Composability in Watermarking Schemes

Jiahui Liu[1] and Mark Zhandry[2]

[1]Massachusetts Institute of Technology, `jiahuiliu.crypto@gmail.com`
[2]NTT Research, `mark.zhandry@ntt-research.com`

Abstract

Software watermarking allows for embedding a mark into a piece of code, such that any attempt to remove the mark will render the code useless. Provably secure watermarking schemes currently seems limited to programs computing various cryptographic operations, such as evaluating pseudorandom functions (PRFs), signing messages, or decrypting ciphertexts (the latter often going by the name "traitor tracing"). Moreover, each of these watermarking schemes has an ad-hoc construction of its own.

We observe, however, that many cryptographic objects are used as building blocks in larger protocols. We ask: just as we can compose building blocks to obtain larger protocols, can we compose watermarking schemes for the building blocks to obtain watermarking schemes for the larger protocols? We give an affirmative answer to this question, by precisely formulating a set of requirements that allow for composing watermarking schemes. We use our formulation to derive a number of applications.

1 Introduction

Watermarking is an old idea, which aims to embed a mark in some object, such that any attempt to remove the mark destroys the object. In software watermarking, this means embedding a mark into program code, such that any attempt to remove the code will make the code useless. Such watermarking aims to deter piracy by identifying the source of pirated software. Recently, software watermarking has become an active area of research within cryptography, with numerous positive results for watermarking cryptographic functionalities, such as trapdoor functions [Nis13], pseudorandom functions [CHN$^+$16, KW17, GKWW21], decryption [CFNP00] (under the name "traitor tracing"), and more [GKM$^+$19].

In this work, we initiate the study of *composing* watermarked functionalities. That is, if a cryptographic primitive A (or perhaps several primitives) is used to build a primitive B, can we use a watermarking scheme for A to realize a watermarking scheme for B? Our aim is to show when such a composition is possible, based on properties of the construction and security proof for B using A.

1.1 Motivation

Abstractions are central to cryptography, as they allow for decomposing various tasks into smaller building blocks, which can then be instantiated independently. The literature is full of results that show how to generically realize one abstraction assuming solutions to one or more input abstractions.

Given that cryptographic primitives are often composed, and that most watermarking schemes with provable security are for cryptographic primitives, an interesting question is whether watermarking schemes can be composed. To the best of our knowledge, this question has not been previously asked. In contrast, existing watermarking schemes are each developed "from scratch." Even if the underlying techniques are similar, watermarking schemes for different primitives must go through separate constructions and security proofs. This can be a time-consuming process.

What does it mean to watermark a PRF? As a running example throughout this this introduction, we will use pseudorandom functions (PRFs), which are one of the main workhorses in symmetric key cryptography, and are used as a central component in many higher-level protocols. In [CHN$^+$16], the authors show how to watermark the evaluation procedure for a certain class of PRFs. We point out, however, that PRFs are typically not considered a cryptographic end goal, but rather a tool used to build other cryptographic notions. So *what, then, is the utility of watermarking a PRF?*

Another fundamental question is the following: for important reasons that we will not get into here, the watermarking guarantee proved by [CHN$^+$16] (and all subsequent work on watermarking PRFs) is weaker than one may expect. Namely, they show that it is impossible to remove the mark without causing the program to fail on *random* inputs. Certain PRFs called "weak PRFs" are only guaranteed secure when the adversary sees evaluations on random inputs, and so for this reason may authors (e.g. [GKWW21, MW22, KN22]) refer to such a scheme as watermarking "weak PRFs." Weak PRFs can be used as building blocks for many applications, though not as many as ordinary PRFs. In the context of using PRFs as a building block, what is implication of watermarking a *weak* PRF?

Composition of Watermarking Schemes. Our thesis is that watermarking, at least in many cases, should be defined and executed in such a way as to be composable, allowing watermarking schemes for building blocks to generically imply watermarking schemes for higher-level protocols. Watermarking a weak PRF, for example, should generically enable watermarking for many applications of weak PRFs, such as CPA-secure symmetric encryption. Naturally, a more ambitious goal is: watermarking a message-authentication code scheme or a digital signatures scheme, when composed with a watermarkable PRF scheme, should lead to a watermarkable CCA-secure SKE scheme.

Not all compositions support watermarking. Certainly, not all composition results from cryptography can be used to compose watermarking schemes. For example, pseudorandom functions can be built from any pseudorandom generator (PRG) [GGM86]. But this does not seem to yield a viable path toward constructing a watermarking scheme for PRFs. After all, what would it even mean to watermarking a PRG, given that the evaluation algorithm for the PRG is public?

1.2 Overview of Our Results

Defining Watermarking Security. Most cryptographic primitives are defined by an interactive game between an adversary A and challenger. Given a security game, we can translate the game into a watermarking definition, as follows. The attacker gets a watermarked secret key, and then tries to produce a program A. We say that A is "good" if it can win the security game with non-negligible advantage. We then require the existence of a tracing algorithm, which can extract the mark from any good program A produced by the adversary. In the case of watermarking public key decryption functionalities, our notion corresponds exactly to existing notions of traitor tracing, since the security game is non-interactive. However, for other primitives, our definition is potentially stronger than existing notions: existing notions only ask for mark extraction for non-interactive programs A that can evaluate some function, whereas we must extract from any A which wins a security experiment, which is potentially interactive. The strengthened definitions will be crucial for our composition theorem, which we now describe.

Composition Theorem. Our first result is a composition theorem, which gives conditions under which a construction of a target primitive P from input primitives $P_1, \cdots, P_k$ can be turned into a compiler for watermarking schemes.

Theorem 1.1 (Main Theorem (Informal)). *If the construction of a target primitive P from input primitives $P_1, \cdots, P_k$ satisfies some given conditions and the watermarking schemes for $P_1, \cdots, P_k$ satisfy the above watermarking security definition, then we can compose the construction to watermarking schemes for $P_1, \cdots, P_k$, black-boxly into a watermarking scheme for P that satisfies the above watermarking definition.*

Our conditions apply to a large class of constructions. They are, very roughly, as follows:

- The construction of P from $P_1, \cdots, P_k$ is black-box. Moreover, the secret key sk for P is $\mathsf{sk} = (\mathsf{sk}_1, \cdots, \mathsf{sk}_k)$ where sk_i is the secret key for P_i.
- The security proof for P turns an adversary A for P into adversaries $A_1, \cdots, A_k$ for $P_1, \cdots, P_k$, respectively, such that if A has non-negligible advantage, so does *at least* one of the P_i. Typical proofs in cryptography utilizing hybrid proofs will usually have this form.
- Moreover, we require a property of the security proof, which we call a "watermarking-compatible reduction". This is a rather technical definition, but roughly we allow the security games for $P, P_1, \cdots, P_k$ to consist of two phases, where the adversary's winning condition is only dependent on the second phase. We require that the reduction respects the phases, in the sense that the second stages of $A_1, \cdots, A_k$ only depend on the second stage of A.
- Depending on the construction and reduction, *not all* input primitives $P_1, \cdots, P_k$ have to be watermarkable to give a watermarking construction for the target primitive P. That is, we can compose the "plain" constructions of these input primitives with the watermarkable versions of the other input primitives to give a watermarkable construction for P.
- If all input primitives (that need to be watermarkable) have collusion-resistant watermarking security, the resulting watermarkable P also satisfies collusion-resistant security. If all input primitives can be traced using only a public key, the same is true of P.

Many reductions in the literature are watermarking-compatible. For example, consider constructing CPA-secure symmetric encryption from weak PRFs. Here, the first stage for CPA-security consists of all queries occurring prior to the challenge query, and the second stage consists of the challenge query and all subsequent queries. The winning condition does not depend on any first stage CPA queries (or second stage). The first and second stages for the weak PRF are just two rounds of queries to the weak PRF oracle, and here again the win condition does not depend on the actually queries. Thus, this reduction is watermarking compatible.

A non-example would be many proofs involving signatures as the target primitive P. The issue is that, the winning condition for a signature scheme security game is that the adversary produces a "new" signature on a message that was not seen in a previous query. But checking this win condition requires knowing all the queries. So if there is a first phase where the attacker can make signature queries, then the win condition is not solely dependent on the second stage. If we let the winning condition depend only on the second stage, the adversary can trivially win by querying a signature in the first stage and give it to the second stage adversary as the final output.

