendix D.

References

[1] André Adelsbach, Stefan Katzenbeisser, and Helmut Veith. Watermarking schemes provably secure against copy and ambiguity attacks. In *ACM workshop on Digital rights management*, 2003.

[2] Foteini Baldimtsi, Aggelos Kiayias, and Katerina Samari. Watermarking public-key cryptographic functionalities and implementations. In *ISC*, 2017.

[3] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[4] 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), 2012.

[5] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3), 2003.

[6] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In *PKC*, 2017.

[7] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.

since *id*'s are just a short counter value one might want to extend them in some deterministic way - else f could also the identity function.

[8] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.

[9] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, 1994.

[10] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *STOC*, 2016.

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

[12] Christian S. Collberg and Clark D. Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Trans. Software Eng.*, 28(8), 2002.

[13] Ingemar J Cox, Matthew L Miller, Jeffrey Adam Bloom, and Chris Honsinger. *Digital watermarking*, volume 1558607145. Springer, 2002.

[14] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*. Springer-Verlag New York, Inc., 1985.

[15] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[16] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, 1984.

[17] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. In *TCC*, 2007.

[18] Jonathan Katz. Analysis of a proposed hash-based signature standard". *Intern. Conference on Research in Security Standardisation (SSR)*, 2016.

[19] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *CCS*, 2013.

[20] Aggelos Kiayias and Qiang Tang. How to keep a secret: leakage deterring public-key cryptosystems. In *CCS*, 2013.

[21] Sam Kim and David J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In *CRYPTO*, 2017.

[22] David Naccache, Adi Shamir, and Julien P. Stern. How to copyright a function? In *PKC*, 1999.

[23] Ryo Nishimaki. How to watermark cryptographic functions. In *EUROCRYPT*, 2013.

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

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

$G_{\mathcal{A}}^{\mathsf{COR}}(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$:

1. The challenger sends Enc_{pk} to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ sends a message M to the challenger.
3. The challenger checks if $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(M)) \neq M$.
4. The game outputs 1 iff $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(M)) \neq M$.

Figure 10: The game for the correctness property with $\pi_{\mathsf{COR}} = 0$.

$G_{\mathcal{A}}^{\mathsf{IND-CPA}}(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$:

1. The challenger $\mathcal{C}$ sends Enc_{pk} to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ sends two messages M_0, M_1 to the challenger.
3. $\mathcal{C}$ chooses randomly $b \in \{0,1\}$ and sends $c_b = \mathsf{Enc}_{pk}(M_b)$ to $\mathcal{A}$.
4. $\mathcal{A}$ sends $b^* \in \{0,1\}$ to $\mathcal{C}$.
5. The game outputs 1 iff $b = b^*$.

Figure 11: The game for IND-CPA security property with $\pi_{\mathsf{IND-CPA}} = 1/2$.

[26] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC*, 2008.

[27] Christine I Podilchuk and Edward J Delp. Digital watermarking: algorithms and applications. *IEEE signal processing Magazine*, 18(4), 2001.

[28] Vidyasagar M Potdar, Song Han, and Elizabeth Chang. A survey of digital image watermarking techniques. In *INDIN*. IEEE, 2005.

[29] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

A (IND-CPA secure) Public Key Encryption Functionality

In this section we give a concrete example of a cryptographic functionality that is useful for our constructions. The public-key encryption functionality can be defined as a pair of algorithms $\langle \mathsf{Enc}, \mathsf{Dec} \rangle$ that should satisfy the properties of *Correctness* and *IND-CPA security*, as they are defined in Figures 13 and 14.

For the correctness definition note that we define it as a game (as per our Definition 2.3): an adversary would break correctness if he could come up with a message M such that its ciphertext, when encrypted under the challenge public key, would not decrypt correctly. A public key encryption functionality should be correct with probability $\pi_{\mathsf{COR}} = 0$.