Applications. We demonstrate that many well-known cryptographic constructions have watermarking-compatible reductions, and therefore we can compose the watermarkable constructions of the input primitives to obtain watermarkable constructions of the target primitive. Within the scope of this work, we give the following examples:

- Two most simple examples are:
  - Watermarkable CPA-secure secret-key encryption scheme from watermarkable weak PRF
  - Watermarkable CCA2-secure secret-key encryption scheme from watermarkable weak PRF and watermarkable MACs.
- Some more advanced examples are:
  - Watermarkable CCA2-secure public-key encryption from watermarkable selectively secure identity based encryption and strong one-time signatures, which can in turn be based on LWE. Here, the strong one-time signature we need is a "plain scheme" which does not need to be watermarkable.
  - Watermarkable CCA2-secure PKE from watermarkable CPA-secure PKE and NIZK (without watermarking), which can in turn be based on LWE too.
  - Watermarkable weak pseudorandom permutation from watermarkable weak PRF.
  - Watermarkable CCA2-secure hybrid encryption scheme from watermarkable CCA2-secure PKE and CCA2-secure SKE (without watermarking).

Additionally, we show that all the input primitives (weak PRF, CPA-secure PKE, selective IBE, signatures, etc.) in the above examples have constructions that satisfy our security definition for watermarking. We can obtain these constructions by using or modifying the existing watermarking schemes in [GKM$^+$19], [GKWW21, MW22]. Therefore, the above composed schemes all have concrete constructions.

We also briefly discuss how the functional encryption construction in [GKP$^+$13] is also watermarking compatible. We cannot possibly elaborate all the concrete examples of watermarking compatible reductions within the scope of this work, but given our generic framework of composition, one can easily verify whether a construction is watermarking compatible by looking into its security proof.

1.3 Other Related Work

[Nis20] presents a general framework for constructing watermarking schemes for any primitive which admits a certain "all-but-one" reduction. The work and ours focus on different aspects of watermarking: theirs is focused on constructing watermarking schemes in the first place, whereas our composition theorem shows how to combine watermarking schemes. We also note that the framework in [Nis20] seems very much tied to the collusion-free setting, whereas ours is much more general, and can accommodate collusions if the input tracing mechanisms are collusion resistant.

1.4 Technical Overview

Watermarking We first briefly recall the definition of watermarking a cryptographic primitive: in a watermakrable cryptographic scheme, apart from the usual evaluation algorithms (such as key generation, encrypt, decrypt or sign and verify), it additionally has a Mark algorithm and an Extract algorithm (as well as the corresponding marking and extraction keys). The Mark algorithm allows one to embed a mark into the secret key used to evaluate the cryptographic functionality; the Extract key allows one to extract a mark from an allegedly marked key. The watermarking security, usually referred to as "unremovability", states that given a marked secret key with an adversarially requested mark τ, the adversary should not be able to produce a circuit that has the same functionality as the secret key such that the above mark τ cannot be extracted from this adversarial circuit.

As explored in this work and some previous works ([GKM$^+$19, GKWW21]), one important definitional aspect in the above security lies in what we mean by the "adversary's output circuit has the same functionality as the original secret key". We will elaborate in the following sections of the overview.

Watermarking-Compatible Reduction Before we go into how we compose watermarking schemes, we expand more on the type of reductions that allow us to build a watermarking composition upon. We call these reductions watermarking-compatible. We will then give a concrete example to help comprehension.

Consider a black-box construction of a target primitive P from input primitives $P_1, \cdots, P_k$ and consider a reduction algorithm $\mathcal{B}$ from security of P to the security of P_i: we let both the adversary $\mathcal{A}$ and the reduction $\mathcal{B}$ be divided into two stages. In the first stage, stage-1 adversary $\mathcal{A}_1$, for the security game of P, receives some public parameters from stage-1 $\mathcal{B}_1$ and makes some queries; $\mathcal{B}_1$ answers these queries by making queries to the oracle provided by the challenger of the security game for P_i.

Entering the second stage, as one can expect, $\mathcal{A}_1$ can give an arbitrary state it to stage-2 adversary $\mathcal{A}_2$. However, what $\mathcal{B}_1$ can give to the second stage reduction $\mathcal{B}_2$ is more restricted: $\mathcal{B}_1$ can give all the public parameters to the second stage but *none of the queries* made by $\mathcal{A}_1$, to $\mathcal{B}_2$. $\mathcal{B}_2$ continues to simulate the query stage to answer $\mathcal{A}$'s queries. In particular, $\mathcal{B}$ records $\mathcal{A}_2$'s queries. Note that the challenge phase of the security game where $\mathcal{A}_2$ (and resp. $\mathcal{B}$) receives its challenge from the challenger always happens in stage 2 [1].

In the final output phase, $\mathcal{B}_2$'s answer to the challenger in game P_i will be dependent on (some of) the following pieces of information: challenger's challenge input to $\mathcal{B}_2$, $\mathcal{A}_2$'s queries made during stage 2, $\mathcal{B}_2$'s randomness used to prepare the challenge input for $\mathcal{A}_2$ and $\mathcal{A}_2$'s final output.

[1]But as we will see in some concrete examples, $\mathcal{A}$ can commit to some "challenge messages" in stage 1, which will be taken into stage-2 as an "auxiliary input" and later used by the challenger to prepare the challenge.

We give some intuition on why we need B to be "oblivious" about $\mathcal{A}$'s queries in stage 1 of the above reduction. Looking forward, in the actual watermarking (unremovability) security game, we can replace "answering $\mathcal{A}$'s queries in stage 1" with a giving out a watermarked secret key to $\mathcal{A}$, where $\mathcal{B}$ will have no idea what inputs $\mathcal{A}$ has evaluated on using the key. We therefore model the reduction as the above to capture this scenario.

Example: CCA2-Secure Secret-Key Encryption To make the above abstract description concrete, we take an example of the reduction from CCA2-security to weak PRF and MAC.

We briefly recall the textbook construction: the scheme's secret key consists of the PRF's secret key sk_1 and MAC's secret key sk_2. The encryption algorithm, on input message m, computes ciphertext $\mathsf{ct} = (r, \mathsf{ct}' = \mathsf{PRF.Eval}(\mathsf{sk}_1, r) \oplus m, \mathsf{sig})$, where $\mathsf{sig} \leftarrow \mathsf{MAC.Sign}(\mathsf{sk}_2, (r, \mathsf{ct}'))$. The decryption algorithm will first verify the signature sig on (r, ct'), if yes continue to decrypt using sk_1, else abort.

It's not hard to see how the CCA2 security game can fit into the format of a two-stage game we have depicted above: stage 1 consists simply of letting the adversary making queries to the encryption and decryption oracles. Entering stage 2, the adversary $\mathcal{A}$ is allowed to make more queries and then submits the challenge plaintexts; then $\mathcal{A}$ receives challenge ciphertexts from the challenger. $\mathcal{A}$ continues to make more admissible queries and finally outputs its answer.

By viewing the security game in two stages, we will recall the security proof for CCA2 security. The proof can go through a case-by-case analysis:

1. In case 1, in stage 2 of the CCA2 security game, there is a decryption query from $\mathcal{A}$ of the form $\mathsf{ct} = (\mathsf{ct}_0, \sigma)$, such that it has never been the output of an encryption query. Also, the decryption oracle did not output $\perp$ on this query.
2. In case 2, there is no such decryption query in stage 2.

In the first case, we can do a reduction to unforgeability of MAC:

- $\mathcal{A}_{\mathsf{MAC}}$ samples its own PRF secret key sk_1. For stage 1: For any encryption query from stage-1 adversary, $\mathcal{A}^1$, stage-1 reduction $\mathcal{A}^1_{\mathsf{MAC}}$ will simulate the response as follows: it will compute the ct_0 part of ciphertext and then query the signing oracle $\mathsf{MAC.Sign}(\mathsf{sk}_2, \cdot)$ in the security game G_{MAC}. For any decryption query, $\mathcal{A}^1_{\mathsf{MAC}}$ will query the verification oracle $\mathsf{MAC.Verify}(\mathsf{sk}_2, \cdot)$ first and decrypt those whose response is not $\perp$.
- After entering stage 2, for any encryption or decryption query from $\mathcal{A}^2$, $\mathcal{A}^2_{\mathsf{MAC}}$ will still simulate the response as the above. Recall that $\mathcal{A}^2_{\mathsf{MAC}}$ will not get to see any queries made in stage 1 and $\mathcal{A}^2_{\mathsf{MAC}}$ will record all queries from $\mathcal{A}_2$.
- In the challenge phase, $\mathcal{A}^2$ sends in challenge messages (m_0, m_1); $\mathcal{A}^2_{\mathsf{MAC}}$ flips a coin $b \leftarrow \{0, 1\}$, prepares and sends the signed encryption ct^* of m_b to $\mathcal{A}^2$. $\mathcal{A}^2_{\mathsf{MAC}}$ continues to simulate the encryption and decryption oracles for $\mathcal{A}^2$, on queries $\mathsf{ct} \neq \mathsf{ct}^*$.
- In the end, $\mathcal{A}^2_{\mathsf{MAC}}$ will look up $\mathcal{A}^2$'s queries: find a decryption query $\mathsf{ct} = (\mathsf{ct}', \sigma)$ such that it has never been the output of an encryption query and the decryption oracle did not output $\perp$ on this query. $\mathcal{A}^2_{\mathsf{MAC}}$ output (ct', σ) as its forgery.

In the second case, we consider a reduction $\mathcal{A}_{\mathsf{PRF}}$ to the weak pseudorandomness of the PRF. In the following reduction, for he sake of simplicity, let's consider a variant of the weak PRF game: the adversary $\mathcal{A}_{\mathsf{PRF}}$ is given adaptive query access to an oracle $F : \{0, 1\}^n \rightarrow \{0, 1\}^m$ that computes a PRF function; there will be a challenge phase where a challenge input r^* is sampled at random; $\mathcal{A}_{\mathsf{PRF}}$ will receive one of $(r^*, F(r^*))$ or $(r^*, y \leftarrow \{0,1\}^m)$ at random and try to tell which one it receives [2].

- In stage 1: $\mathcal{A}_{\mathsf{PRF}}$ samples its own MAC key sk_2. For any encryption query from $\mathcal{A}^1$, $\mathcal{A}^1_{\mathsf{PRF}}$ will simulate the response as follows: it will query the PRF oracle $F(\cdot) := \mathsf{PRF.Eval}(\mathsf{sk}_1, \cdot)$ on a fresh r of its own choice. After obtaining $(r, F(r))$, $\mathcal{A}_{\mathsf{PRF}}$ signs the message $(r, F(r) \oplus m)$ using sk_2 and sends the entire ciphertext to $\mathcal{A}$.
  Similarly, for decryption queries, $\mathcal{A}_{\mathsf{PRF}}$ first verifies the signature and then queries $F(\cdot)$ oracle to decrypt.
- After entering stage 2, for any encryption query from $\mathcal{A}^2$, $\mathcal{A}^2_{\mathsf{PRF}}$ will still simulate the response as the above. Recall that $\mathcal{A}^2_{\mathsf{PRF}}$ will not get to see any queries made in stage 1.
- In the challenge phase, $\mathcal{A}^2_{\mathsf{PRF}}$ will receive a challenge input-output pair (r^*, y^*) from the weak PRF challenger: $r^* \leftarrow \{0,1\}^\ell, y^* \in \{0,1\}^\ell$ where y^* is either $\mathsf{PRF.Eval}(\mathsf{sk}_1, r^*)$ or uniformly random.
- $\mathcal{A}^2$ sends in challenge messages, (m_0, m_1); $\mathcal{A}^2_{\mathsf{PRF}}$ flips a coin $b \leftarrow \{0,1\}$ and sends $\mathsf{ct} = (r^*, y^* \oplus m_b, \sigma^*)$ to $\mathcal{A}^2$.
- If $\mathcal{A}^2$ guesses the correct b, then $\mathcal{A}^2_{\mathsf{PRF}}$ output 0, for "y^* is $\mathsf{PRF.Eval}(\mathsf{sk}_1, r^*)$", else $\mathcal{A}^2_{\mathsf{PRF}}$ output 1, for "y^* is uniformly random".

Note that in the above reductions, both $\mathcal{A}_{\mathsf{PRF}}$ and $\mathcal{A}_{\mathsf{MAC}}$ do not need the stage-1 queries from $\mathcal{A}$ to help them win their own games.

Definition and Composition Framework of Watermarking Now we relate the above reduction to the watermarking goal: in our watermarking (i.e. unremovability) security game for primitive P, the watermarking adversary $\mathcal{A}$ acts like a "stage-1" adversary in the original security game for the underlying primitive P. Then $\mathcal{A}$ produces a program C —-this signifies the end of "stage 1" of the security game.

A notable feature of our extraction algorithm is the following. The adversarial program C produced by $\mathcal{A}$ will be treated as a stage-2 adversary by the extraction algorithm: the extraction algorithm will try to extract a watermark by having only black-box access to C. Since C is supposed to function like a stage-2 adversary that wins the security game of P, the extraction procedure ensures C operates properly by simulating the stage-2 security game for C.

Finally, $\mathcal{A}$ wins if C is a "good" program in winning the corresponding security game for P, but no valid watermarks can be extracted from C.

Suppose a target primitive P can be built from input primitives $P_1, \cdots, P_k$ via watermarking-compatible reductions, and each watermarking scheme for P_i satisfies our definition, then we can compose the watermarkable versions of $P_1, \cdots, P_k$ to give a watermarkable P.

The composition construction works roughly as follows:

1. A watermarkable P's key generation algorithm is similar to the unwatermarkable (plain) construction of P: by generating all keys of $P_1, \cdots, P_k$ and concatenate them, for secret/public keys respectively. The extra step is that now we also concatenate the marking keys and extraction keys generated from the watermarkble $P_1, \cdots, P_k$.
2. How P evaluates is exactly the same as in the plain construction of P form $P_1, \cdots, P_k$: by running the $P_1, \cdots, P_k$ algorithms as black-box subroutines.

[2]Watermarkable weak PRF with this type of security can be constructed in [GKM+19].

3. To watermark P, we simply watermark the secret keys of the $P_1, \cdots, P_k$ respectively. The watermarked key for P is then just the concatenation of the watermarked keys for $P_1, \cdots, P_k$. Since the construction is black-box and assumes the key is just a tuple of keys for $P_1, \cdots, P_k$, the evaluation with watermarked keys is still the same, running the evaluation algorithm with black-box from subroutines $P_1, \cdots, P_k$, except now with the marked keys respectively. We obtain correctness (functionality-preserving property), following from the correctness of the plain construction and functionality preserving properties of watermarkable $P_1, \cdots, P_k$.
4. In order to extract a mark, we turn any pirated algorithm A for P into pirated algorithm $A_1, \cdots, A_k$ for $P_1, \cdots, P_k$. We do this by applying the security reduction for P to the pirated algorithm A, and let $A_1, \cdots, A_k$ be the adversaries produced by the reduction. We then interpret $A_1, \cdots, A_k$ as pirated programs for $P_1, \cdots, P_k$, and attempt to extract the mark from each of them. Note that any mark extraction algorithm is supposed to use only the input-output behavior of the pirate program.
   The guarantee of the reduction is that one of these pirated programs A_i must actually be a "good" program in the security game for the corresponding primitive P_i, which means that mark extraction must succeed for that A_i. The security proof therefore shows that we are guaranteed to extract some mark [3].

Formalizing this composition takes some care, as we need to ensure that the mark extraction algorithm has the ability to actually transform A into each of the A_i. There are a couple issues with getting this to work:

- In a reduction, it is typically assumed that, when the reduction is attacking P_i, it has access to the keys for $P_j, j \neq i$. This access is often used to simulate the view of A when constructing A_i. In our watermarking construction, we therefore need to ensure that the tracing algorithm has this information.
- Security proofs often work via hybrid arguments that change various terms. Importantly, the hybrid arguments get to simulate the *entire* game. For us, however, we only get to simulate the second stage of the game; the first stage has actually already been fixed by the time the adversary produces its pirated program A. So we have to ensure that the tracing algorithm can carry out the reduction to create a program A_i, even though it cannot simulate the entire security experiment from the very beginning.

Our definition and proof take care of these issues and more, to give a composition theorem that applies any time our criteria are met.

Note that the restriction to games where the winning condition is independent of the first stage seems necessary. This is because the adversary actually gets in its possession a watermarked key, which lets it compute the various cryptographic functions of the key. For example, in the case of digital signatures, the adversary can actually compute signatures for itself. In the case of encryption, the adversary can encrypt and decrypt messages. Moreover, there is no way to track what the adversary is doing, since it is all happening internally to the adversary rather than being explicitly queried (suppose for example, that the adversary signs a few messages, but has its entire state encrypted under a fully homomorphic encryption (FHE) scheme so that the

[3] The Extract algorithm assumes the (input) adversarial circuit to be possibly stateful and interactive, but does not require any input circuit to be stateful and interactive. For example, an honestly generated watermarked circuit is not stateful or interactive in our construction.

signature and message being signed are all "under the hood" of the FHE scheme). We model this ability by having a first stage of the security experiment where the adversary can freely query the functionalities, but then do not have the win condition depend on those queries.