Also, in both correctness and IND-CPA games, notice that according to Definition 2.3 they should get as input two algorithms C_1, C_2 that correspond to an instance of the candidate implementation. Thus, in Figures 13 and 14 the challenger receives as input $(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$ for some key pair (sk, pk). In a cryptographic implementation these keys would have been generated by the corresponding Gen algorithm.

$G_{\mathcal{A}}^{\mathsf{m-IND-CPA}}(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$:

1. The challenger $\mathcal{C}$ sends Enc_{pk} to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ sends two tuples of messages $(M_{0,1}, \ldots, M_{0,\ell})$, $(M_{1,1}, \ldots, M_{1,\ell})$ to the challenger.
3. $\mathcal{C}$ chooses randomly $b \in \{0,1\}$, and sends $(c_{b,1}, \ldots, c_{b,\ell})$ to $\mathcal{A}$, where $c_{b,j} = \mathsf{Enc}_{pk}(M_{b,j})$, for $j \in \{1, \ldots, \ell\}$.
4. $\mathcal{A}$ sends $b^* \in \{0,1\}$ to $\mathcal{C}$.
5. The game outputs 1 iff $b = b^*$.

Figure 12: IND-CPA security property for multiple messages, ℓ is polynomial in the security parameter λ.

A.1 IND-CPA security for multiple messages.

In Figure 15, we define the game for IND-CPA security for multiple-messages, where the adversary sumbits two tuples of messages to the Challenger instead of two different messages, as in the stardard IND-CPA security game. Using a hybrid argument, we can show that IND-CPA security implies IND-CPA security for multiple messages.

B (IND-CPA secure) Public Key Encryption Functionality

In this section we give a concrete example of a cryptographic functionality that is useful for our constructions. The public-key encryption functionality can be defined as a pair of algorithms $\langle \mathsf{Enc}, \mathsf{Dec} \rangle$ that should satisfy the properties of *Correctness* and *IND-CPA security*, as they are defined in Figures 13 and 14.

For the correctness definition note that we define it as a game (as per our Definition 2.3): an adversary would break correctness if he could come up with a message M such that its ciphertext, when encrypted under the challenge public key, would not decrypt correctly. A public key encryption functionality should be correct with probability $\pi_{\mathsf{COR}} = 0$.

Also, in both correctness and IND-CPA games, notice that according to Definition 2.3 they should get as input two algorithms C_1, C_2 that correspond to an instance of the candidate implementation. Thus, in Figures 13 and 14 the challenger receives as input $(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$ for some key pair (sk, pk). In a cryptographic implementation these keys would have been generated by the corresponding Gen algorithm.

B.1 IND-CPA security for multiple messages.

In Figure 15, we define the game for IND-CPA security for multiple-messages, where the adversary sumbits two tuples of messages to the Challenger instead of two different messages, as in the stardard IND-CPA security game. Using a hybrid argument, we can show that IND-CPA security implies IND-CPA security for multiple messages.

$\underline{G_{\mathcal{A}}^{\mathsf{COR}}(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})}$:

1. The challenger sends Enc_{pk} to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ sends a message M to the challenger.
3. The challenger checks if $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(M)) \neq M$.
4. The game outputs 1 iff $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(M)) \neq M$.

Figure 13: The game for the correctness property with $\pi_{\mathsf{COR}} = 0$.

$\underline{G_{\mathcal{A}}^{\mathsf{IND-CPA}}(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})}$:

1. The challenger $\mathcal{C}$ sends Enc_{pk} to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ sends two messages M_0, M_1 to the challenger.
3. $\mathcal{C}$ chooses randomly $b \in \{0,1\}$ and sends $c_b = \mathsf{Enc}_{pk}(M_b)$ to $\mathcal{A}$.
4. $\mathcal{A}$ sends $b^* \in \{0,1\}$ to $\mathcal{C}$.
5. The game outputs 1 iff $b = b^*$.

Figure 14: The game for IND-CPA security property with $\pi_{\mathsf{IND-CPA}} = 1/2$.