Watermarking Composition Example: CCA2-Secure SKE Now we demonstrate how the above abstract watermarking composition works by using the concrete example of a CCA2-secure SKE built from weak PRF and MAC again.

Consider having input constructions as watermarkable weak PRF and watermarkable MAC which satisfies our syntax and unremovability security requirements, a watermarking CCA-secure scheme consists of several algorithms: $\mathsf{WMSetup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Mark}, \mathsf{Extract}$. The $\mathsf{WMSetup}$ generates the secret key, marking key and extraction key by concatenating the corresponding keys from the PRF and MAC. The marking algorithm is on input $\mathsf{sk} = (\mathsf{sk}_{\mathsf{PRF}}, \mathsf{sk}_{\mathsf{MAC}})$ and watermarking message τ, output $\tilde{\mathsf{sk}} = (\tilde{\mathsf{sk}}_{\mathsf{PRF}} \leftarrow \mathsf{PRF.Mark}(\mathsf{sk}_{\mathsf{PRF}}, \tau), \tilde{\mathsf{sk}}_{\mathsf{MAC}} \leftarrow \mathsf{PRF.Mark}(\mathsf{sk}_{\mathsf{MAC}}, \tau))$.

Finally, the $\mathsf{Extract}$ algorithm is slightly more complex: on input extraction key $\mathsf{xk} = (\mathsf{xk}_{\mathsf{MAC}}, \mathsf{xk}_{\mathsf{PRF}})$ and a circuit C, do *both* of the following:

1. Create a circuit C_{PRF}:
   - Simulate the stage-2 CCA2-security for C (as described in our previous paragraph on watermarking-compatible reduction from CCA2 security to PRF). Note that just as a real reduction C_{PRF} is only hardcoded with the MAC extraction key $\mathsf{xk}_{\mathsf{MAC}}$ (unless the underlying watermarkable PRF scheme has a public extraction key), which enables it to simulate the signing and verification of MAC; to compute the part that involves the PRF evaluation oracle, C_{PRF} need to make external queries to some "challenger" to get answers.

     How C_{PRF} uses C to get its final output is exactly the same as in the watermarking-compatible reduction from CC SKE to weak PRF we described.

   After C_{PRF} is created, the extraction algorithm uses the watermarkable PRF's extraction algorithm to extract a mark from C_{PRF}: $\tau/\perp \leftarrow \mathsf{PRF.Extract}(\mathsf{xk}_{\mathsf{PRF}}, C_{\mathsf{PRF}})$. Note that the "external queries" made by the circuit will be answered now by the $\mathsf{PRF.Extract}(\mathsf{xk}_{\mathsf{PRF}}, \cdot)$ algorithm, because it treats C_{PRF} as a stage-2 adversary in the weak pseudorandomness security game and uses the extraction key $\mathsf{xk}_{\mathsf{PRF}}$ to simulate the game.
2. Create a circuit C_{MAC}:
   - Simulate the stage-2 CCA2-security for C (as described in our previous paragraph on watermarking-compatible reduction from CCA2 security to MAC). Note that just as a real reduction C_{MAC} is only hard-coded with the PRF extraction key $\mathsf{xk}_{\mathsf{PRF}}$, which enables it to simulate the oracles of PRF; to compute the part that involves the MAC signing/verifying oracles, C_{MAC} need to make external queries to some "challenger" to get answers.

     How C_{MAC} uses C to get its final output is exactly the same as in the watermarking-compatible reduction from CCA SKE to MAC we described.

   After C_{MAC} is created, the extraction algorithm will use the watermarkable MAC's extraction algorithm to extract a mark from C_{MAC}: $\tau/\perp \leftarrow \mathsf{MAC.Extract}(\mathsf{xk}_{\mathsf{MAC}}, C_{\mathsf{PRF}})$. Note that the "external queries" made by the circuit will be answered by the algorithm $\mathsf{MAC.Extract}(\mathsf{xk}_{\mathsf{MAC}}, \cdot)$ algorithm, because it treats C_{MAC} as a stage-2 adversary in the MAC security game and uses the extraction key $\mathsf{xk}_{\mathsf{MAC}}$ to simulate the game.

We say that the adversary $\mathcal{A}$ wins the watermarkable CCA2-secure SKE's unremovability game if no (previously queried) watermark can be extracted from either of the above circuits $C_{\mathsf{PRF}}, C_{\mathsf{MAC}}$ created during the extraction algorithm and C is a "good" program in winning the CCA2 security game. The intuition for the security proof is relatively straightforward: if C is a "good" program for winning CCA2-security game, then at least one of C_{PRF} and C_{MAC} must be a "good" program for winning its corresponding security game pf weak PRF or MAC, by the property of the reduction.

For instance, imagine a reduction algorithm $\mathcal{B}_{\mathsf{PRF}}$ to the unremovability of watermarkable PRF: $\mathcal{B}_{\mathsf{PRF}}$ will simulate the marking query stage for $\mathcal{A}$ by making marking queries to the marking oracle of the weak PRF challenger, along with the MAC key it sampled on its own. Note that the evaluation queries to the PRF evaluation oracle can now be replaced by having access to a marked PRF key. After $\mathcal{A}$ outputs circuit C, $\mathcal{B}_{\mathsf{PRF}}$ simply creates the same circuit C'_{PRF} as the C_{PRF} created in the above extraction algorithm. If the C_{PRF} circuit created in the extraction is "good" at winning the weak pseudorandomness game, then so is C'_{PRF}. Since we cannot extract a watermark from C_{PRF}, neither can we extract from C'_{PRF} because they have the same input-output behavior when using the same C as its subroutine black-boxly. The same argument applies to the reduction to unremovability of watermarkable MAC.

More Advanced Watermarking Constructions with "Unwatermarkable" Building Blocks The watermarking reduction in the above example of CCA2-secure SKE is relatively straightforward to acquire. Some similar construction examples include a watermarkable weak PRP from watermarkable weak PRF (see Appendix B).

However, there exist various constructions that look markedly distinct from the above example of CCA2-secue SKE and may be much more involved.

More importantly, some constructions have building blocks which are cryptographic primitives that don't make sense to watermark, for instance, a non-interactive zero knowledge proof scheme. Nevertheless, we show that a large class of them can be watermarking-compatible and in many scenarios, we simply do *not* need to watermark all the building blocks.

We briefly discuss two examples on a high level:

- **CCA-secure PKE from IBE and One-Time Signatures** The first example is the CCA2-secure PKE from selectively secure IBE and one-time signatures by [BCHK07].
  In the original scheme, the encryption samples a fresh signature signing/verification key pair $(\mathsf{sk}, \mathsf{vk})$, compute an IBE ciphertext by using the vk as the identity, then signs this ciphertext using signing key sk, and finally outputs the IBE ciphertext, the signature and vk.
  To decrypt, one first verifies the signature under vk, if valid then derives an identity-embedded decryption key from the IBE master secret key using vk as the identity and decrypt the ciphertext.
  Note that in this scheme, the keys for one-time signatures are only sampled "online" upon every encryption algorithm and how to watermark such keys is not well-defined. However, as it turns out—we don't have to watermark the signing key in our composed construction. A watermarkable CCA2-secure PKE using the above construction only has to watermark the IBE secret key. The security proof would show that a successful unremovability adversary can help us either break the unremovability security of the watermark IBE scheme or break the strong one-time unforgeability security of the signature scheme. In other words,

leveraging the "plain" security of the one-time signature scheme suffices for the composed watermarking scheme.
In more detail, we can show the following: the pirated algorithm C produced by an adversary is a "good" program for breaking the CCA-security and meanwhile no watermark can be extracted from C. However, it does not help us win the selective IBE CPA-security game. Then, simply following the hybrid argument in the security analysis of the CCA-secure PKE itself, we observe that C must be breaking the security of the one-time signature scheme. Thereby, we can use C black-boxly to create a reduction for the one-time signature scheme.

- **CCA-secure PKE from CPA-secure PKE and NIZK** A second example is the [NY90] CCA-secure PKE scheme built from CPA-secure PKE and a non-interactive zero-knowledge proof scheme (NIZK). The encryption algorithm encrypts a message twice under two different public keys and uses a NIZK proof to prove that they encrypt the same message.
The watermarkable version of the above construction does not watermark the NIZK scheme, but only the two decryption keys of the PKE scheme. How the NIZK scheme is used in the watermarking scheme is less obvious to see: intuitively, the extraction algorithm will try to create two circuits that break the underlying CPA security game of the PKE, with the corresponding keys. If we look carefully into the proof for the [NY90] scheme, to make this reduction go through, one has to first work in a hybrid game where the real NIZK proof in the challenge ciphertext is replaced with a simulated proof. Our extraction algorithm thereby uses such a simulated proof when interacting with the input circuit C. The security of NIZK helps us say that if C is "good" in the original CCA-security game, so will C be good in the game simulated by the extraction algorithm.