$\underline{G_{\mathcal{A}}^{\mathsf{m-IND-CPA}}(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})}$:

1. The challenger $\mathcal{C}$ sends Enc_{pk} to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ sends two tuples of messages $(M_{0,1}, \ldots, M_{0,\ell})$, $(M_{1,1}, \ldots, M_{1,\ell})$ to the challenger.
3. $\mathcal{C}$ chooses randomly $b \in \{0,1\}$, and sends $(c_{b,1}, \ldots, c_{b,\ell})$ to $\mathcal{A}$, where $c_{b,j} = \mathsf{Enc}_{pk}(M_{b,j})$, for $j \in \{1, \ldots, \ell\}$.
4. $\mathcal{A}$ sends $b^* \in \{0,1\}$ to $\mathcal{C}$.
5. The game outputs 1 iff $b = b^*$.

Figure 15: IND-CPA security property for multiple messages, ℓ is polynomial in the security parameter λ.

C $(1-\rho/3)$-Unforgeability proof of Section 5.

Recall that if $\mathcal{A}$ wins $(\rho/3)$-unforgeability game then, according to the security game of Figure 4, the following two conditions hold:

1. C^* is $(\rho/3)$-far from all the Decryption functions returned by the $\mathsf{CorruptOracle}$, e.g. $\mathsf{Dec}_{sk_{i_1}}, \ldots, \mathsf{Dec}_{sk_{i_q}}$.
2. The algorithm Detect returns marked. This means that there is $(j^*, \cdot) \in \mathsf{Marked}$ such that C^* decrypts correctly under sk_{j^*} at least $\lambda/2$ out of λ/ρ ciphertexts distributed according to $\mathcal{D}_j$.

We will first compute $\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins]$. Based on condition (ii), we define the event E_{cor} =" $(j^*, \cdot) \in \mathsf{Corrupted}$" The complementary event defined as $\neg E_{cor}$ ="$(j^*, \cdot) \notin \mathsf{Corrupted}$" (i.e. $j^* \in \mathsf{Marked} \setminus \mathsf{Corrupted}$). Therefore, we have that

$$\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins] = \mathsf{Pr}[\mathcal{B}\ wins \wedge E_{cor}|\mathcal{A}\ wins] + \mathsf{Pr}[\mathcal{B}\ wins \wedge \neg E_{cor}|\mathcal{A}\ wins] \quad (3)$$

$$\mathsf{Pr}[\mathcal{B}\ wins \wedge E_{cor}|\mathcal{A}\ wins] = \mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins, E_{cor}]\mathsf{Pr}[E_{cor}|\mathcal{A}\ wins]. \quad (4)$$

We will compute $\mathsf{Pr}[E_{cor}|\mathcal{A}\ wins]$. Since $\mathcal{A}$ wins, by condition (i) we have that for all $(j, \cdot) \in \mathsf{Corrupted}$ it holds that $C^* \nsim_{\gamma,\mathcal{D}_j} \mathsf{Dec}_{sk_j}$. Similarly to the ρ-unremovability proof, we denote as X_j the random variable which counts the number of ciphertexts from distribution $\mathcal{D}_j$ (out of λ/ρ) for which the circuit C^* returns the same output with Dec_{sk_j}. By claim 2.2, we have that the watermarking scheme preserves perfect correctness without introducing any decryption error and therefore X_j counts the number of ciphertexts from $\mathcal{D}_j$ which are correctly decrypted by C^* under sk_j.

We have that $\mu = E[X_j] \leq \frac{\lambda}{\rho} \cdot \frac{\rho}{3} = \lambda/3$. We define $\hat{X}_j$ the random variable with $\mu = \lambda/3$. Therefore, we have that $\mathsf{Pr}[X_j \geq \lambda/2] \leq \mathsf{Pr}[\hat{X}_j \geq \lambda/2]$. By using the following Chernoff bound

$$\mathsf{Pr}[X_j \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu}, \text{ for all } 0 < \delta < 1$$

and setting $\delta = 1/2$, we have that $\mathsf{Pr}[\hat{X}_j \geq \lambda/2] \leq e^{-\lambda/30}$. Thus, it holds that $\mathsf{Pr}[X_j \geq \lambda/2] \leq e^{-\lambda/30}$ which is negligible in λ. Since $\mathcal{A}$ makes q queries to the $\mathsf{CorruptOracle}$, we have that

$$\mathsf{Pr}[E_{cor}|\mathcal{A}\ wins] \leq q \cdot e^{-\lambda/30}, \text{ which is also negligible in } \lambda. \quad (5)$$

To simplify the presentation of the proof, we set $\mathsf{Pr}[E_{cor}|\mathcal{A}\ wins] = \mathsf{negl}(\lambda)$. By (3),(4), (5), we have that

$$\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins] = \mathsf{negl}(\lambda) + \mathsf{Pr}[\mathcal{B}\ wins \wedge \neg E_{cor}|\mathcal{A}\ wins]. \quad (6)$$

We now compute $\mathsf{Pr}[\mathcal{B}\ wins \wedge \neg E_{cor}|\mathcal{A}\ wins]$.

$$\begin{aligned}\mathsf{Pr}[\mathcal{B}\ wins \wedge \neg E_{cor}|\mathcal{A}\ wins] &= \mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}wins, \neg E_{cor}] \cdot \mathsf{Pr}[\neg E_{cor}|\mathcal{A}\ wins] \\ &= \mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}wins, \neg E_{cor}](1 - \mathsf{Pr}[E_{cor}|\mathcal{A}\ wins]). \quad (7)\end{aligned}$$

By (5), we have that

$$\mathsf{Pr}[\mathcal{B}\ wins \wedge \neg E_{cor}|\mathcal{A}\ wins] = (1 - \mathsf{negl}(\lambda))\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}wins, \neg E_{cor}]. \quad (8)$$

We will now compute $\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins, \neg E_{cor}]$. We define the event

Guess ="$\mathcal{B}$ guesses j^* where $(j^*, \cdot) \in \mathsf{Marked}$ and C^* decrypts correctly under sk_{j^*} at least $\lambda/2$ out of λ/ρ ciphertexts distributed according to $\mathcal{D}_{j^*}$."

Based on that event we have that

$$\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins, \neg E_{cor}] = \mathsf{Pr}[\mathcal{B}\ wins \wedge \mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}] + \mathsf{Pr}[\mathcal{B}\ wins \wedge \neg\mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}]. \quad (9)$$

We first compute $\mathsf{Pr}[\mathcal{B}\ wins \wedge \mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}]$.

$$\begin{aligned}\mathsf{Pr}[\mathcal{B}\ wins \wedge \mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}] &= \mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins, \neg E_{cor}, \mathsf{Guess}] \cdot \\ &\quad \mathsf{Pr}[\mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}] \\ &= 1 \cdot \mathsf{Pr}[\mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}]. \end{aligned} \quad (10)$$

Then,

$$\begin{aligned}\mathsf{Pr}[\mathcal{B}\ wins \wedge \neg\mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}] &= \mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins, \neg E_{cor}, \neg\mathsf{Guess}] \cdot \\ &\quad \mathsf{Pr}[\neg\mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}].\end{aligned}$$

We continue by computing $\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins, \neg E_{cor}, \neg\mathsf{Guess}]$. As we will see below, we need the following claim.

Claim 4.1: *Assume that a circuit C^* decrypts correctly $\lambda/2$ out of λ/ρ ciphertexts which correspond to encryptions of uniformly chosen plaintexts under a public key pk_i. Let pk_{i^*} a public key different than pk_i and two tuples of unifomly chosen plaintexts $(M_1, ..., M_{\lambda/\rho}), (M_1^*, ..., M_{\lambda/\rho}^*)$. If C^* is given as input $(c_1, ..., c_{\lambda/\rho})$ where $c_i = \mathsf{Enc}_{pk_{i^*}}(M_1)$ (for $i = 1, ...m\lambda/\rho$), then C^* decrypts c_i to M_i^* for at least $\lambda/2$ values if i only with negligible probability.*