To summarize, we only have to watermark the secret keys of input primitives P_i which have their keys generated in the main Key Generation algorithm of the target primitive P and have their secret keys used during the evaluation algorithm of P. If an input primitive has its keys sampled "freshly" during every invocation of P's evaluation algorithms or sampled in key generation, but not used in P's evaluation algorithm (e.g. only used in the security proof instead), then we don't need a watermarkable version of P_i to build a watermarkable P. Leveraging P_i's original security property suffices.

More advanced examples include the functional encryption from attribute-based encryption, FHE and garbled circuits in [GKP$^+$13], where we only have to watermark the ABE scheme; a hybrid CCA-secure encryption scheme from CCA-secure PKE and CCA secure SKE, where we only have to watermark the PKE scheme. We refer the readers to the construction and discussions of these examples in Appendix A, 6, B,E, F.

2 Organizations

3 Definitions: General Cryptographic Primitive and Watermarking-Compatible Constructions

3.1 General Cryptographic Primitive

In this section, we present syntax and definitions for a general cryptographic primitive. The notations and definitions formalized in this section will assist the demonstration of our generic watermarking framework.

General Cryptographic Primitive Syntax A cryptographic primitive $P = (\mathsf{KeyGen}, \mathsf{SecEval}, \mathsf{PubEval})$ consists of the following algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$: is a (randomized) algorithm that takes a security parameter λ and interacts with an adversary $\mathcal{A}$: $(\mathsf{sk}, \mathsf{pk}) \leftarrow (\mathcal{A} \Leftrightarrow \mathsf{KeyGen}(1^\lambda))$ where sk is some secret information unknown to $\mathcal{A}$, and $\mathcal{A}$ can get some public information pk from the interaction.
- $\mathsf{SecEval}(\mathsf{sk}, \mathsf{pk}, x \in \mathcal{X}) \to y \in \mathcal{Y}_s$: a secret-evaluation algorithm that takes in the secret information sk, public information pk and some input x from input space $\mathcal{X}$, and output a value $y \in \mathcal{Y}_s$.
- $\mathsf{PubEval}(\mathsf{pk}, x \in \mathcal{X}) \to y \in \mathcal{Y}_p$: a public-evaluation algorithm that takes in the public information pk and some input x from input space $\mathcal{X}$, and output a value $y \in \mathcal{Y}_p$.

More generally, the KeyGen algorithm can generate several secret keys $\mathsf{sk}_1, \cdots, \mathsf{sk}_\ell$ for some polynomial ℓ. There will be ℓ different secret algorithms $\{\mathsf{SecEval}_i(\mathsf{sk}_i, \mathsf{pk}, x \in \mathcal{X}_{s,i})\}_{i\in[\ell]}$ and (some constant) m different public algorithms $\{\mathsf{PubEval}_j(\mathsf{pk}, x \in \mathcal{X}_{p,j})\}_{j\in[m]}$. For example, an identity-based encryption scheme can have two decryption algorithms, one using the master secret key, the other with a secret key embedded with an identity.

However, without loss of generality, we will make two simplifications:

- All algorithms have their input space padded to be the same length as $\mathcal{X}$;
- We view all secret algorithms $\{\mathsf{SecEval}_i(\mathsf{sk}_i, \mathsf{pk}, x \in \mathcal{X}_{s,i})\}_{i\in[\ell]}$ as one algorithm $\mathsf{SecEval}(\mathsf{sk}, \mathsf{pk}, \cdot)$ that will take in an index $i \in [\ell]$ to decide which mode to use, similarly for public algorithms. But we will assume such an index specification to be implicit and omit the use of indices in the algorithm to avoid an overflow of letters.

Occasionally, we denote all the algorithms $(\{\mathsf{SecEval}_i(\mathsf{sk}_i, \mathsf{pk}, x \in \mathcal{X}_{s,i})\}_{i\in[\ell]}, \{\mathsf{PubEval}_j(\mathsf{pk}, x \in \mathcal{X}_{p,j})\}_{j\in[m]})$ as a combined functionality hardcoded with the corresponding keys $\mathsf{Eval}(\mathsf{pk}, \mathsf{sk})$. We call it Eval for short.

Correctness for Predicate F_R Before going into the correctness property, we first define a notion important to our generic definition:

Definition 3.1 (Predicate). *A predicate $F(C, x, z_1, \cdots, z_k, r)$ is a binary outcome function that runs a program C on a some input x to get output y, and outputs 0/1 depending on whether $(x, y, z_1, \cdots, z_k, r) \in R$ for some binary relation defined by R. The randomness of input x, program C both depend on randomness r. . $z_1, , z_k$ are auxiliary inputs that specify the relation.*

A correctness property of a primitive P with respect to predicate F_R says that

Definition 3.2 (Correctness for Predicate F_R). *P satisfies correctness if there exists some function $\epsilon = \epsilon(\lambda) \in [0, 1]$ so that for all $\lambda \in \mathbb{N}, x \in \mathcal{X}$ [4]:*

$$\Pr_{r \leftarrow D_r, (\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)}[F_R(\mathsf{Eval}, \mathsf{pk}, \mathsf{sk}, x, r) = 1] \geq 1 - \epsilon.$$

[4]For a cryptographic primitive P, there can be many different correct properties, each defined with respect to a different predicate.

The randomness used in checking the predicate is sampled from a distribution D_r. We can simply take D_r to be the uniform distribution, which can be mapped to any distribution we need when computing the predicate.

All ϵ within the scope of this work is negligible in λ.

Remark 3.3. *To further explain the above correctness property, we consider the following (abstract) example. Given* $\mathsf{Eval} = (\{\mathsf{SecEval}_i(\mathsf{sk}_i, \mathsf{pk}, x \in \mathcal{X})\}_{i\in[\ell]}, \{\mathsf{PubEval}_j(\mathsf{pk}, x \in \mathcal{X}\}_{j\in[m]})$ *, F_R samples input $x \in \mathcal{X}$ using the first part of r; then it runs algorithm (supposing randomized, using part 2 of string r)* $\mathsf{SecEval}_1(\mathsf{sk}_1, \mathsf{pk}, x)$ *to give some outcome y_1; then runs (supposing deterministic)* $\mathsf{PubEval}_1(\mathsf{pk}_1, y_1)$ *to give outcome y_2; check if $(x, y_1, y_2) \in R$; output 1 if yes, 0 otherwise.*

A concrete example is a public key encryption scheme: the predicate is encrypting a message x using r and then decrypting it to check if one can recover the original message.

Game-based Security A security property of the cryptographic primitive P is described by a interactive procedure G_P between a challenger and adversary $\mathcal{A}$.

G_P will involve $\mathsf{KeyGen}, \mathsf{SecEval}, \mathsf{PubEval}$ as its subroutines. G_P outputs a bit 1 if the game is not aborted and a certain condition has been met at the end of the game; else it outputs 0.

The security of a primitive P with respect to G_P says: For all $\lambda \in \mathbb{N}$, there exists some function $\eta = \eta(\lambda)$ such that for all *admissible* $\mathcal{A}$, there exists a function $\mathsf{negl}(\lambda)$:

$$\Pr[G_P(1^\lambda, \mathcal{A}) = 1] \leq \eta + \mathsf{negl}(\lambda)$$

where η is the trivial probability for any admissible $\mathcal{A}$ to make G_P output 1 and the probability is over the randomness used in G_P.

2-Stage Game-based Security To be compatible with the context of watermarking, we will view the game G_P as a 2-stage security game. 2-stage game-based security is a central notion that connects a prmitive's "plain security" to its watermarking security.

The KeyGen procedure and a first part of the interactions between the challenger and $\mathcal{A}$ are taken out of the original game and executed as a first stage G_P^1. Then, G_P^2 denotes the rest of the game and will take in parameters $(\mathsf{sk}, \mathsf{pk})$ generated in stage 1 as well as some auxiliary input.