As already discussed in the outline of the analysis there are three cases:

- C^* decrypts at least $\lambda/\rho - \lambda/2$ ciphertexts of the tuple (which are encrypted under pk_{j^*}) to messages completely irrelevant to the corresponding plaintexts of both tuples challenged in step 5 (i.e. $C^*(c_{b,i}) \neq M_{0,i}$ and $C^*(c_{b,i}) \neq M_{1,i}$). In this case, $\mathcal{B}$ outputs a random bit and wins with probability 1/2.

- C^* decrypts correctly at least $\lambda/2$ ciphertexts of the tuple $(c_{b,1}, \ldots, c_{b,\lambda/\rho})$ and therefore $\mathcal{B}$ guesses the correct bit and wins. In other words, in this case C^* decrypts correctly a portion of ciphertexts encrypted under two different keys. we can just set the probability that $\mathcal{B}$ guesses the correct bit in this case to be at least 1/2.

- C^* decrypts at least $\lambda/2$ out of λ/ρ ciphertexts in corresponding plaintexts of the opposite tuple (the one **not** selected by the Challenger). Namely, if the Challenger chooses $b = 0$, and encrypts the plaintexts $(M_{0,1}, \ldots, M_{\lambda/\rho})$ under pk_{j^*}, this would mean that C^* decrypts $c_{0,j}$ to $M_{1,j}$ (for at least $\lambda/2$ values of j). By claim 4.1, this holds only with negligible probability and and therefore the $\mathcal{B}$ guesses the correct bit with probability 1/2.

As a result, we have that $\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins, \neg E_{cor}, \neg\mathsf{Guess}] \geq 1/2$. Then,

$$\mathsf{Pr}[\mathcal{B}\ wins \wedge \neg\mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}] \geq \frac{1}{2}\big(1 - \mathsf{Pr}[\mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}]\big) \quad (11)$$

By (9),(10),(11), we have that

$$\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins, \neg E_{cor}] \geq \frac{1}{2} + \frac{\mathsf{Pr}[\mathsf{Guess}|\mathcal{A}\ wins, \neg E_{cor}]}{2} = \frac{1}{2} + \frac{1}{2m}. \tag{12}$$

By (8),(12), we have that

$$\mathsf{Pr}[\mathcal{B}\ wins \wedge \neg E_{cor}|\mathcal{A}\ wins] \geq (1 - \mathsf{negl}(\lambda))\Big(\frac{1}{2} + \frac{1}{2m}\Big). \tag{13}$$

By (6),(13),

$$\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins] = \mathsf{negl}(\lambda) + (1 - \mathsf{negl}(\lambda))\Big(\frac{1}{2} + \frac{1}{2m}\Big). \tag{14}$$

We will now compute $\mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins]$. We first define the event Abort = "$\mathcal{B}$ aborts". Recall that this event happens only if the adversary $\mathcal{A}$ makes a query to the $\mathsf{CorruptOracle}$ for the public key pk_{i^*} which is given to $\mathcal{B}$ by the IND-CPA challenger. Therefore, we have that

$$\mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins] = \mathsf{Pr}[\mathcal{B}\ wins \wedge \mathsf{Abort}|\neg\mathcal{A}\ wins] + \mathsf{Pr}[\mathcal{B}\ wins \wedge \neg\mathsf{Abort}|\neg\mathcal{A}\ wins] \tag{15}$$

$$\begin{aligned}\mathsf{Pr}[\mathcal{B}\ wins \wedge \mathsf{Abort}|\neg\mathcal{A}\ wins] &= \mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins, \mathsf{Abort}]\mathsf{Pr}[\mathsf{Abort}|\neg\mathcal{A}\ wins] \\ &= \frac{1}{2}\mathsf{Pr}[\mathsf{Abort}|\neg\mathcal{A}\ wins].\end{aligned} \tag{16}$$

We now compute $\mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}]$. We define the two possible events which may hold in the case that $\mathcal{B}$ does not abort and $\mathcal{A}$ does not win.

- $\mathsf{Unmarked}$ ="C^* is unmarked".
- Notfar = " There is $(j, \cdot) \in \mathsf{Corrupted}$ s.t. C^* is **not** $(\rho/3)$-far from Dec_{sk_j}."

Notice that these two events are not disjoint, since when the event Notfar takes place Detect can return either marked or unmarked on input C^*. However, since $\mathcal{A}$ does not win, it holds that

$$\mathsf{Pr}[\neg\mathsf{Unmarked}|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}] = \mathsf{Pr}[\neg\mathsf{Unmarked} \wedge \mathsf{Notfar}|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}] \tag{17}$$

In the case where Detect returns unmarked $\mathcal{B}$ will output a random bit. In the case where Detect returns marked, we should examine how C^* behaves on input the tuple of ciphertexts $(c_{b,1}, \ldots, c_{b,\lambda/\rho})$ given by the Challenger. For this case we distinguish three cases in exactly the same way as in the analysis for the computation of $\mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins, \neg E_{cor}, \neg\mathsf{Guess}]$. Hence, using the claim 4.1, we conclude that when Notfar happens and Detect returns marked on input C^*, $\mathcal{B}$ wins with probability at least $1/2$. Therefore,

$$\begin{aligned}\mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}] &= \mathsf{Pr}[\mathcal{B}\ wins \wedge \mathsf{Unmarked}|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}] \\ &\quad + \mathsf{Pr}[\mathcal{B}\ wins \wedge \neg\mathsf{Unmarked} \wedge \mathsf{Notfar}|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}]\end{aligned} \tag{18}$$

$$\mathsf{Pr}[\mathcal{B}\ wins \wedge \mathsf{Unmarked}|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}] = \frac{1}{2}\mathsf{Pr}[\mathsf{Unmarked}|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}]. \tag{19}$$

$$\mathsf{Pr}[\mathcal{B}\ wins \wedge \neg\mathsf{Unmarked} \wedge \mathsf{Notfar}|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}] =$$
$$\mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}, \mathsf{Notfar}, \neg\mathsf{Unmarked}] \cdot \mathsf{Pr}[\neg\mathsf{Unmarked}|\ \neg\mathcal{A}\ wins, \neg\mathsf{Abort}] \quad (20)$$

As we analyzed before, by claim 4.1

$$\mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}, \mathsf{Notfar}, \neg\mathsf{Unmarked}] \geq 1/2. \quad (21)$$

Then,

$$\mathsf{Pr}[\mathcal{B}\ wins \wedge \neg\mathsf{Unmarked} \wedge \mathsf{Notfar}|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}] \geq \frac{1}{2}\mathsf{Pr}[\neg\mathsf{Unmarked}|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}]. \quad (22)$$

By (18),(19),(22), we have that

$$\mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins, \neg\mathsf{Abort}] \geq \frac{1}{2}. \quad (23)$$

By (15),(16),(23), we have that

$$\mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins] \geq \frac{1}{2}\Big(1 - \mathsf{Pr}[\neg\mathsf{Abort}|\neg\mathcal{A}\ wins]\Big) + \frac{1}{2}\mathsf{Pr}[\neg\mathsf{Abort}|\neg\mathcal{A}\ wins] = \frac{1}{2}. \quad (24)$$

Finally,

$$\mathsf{Pr}[\mathcal{B}\ wins] = \mathsf{Pr}[\mathcal{B}\ wins|\mathcal{A}\ wins]\mathsf{Pr}[\mathcal{A}\ wins] + \mathsf{Pr}[\mathcal{B}\ wins|\neg\mathcal{A}\ wins]\mathsf{Pr}[\neg\mathcal{A}\ wins] \quad (25)$$

By (14), (24),

$$\mathsf{Pr}[\mathcal{B}\ wins] \geq \frac{1}{2} + \frac{\alpha}{2m} - \mathsf{negl}(\lambda), \quad (26)$$

Hence, $\mathcal{B}$ breaks $IND-CPA$ security with non-negligible probability. □

D Identity based encryption

D.1 IBE preliminaries

Following Definition 2.3 in Section 2, identity-based encryption is a functionality with three algorithms IBE.Extract, IBE.Encrypt and IBE.Decrypt and two properties, i.e. correctness and IND-ID-CPA security as these are defined in Figures 16, 17 respectively. Next, we give a definition of an IBE scheme. We follow the definition of Boneh-Franklin paper [5].