Definition 3.4 (2-Stage Game-based Security). *A security property of the cryptographic primitive P is described by a stage-1 (possibly interactive) key generation procedure a challenger and adversary $\mathcal{A}_1$ followed by a fixed-parameter game G_P^2 between a challenger and adversary $\mathcal{A}_2$.*

We denote $G_P^1(1^\lambda, \mathcal{A}_1)$ as the first stage adversary $\mathcal{A}_1$ interacting a challenger which runs the $\mathsf{KeyGen}(1^\lambda)$, $\mathsf{SecEval}, \mathsf{PubEval}$ *algorithm; together they output a key pair* $(\mathsf{sk}, \mathsf{pk})$ *and some auxiliary parameters* aux *which will be later used in the game G_P. $\mathcal{A}_1$ only gets* $\mathsf{pk}, \mathsf{aux}$ *but may make arbitrary polynomial number of admissible queries to oracles provided by the challenger during the interaction.*

The stage-2 game challenger G_P^2 is parametrized by inputs $\mathsf{sk}, \mathsf{pk}, \mathsf{aux}$ *generated during stage 1 and will involve* $\mathsf{SecEval}, \mathsf{PubEval}$ *as its subroutines. Stage-2 adversary $\mathcal{A}_2$ gets an arbitrary polynomial size state* st *from stage 1 $\mathcal{A}_1$. G_P^2 outputs a bit 1 if the game is not aborted and a certain condition has been met at the end of the game; else it outputs 0.*

The security of a primitive P with respect to G_P says: there exists some function $\eta = \eta(\lambda)$ such that for all admissible $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *and any non-negligible $\gamma = \gamma(\lambda)$, there exists a function* $\mathsf{negl}(\lambda)$ *for all $\lambda \in \mathbb{N}$:*

$$\Pr[\mathcal{A}_2(\mathsf{st}) \textit{ is } \gamma\textit{-good in } G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \cdot) : \{(\mathsf{sk}, \mathsf{pk}), \mathsf{aux}, \mathsf{st}\} \leftarrow G_P^1(\mathcal{A}_1, 1^\lambda)] \leq \mathsf{negl}(\lambda)$$

where $\mathcal{A}_2$ is said to be γ-good if:

$$\Pr[G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2) = 1] \geq \eta + \gamma$$

where η is the trivial probability for any admissible $\mathcal{A}$ to make G_P output 1 and the probability is over the randomness used in KeyGen *and by G_P.*

Remark 3.5. *While some readers may find the introduction of parameter γ in the above 2-stage game confounding, we would like to make a note that the above two definitions are essentially equivalent.*

An alternative way of stating the 2-stage game security would be: there exists some function $\eta = \eta(\lambda)$ such that for all admissible *$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists two negligible functions* $\mathsf{negl}_1(\lambda), \mathsf{negl}_2(\lambda)$ *such that for all $\lambda \in \mathbb{N}$:*

$$\Pr\left[\Pr[G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2(\mathsf{st})) = 1] \leq \eta + \mathsf{negl}_1(\lambda) : \{(\mathsf{sk}, \mathsf{pk}), \mathsf{aux}, \mathsf{st}\} \leftarrow G_P^1(\mathcal{A}_1, 1^\lambda)\right] \geq 1 - \mathsf{negl}_2(\lambda)$$

Remark 3.6 (Division into a 2-stage Game)**.** *How we divide a security game into two stages depends on the application and suppose that the game G_P has different stages by its definition, the way we divide stage-1 and 2 is usually not the same as in the original definition of G_P. We will see examples later.*

In most settings, stage-1 game G_P^1 only involves running the key generation KeyGen *and letting $\mathcal{A}_1$ make some queries. In a few special settings, $\mathcal{A}_1$ needs to commit to some challenge messages which will be put into* aux *and be part of the input to stage 2. Examples include the challenge attribute/identity in an attribute/identity-based encryption, because the queries made in the first stage need to go through the "admissibility" check that depends on $\mathcal{A}$'s choice of challenge messages.*

Remark 3.7. *In the rest of this work, it is usually clear from the context which stage of G_P we are refering to because G_P^2 will take in paramters* sk, pk, aux *while the entire game G_P takes in only security parameter 1^λ. Plus G_P^1 is seldomly used explicitly in our language. Occasionally we will omit the superscript and slightly abuse the notation to denote G_P^2 as $G_P(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \cdot)$*

For our convenience in later notations, we also give the following simple definition:

Definition 3.8 (Stage-2 Game View)**.** *The stage-2 G_P^2 can also output a view* $\mathsf{view}(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2)$ *that is the transcript of interaction of $\mathcal{A}_2$ with the challenger in G_P^2, including the final output.*

Remark 3.9. *We make a few notes on the scope of security games we consider in this work:*

1. *We mainly focus on game-based (and falsifiable) security notions within the scope of this work. We need the challenger in the security to be efficient for the watermarking construction to be efficient.*
2. *All the admissible adversary need to be PPT. We do not consider security models other than polynomially bounded in time/circuit size (such as bounded storage).*

3.2 Watermarking-Compatible Construction of Cryptographic Primitive

In this section, we characterize what type of black-box cryptographic constructions are watermarking compatible. As we will see in later sections, watermarking compatible constructions allow us to give a watermarking scheme for the target primitive constructed, black-boxly from the watermarking schemes of the building blocks.

Outline and Intuition Let use denote P as the target, or outcome primitive built in a construction. Let $P_i, i \in [k]$ denote each underlying primitive we use to build P.

Intuitively, one expects to watermark all underlying building blocks to ensure watermarking security of the target primitive P. As discussed in our technical overview section "More advanced watermarking constructions with unwatermarked building blocks", many constructions possess building blocks of primitives that do *not* need to be watermarkable to ensure the watermarkability of the final target primitive.

Here, we discuss the matter in more details: we need the partitioning for primitives in $\{P_i\}_{i\in[k]}$ into S and $\bar{S}$. These primitives in these two sets will play different roles when we construct the watermarking scheme for the target primitive P: the primitives in set S need to be watermarkable and those in set $\bar{S}$ do not need to be watermarkable. The reasoning is that the primitives in set S will have their secret keys generated during the Key Generation of the target primitive P, but primitives in set $\bar{S}$ will only have their secret keys generated freshly during every run of $\mathsf{SecEval}$ or $\mathsf{PubEval}$ algorithm of the target primitive P.

A simple example is CCA2-secure PKE scheme based on IBE and one-time signatures, which we discussed in the technical overview. During the encryption algorithm, one generates fresh one-time signature keys and the message is encrypted with the one-time signature's verification key as the identity. Therefore, the one-time signature in the above construction belongs to set $\bar{S}$ because its keys are only generated during the encryption algorithm.

In the watermarking construction for P, the $\mathsf{KeyGen}, \mathsf{PubEval}, \mathsf{SecEval}$ algorithms will follow from the plain construction for P from primitives in set S and $\bar{S}$. Therefore, we cannot watermark the keys for the primitives in set $\bar{S}$ because their keys are not generated when we give out the watermarked key (which only contains keys in set S) to a user. We therefore distinguish these two sets in our presentation for watermarking-compatible construction in this section and watermarkable implementation in Section 4.2.

Moreover, looking forward: we will observe that we indeed do *not* need watermarking security (i.e. unremovability) for the primitives in set $\bar{S}$, to achieve watermarking security for the target primitive P. We only need to rely on their "plain security" (e.g. unforgeability for a signature scheme, IND-CPA-security for an encryption scheme).

Notations Now we give formalization for the above outline.

Suppose a cryptographic primitive P is constructed black-boxly from primitives $P_1, P_2, \cdots, P_k$ in the following way, where $P_i, P_j, i \neq j$ are allowed to be the same primitive.

- Let $\mathcal{S}$ be some fixed subset of $[k]$ defined in the construction. $\mathcal{S}$ specifies two ways of using primitive P_i. The major difference is: for primitives $P_i, i \in \mathcal{S}$, the algorithm $\mathsf{KeyGen}(1^\lambda)$ will compute $P_i.\mathsf{KeyGen}(1^\lambda)$ and the secret keys $P_i.\mathsf{sk}$ generated will be used in the secret evaluation algorithm $\mathsf{wSecEval}$.
  Without loss of generality, we let the first $|\mathcal{S}|$ number of P_i's be those corresponding to the set $\mathcal{S}$.
- For primitives $P_i, i \notin \mathcal{S}$, the algorithm $\mathsf{KeyGen}(1^\lambda)$ will not compute $P_i.\mathsf{KeyGen}(1^\lambda)$. They will either (1) have their keys generated freshly upon every run of $\mathsf{SecEval}$ or $\mathsf{PubEval}$ (2) will have a reference string (which can be viewed as a public key) generated during $\mathsf{KeyGen}(1^\lambda)$, but have no secret keys or their secret keys will not be used in the $\mathsf{SecEval}$ algorithm of the target primitive P.

- For the sake of formality, we make a further division of the set $\bar{S}$ into T_S, T_P, T_K. They perform slightly different functionalities in the construction.
  - Let $\mathcal{T}_S \in [k]$ denote the set of indices i where P_i's keys will be generated during the secret evaluation algorithm $\mathsf{SecEval}$;
  - Let $\mathcal{T}_P \in [k]$ denote the set of indices i where P_i's keys will be generated during the public evaluaton algorithm $\mathsf{PubEval}$;
  - Let $\mathcal{T}_K$ denote the set of indices i where P_i have no secret keys/secret keys are not used in $\mathsf{SecEval}$ and will have a public reference string (which can be viewed as a public key) generated during $\mathsf{KeyGen}(1^\lambda)$.
    [5].

Watermarking-Compatible Construction Syntax

$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$:

1. compute $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow P_i.\mathsf{KeyGen}(1^\lambda)$ for all $i \in \mathcal{S}$; compute $(\mathsf{pk}_i, \mathsf{td}_i) \leftarrow P_i.\mathsf{KeyGen}(1^\lambda)$ for all $i \in \mathcal{T}_K$.
2. output $\mathsf{sk} = (\{\mathsf{sk}_i\}_{i \in \mathcal{S}})$; $\mathsf{pk} = (\{\mathsf{pk}_i\}_{i \in \mathcal{S} \cup \mathcal{T}_K})$.

$\mathsf{SecEval}(\mathsf{sk}, \mathsf{pk}, x \in \mathcal{X}) \to y \in \mathcal{Y}_s$: is an algorithm that

1. uses $P_i.\mathsf{SecEval}(\mathsf{sk}_i, \mathsf{pk}_i, \cdot), i \in \mathcal{S}$ as subroutines.
2. uses $P_i.\mathsf{PubEval}(\mathsf{pk}_i, \cdot), i \in \mathcal{S} \cup \mathcal{T}_K$ as subroutines.
3. computes $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow P_j.\mathsf{KeyGen}(1^\lambda)$ for some $j \in \mathcal{T}_S$ (can include these pk_j generated as part of the output).

$\mathsf{PubEval}(\mathsf{pk}, x \in \mathcal{X}) \to y \in \mathcal{Y}_p$: is an algorithm that

1. uses $P_i.\mathsf{PubEval}(\mathsf{pk}_i, \cdot), i \in \mathcal{S} \cup \mathcal{T}_K$ as subroutines.
2. computes $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow P_j.\mathsf{KeyGen}(1^\lambda)$ for all $j \in \mathcal{T}_P$ (may include these $\{\mathsf{pk}_j\}_j$ generated as part of the output).

We say the construction is watermarking compatible if the above construction of P satisfies:

1. **Correctness of Construction** : the above construction of P satisfies a correctness property defined by predicate F_R. Moreover, the proof of this correctness property follows from correctness properties of $P_1, \cdots, P_k$ for predicates F_{R_i} respectively.
2. **Security of Construction**: The above construction satisfies security defined with game G_P. The security can be proved from the security properties of $P_1, \cdots, P_k$ with respect to some game $G_{P_1}, \cdots, G_{P_k}$
3. **Watermarking Compatible Reduction** The above security proof is of the following format, which we call *Watermarking-Compatible Reduction*, shown below.

Among the above 3 properties, the first two are natural properties that come with any black-box cryptographic constructions with provable security and correctness. We will focus on discussing property 3.

[5]As we will see, primitives in $\mathcal{T}_k$ will have their secret keys (called trapdoors td here) used only in the security proofs.

Looking forward: the sets $\mathcal{T}_S, \mathcal{T}_P$ will play the same role in the watermarking reduction, and the set $\mathcal{T}_K$ will incur a slightly different argument but will essentially play the same role as the other primitives in set $\bar{S}$. That is, only their plain security is required to achieve the watermarking security of the target primitive P.

Watermarking Compatible Reductions On a high-level, a watermarking-compatible reduction is essentially a natural security reduction, except with the following feature: we view both the the security game G_P for the target security primitive P and the security game G_{P_i} for the input primitive P_i as two-stage games defined in Definition 3.4. The supposed adversary $\mathcal{A}$ for G_P will just be any usual PPT adversary. But we restrict the reduction in stage 2 to be "oblivious" about the queries made by $\mathcal{A}$ in stage 1: that is, it cannot pass on any queries made by $\mathcal{A}$ to the stage 2 reduction but we should nevertheless make the reduction go through successfully.

We divide our watermarking compatible reductions into two types. Type 1 reduction captures the reduction from breaking the security of P to breaking the security of a primitive $P_i, i \in \mathcal{S}$, i.e. the primitives that have their secret keys generated in the KeyGen algorithm and used in the $\mathsf{SecEval}$ algorithm of P.

Type 2 reduction accordingly captures the reduction from breaking the security of P to breaking the security of a primitive $P_i, i \in \bar{\mathcal{S}}$.

The main difference is in how they are used in proving the unremovability security in the composed watermarking scheme. In the actual watermarking unremovability, type 1 reduction is supposed to be oblivious about the queries made by the adversary and will eventually lead to breaking the unremovability of the underlying primitive; type 2 reduction operates similarly but will eventually lead to breaking the plain security of the underlying primitive.

Watermarking-Compatible Reduction Type 1 First we consider reductions for primitives P_i for $i \in \mathcal{S}$.

1. Consider a reduction to P_i for the j-th $P_i \in \mathcal{S}$; let $\mathcal{A}_i = (\mathcal{A}_i^1, \mathcal{A}_i^2)$ be the two-stage adversary for the 2-stage game of primitive P_i defined in Definition 3.4.
2. $\mathcal{A}_i$ receives public key pk_i
3. $\mathcal{A}_i$ prepares $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathsf{KeyGen}_j(1^\lambda)$ for all $j \neq i, j \in \mathcal{S}$ and $\mathcal{A}_i$ sends all $(\{\mathsf{pk}_j\}_{j\in\mathcal{S}})$ to $\mathcal{A}_1$.
4. $\mathcal{A}_i$ simulates the security game G_P for P with $\mathcal{A}$, while being the adversary in game G_{P_i}.
   - In stage 1, $\mathcal{A}_i^1$ provides the oracles needed for game G_P^1 and answer $\mathcal{A}^1$'s queries as follows:
     - For any (admissible) queries in the game G_P^1, if sk_i is required to compute the answer for the query, $\mathcal{A}_i$ can use the oracles provided in game $G_{P_i}^1$. To answer the entire query, $\mathcal{A}_i$ may finish the rest of the computation using $\{\mathsf{sk}_j\}_{j\neq i}$.
     - If only $\{\mathsf{sk}_j\}_{j\neq i}$ are needed, $\mathcal{A}_i$ answers the queries by itself.
   - During the interaction of stage-1 game, $\mathcal{A}^1$ and $\mathcal{A}_i^1$ generates an auxiliary information aux which is public to both of them.
   - Upon entering stage 2, $\mathcal{A}^1$ generates an arbitrarily polynomial-size state st and gives it to $\mathcal{A}^2$.
   - $\mathcal{A}_i$ also enters its second stage $\mathcal{A}_i^2$. $\mathcal{A}_i^2$ also receives all of $(\{\mathsf{sk}_j\}_{j\neq i}, \{\mathsf{pk}\}_{i\in\mathcal{S}}, \mathsf{aux})$ from But $\mathcal{A}_i^2$ does *not* obtain any of $\mathcal{A}^1$'s queries in stage 1.
   - $\mathcal{A}_i^2$ simulates the stage-2 game G_P^2 for $\mathcal{A}^2$, using the above strategy as A_i^1 uses. Note that the oracle operations done by $\mathcal{A}_i^2$ and the conditions on admissible queries *cannot* be dependent on of $\mathcal{A}^1$'s queries in stage 1, because $\mathcal{A}_i^2$ cannot see these queries.
   - $\mathcal{A}_i^2$ also records all of $\mathcal{A}^2$'s queries.
   - In the challenge phase of $G_{P_i}^2$ $\mathcal{A}_i^2$ receives a challenge input inp_i from the challenger.

- In G_P's challenge phase, $\mathcal{A}_i^2$ samples some randomness r and prepares a challenge input inp for $\mathcal{A}$ using r and inp_i. $\mathcal{A}_i^2$ sends inp_i to $\mathcal{A}_2$. [6]
- $\mathcal{A}_i^2$ continues to simulate the oracles needed in G_P^2 game for $\mathcal{A}^2$ if there are further query stages.