Definition D.1 (IBE scheme)**.** An IBE scheme consists of four algorithms, IBE.Setup, IBE.Extract, IBE.Encrypt and IBE.Decrypt described as follows:

- IBE.Setup(1^λ): outputs a master key msk for the *private key generator (PKG)* and some system parameters $param$.
- IBE.Extract$(param, msk, \mathrm{ID})$: outputs a private key for ID, sk_{ID} for the identity ID.
- IBE.Encrypt$(param, \mathrm{ID}, M)$: produces the public key pk_{ID} for ID and outputs an encryption C for the plaintext M under pk_{ID}.
- IBE.Decrypt$(param, sk_{\mathrm{ID}}, C)$: outputs a plaintext M as the decryption of the ciphertext C.

$G_{\mathcal{A}}^{\mathsf{COR}}(\mathsf{IBE.Extract}_{msk}, \mathsf{IBE.Encrypt}_{param}, \mathsf{IBE.Decrypt}(\mathsf{IBE.Extract}_{\mathsf{msk}}))$:

1. The Challenger sends $\mathsf{IBE.Encrypt}_{param}$ to $\mathcal{A}$.
2. The adversary sends an identity ID and a message M to the Challenger.
3. The Challenger checks if $\mathsf{IBE.Decrypt}(\mathsf{IBE.Extract}_{msk}(ID), \mathsf{IBE.Encrypt}_{param}(ID, M)) = M$.
4. The game outputs 1 iff $\mathsf{IBE.Decrypt}(\mathsf{IBE.Extract}_{msk}(ID), \mathsf{IBE.Encrypt}_{param}(ID, M)) = M$.

Figure 16: Correctness property with $\pi_{\mathsf{COR}} = 0$.

$G_{\mathcal{A}}^{\mathsf{IND-ID-CPA}}(\mathsf{IBE.Extract}_{msk}, \mathsf{IBE.Encrypt}_{param}, \mathsf{IBE.Decrypt}(\mathsf{IBE.Extract}_{\mathsf{msk}}))$:

1. $\mathcal{A}$ receives $\mathsf{IBE.Encrypt}_{param}$ by the Challenger.
2. $\mathcal{A}$ can request private keys for a number of identities its choice, e.g. $D_1, \ldots, ID_q$. Given an identity ID_i, the Challenger replies by running $\mathsf{Extract}_{\mathsf{msk}}(\mathsf{ID_i})$ which outputs the corresponding secret key.
3. $\mathcal{A}$ sends ID, M_0, M_1 to the Challenger such that ID has not been issued as query in the previous step and $M_0 \neq M_1$.
4. The Challenger chooses uniformly at random $b \leftarrow \{0, 1\}$, computes $C_b = \mathsf{IBE.Encrypt}_{param}(ID, M_b)$ and returns C_b to the adversary.
5. $\mathcal{A}$ may repeat step 2 with the restriction that it cannot request the secret key for the identity sent to the Challenger in the previous step.
6. $\mathcal{A}$ outputs $b^* \in \{0, 1\}$.
7. The game outputs 1 iff $b = b^*$.

Figure 17: IND-ID-CPA property with $\pi_{\mathsf{IND-ID-CPA}} = 1/2$.

D.2 Proof of Theorem 2

Theorem D.1. *Let* $\langle \mathsf{IBE.Setup}, \mathsf{IBE.Extract}, \mathsf{IBE.Encrypt}, \mathsf{IBE.Decrypt} \rangle$ *be an IBE scheme with plaintext space of exponential size which satisfies correctness and IND-ID-CPA security. Then, the scheme of Figure 9 is a watermarking scheme for the public key encryption functionality. Namely, it satisfies Detection-correctness, is Functionality property-preserving, and achieves* ρ*-Unremovability and* $(1-\rho/3)$*-Unforgeability with* $\rho \geq \frac{1}{\mathsf{poly}(\lambda)}$.

The scheme of Figure 9 is based on the same idea as the scheme of Section 5, thus, the proofs for detection correctness, ρ-Unremovability and $(1-\rho/3)$-Unforgeability are very similar and thus omitted. However, this is a scheme showing how to watermark a cryptographic *functionality* and not an implementation and therefore we prove that it satisfies property-preserving as this is defined in Definition 3.5. We prove this property in the following lemma.

Lemma 5. *If the IBE scheme satisfies perfect correctness and IND-ID-CPA security, then the watermarking scheme of Figure 9 is property-preserving for the public key encryption functionality according to Definition 3.5.*

Proof. In the next two claims we show separately that the scheme of Figure 9 preserves correctness and IND-CPA security, in the sense of Definition 3.5.
Claim 5.1: If the IBE scheme satisfies IND-ID-CPA security, then for any PPT adversary $\mathcal{A}$ it holds that

$$\mathsf{Pr}[G_{\mathcal{A}}^{\mathsf{wm-IND-CPA}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be a PPT adversary which breaks property-preserving game for IND-CPA security with non-negligible advantage α, i.e. $\mathsf{Pr}[G_{\mathcal{A}}^{\mathsf{wm-IND-CPA}}(1^\lambda) = 1] \geq 1/2 + \alpha$. We construct a PPT adversary $\mathcal{B}$ which breaks IND-ID-CPA security with the same probability as follows. For simplicity, we omit the function f in this proof.

1. $\mathcal{B}$ receives $\mathsf{IBE.Encrypt}_{param}$ by the IND-ID-CPA Challenger.
2. $\mathcal{B}$ initializes the variables $i \leftarrow 0$, $\mathsf{state} \leftarrow 0$ [8] and forwards to $\mathcal{A}$ the public parameters $\mathsf{IBE.Encrypt}_{param}$.
3. Whenever $\mathcal{A}$ makes a $\mathsf{ChallengeOracle}$ query, $\mathcal{B}$ chooses as the next identity the value $i+1$, and responds with $\mathsf{IBE.Encrypt}_{param}(i+1)$. $\mathcal{B}$ updates state to $\mathsf{state}+1$ and i to $i+1$. When $\mathcal{A}$ makes a $\mathsf{CorruptOracle}$ query for an index i, $\mathcal{B}$ makes an Extract query in order to receive the secret key sk_i for identity i, and sends Dec_{sk_i} to $\mathcal{A}$ together with the corresponding encryption function. $\mathcal{B}$ answers $\mathsf{DetectOracle}$ queries by just running the algorithm Detect of the scheme.
4. $\mathcal{A}$ chooses an instance i, which means that the (standard) IND-CPA security game will be run for the public key of identity $ID = i$, and then chooses to messages M_0, M_1.
5. $\mathcal{B}$ also submits $ID = i$ and M_0, M_1 to the Challenger and receives a cipheretxt c_b.

[8] The variables, i, state in this scheme have the same value.

6. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

From the structure of above reduction, it is easy to show that $\mathcal{B}$ breaks IND-ID-CPA security with the same advantage with $\mathcal{A}$, which is non-negligible. □

Claim 5.2: If the IBE scheme satisfies perfect correctness, then for any PPT adversary $\mathcal{A}$ it holds that

$$\mathsf{Pr}[G_{\mathcal{S}}^{\mathsf{wm}-\mathsf{COR}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

Proof. This proof is straghtforward. Due to the perfect correctness of the IBE scheme, correctness of the watermarking scheme holds with no error.

□

□