5. $\mathcal{A}_i$ computes an efficiently computable function $f_i(\mathsf{out}, r, \mathcal{Q}, \mathsf{inp}_i)$ on input of $\mathcal{A}^2$'s final answer out, the randomness used to prepare $\mathcal{A}^2$'s challenge input, $\mathcal{A}^2$'s queries $\mathcal{Q}$ and A_i's challenge input inp_i, and gives it to the challenger of primitive P_i.

Remark 3.10. *In the actual watermarking unremovability reduction, $\mathcal{A}_i^1$ will receive a watermarked secret key $\tilde{\mathsf{sk}_i}$ and simulate the queries required using the oracles in game $G_{P_i}^1$ with $\tilde{\mathsf{sk}_i}$ instead.*

More generically, we can also model the queries made by $\mathcal{A}^1$ in stage 1, related to the keys in the set $\mathcal{S}$, $\{\mathsf{sk}_i\}_{i\in\mathcal{S}}$ as some arbitrary polynomial size (admissible) leakage on these keys. In the plain security reduction, such leakage correspond to $\mathcal{A}^1$'s adaptive queries made to the oracles provided. In the watermarking unremovability game, it corresponds to admissible marking queries where we give out marked secret keys.

Watermarking-Compatible Reduction Type 2 Watermarking-compatible reduction type 2 operates essentially the same as the above reduction for P_i but for some $i \notin \mathcal{S}$.

The main difference is that in the actual watermarking unremovability security game, in order to simulate oracle queries for G_P^1, the stage 1 reduction $\mathcal{A}_i^1$ will actually need oracles in the game $G_{P_i}^1$. Naturally, this is because the primitives in set $\bar{\mathcal{S}}$ are not watermarked in the watermarking composition and the corresponding reduction will not get a watermarked key from the challenger, but only the oracles provided in the plain security game of G_{P_i}. Even though this also means that the reduction A_i^1 gets to observe $\mathcal{A}^1$'s queries (using the oracles of the game G_{P_i}) in the actual watermarking security game, and can make use of them in stage 2, A_i^1 does not need to make any of such queries to the challenger since the keys of any primitive in set $\bar{\mathcal{S}}$ will only be sampled online by A_i^1 itself. Meanwhile, $\mathcal{A}_i^2$ should still not inherit any queries related to the "watermarked" primitives in set $\mathcal{S}$ since in the real watermarking unremovability game, $\mathcal{A}_i^2$ is not supposed to see them.

1. Consider doing reduction to P_i for some $P_i, i \notin \mathcal{S}$; let $\mathcal{A}_i = (\mathcal{A}_i^1, \mathcal{A}_i^2)$ be the two-stage adversary for primitive P_i, where the stages are defined by Definition 3.4.
2. $\mathcal{A}_i$ receives public key pk_i from the challenger.
3. $\mathcal{A}_i$ prepares $(\mathsf{sk}_\ell, \mathsf{pk}_\ell) \leftarrow P_\ell.\mathsf{KeyGen}(1^\lambda)$ for all $\ell \in \mathcal{S}$ $\mathcal{A}_i$ sends all $(\{\mathsf{pk}_\ell)$ to $\mathcal{A}_1$.
4. $\mathcal{A}_i$ simulates the security game G_P for P with $\mathcal{A}$:
   - In stage-1 game G_P^1, for any (admissible) queries if $\mathsf{sk}_\ell, \ell \in \mathcal{S}$ is required to compute the answer for the query, $\mathcal{A}_i$ can answer the query since it possesses the keys $\{\mathsf{sk}_\ell\}_{\ell\in\mathcal{S}}$.
     The secret keys of primitives $\{P_j\}_{j\notin\mathcal{S}}$ are all sampled freshly upon every run of $\mathsf{wSecEval}(\mathsf{sk}, \mathsf{pk}, \cdot)$ (or sampled freshly in $\mathsf{wPubEval}(\mathsf{pk}, \cdot)$), which $\mathcal{A}_i^1$ can do it on its own.
   - During the interaction of stage-1 game, $\mathcal{A}^1$ and $\mathcal{A}_i^1$ generates an auxiliary information aux which is public to both of them.
   - Entering stage 2, $\mathcal{A}^1$ generates an arbitrarily polynomial-size state st and gives it to $\mathcal{A}^2$. $\mathcal{A}_i$ also enters its second stage $\mathcal{A}_i^2$. $\mathcal{A}_i^2$ also receives all of $(\{\mathsf{sk}_\ell\}_{\ell\in\mathcal{S}}, \{\mathsf{pk}_\ell\}_{\ell\in\mathcal{S}}, \mathsf{aux})$ but does *not* obtain any of $\mathcal{A}^1$'s queries involving the use of $\{\mathsf{sk}_\ell\}_{\ell\notin\mathcal{S}}$.

[6]Note that G_P's challenge phase may be before, after or concurrent with G_{P_i}'s challenge phase depending on the reduction. Similarly, the challenge $\mathcal{A}^2$ receives may be dependent on inp.

- $\mathcal{A}_i^2$ continues to provide the oracles needed for game G_P^2 and answer $\mathcal{A}^2(\mathsf{st})$'s queries using the above strategy as A_i^1 uses.
- $\mathcal{A}_i^2$ also records all of $\mathcal{A}^2$'s queries.
- In the challenge phase of G_{P_i}, $\mathcal{A}_i^2$ receives a challenge input inp_i from the challenger.
- In G_P^2's challenge phase, $\mathcal{A}_i^2$ samples some randomness r and prepares a challenge input inp for $\mathcal{A}$ using r and inp_i. $\mathcal{A}_i^2$ sends inp_i to $\mathcal{A}_2$.
- $\mathcal{A}_i^2$ continues to simulate the oracles needed in G_P^2 game for $\mathcal{A}^2$ if there are further query stages

5. $\mathcal{A}_i$ computes an efficiently computable function $f_i(\mathsf{out}, r, \mathcal{Q}, \mathsf{inp}_i)$ on input of $\mathcal{A}^2$'s final answer out, its queries $\mathcal{Q}$ and A_i's challenge input inp_i and secrret radomness r, and gives it to the challenger of primitive P_i.

Properties of Watermarking-Compatible Reductions We further give some properties of watermarking-compatible reduction. These properties come with any natural black-box security roof with hybrid argument and reductions. We present them here for the sake of convenience and use them as facts later.

Fact 3.11 (Reduction Property 1). *A watermarking-compatible reduction guarantees that: if there exists some $\mathcal{A}$ such that the advantage of $\mathcal{A}^2$ winning the game G_P is non-negligible, i.e.* $\Pr[G_P(1^\lambda, A) = 1] \geq \eta + \gamma$ *where η is the trivial winning probability and γ is non-negligible in λ, then there exists some $i \in [k]$ such that $\mathcal{A}_i$, using the above reduction strategy, wins G_{P_i} with some non-negligible advantage γ_i.*

Remark 3.12. *The above property follows naturally from any hybrid argument of proof.*

If using the language of the 2-stage security game: If there exists some adversary $\mathcal{A} = (\mathcal{A}^1, \mathcal{A}^2)$ such that for some non-negligible ϵ and some non-negligible γ, we have:

$$\Pr\left[\Pr[G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2(\mathsf{st})) = 1] \geq \eta + \gamma : \{(\mathsf{sk}, \mathsf{pk}), \mathsf{aux}, \mathsf{st}\} \leftarrow G_P^1(\mathcal{A}_1, 1^\lambda)\right] \geq \epsilon$$

Then there exists some $i \in [k]$ such that $\mathcal{A}_i = (\mathcal{A}_i^1, \mathcal{A}_i^2)$, using the above reduction strategy, such that for some non-negligible γ_i, ϵ_i, we have:

$$\Pr\left[\Pr[G_{P_i}^2(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{aux}, \mathcal{A}_i^2(\mathsf{st})) = 1] \geq \eta + \gamma_i : \{(\mathsf{sk}_i, \mathsf{pk}_i), \mathsf{aux}, \mathsf{st}\} \leftarrow G_{P_i}^1(\mathcal{A}_i^1, 1^\lambda)\right] \geq \epsilon_i.$$

Fact 3.13 (Reduction Property 2). *For all $i \in \mathcal{S}$, once give the secret key $P_i.\mathsf{sk}$ in the clear, there exists a PPT algorithm T_i that wins the security game G_P with probability 1.*

Additionally, such a T_i can be used black-boxly by the watermarking-compatible reduction algorithm $\mathcal{A}_i$ to win the security game G_{P_i} with noticeable probability.

Reduction Function Recall that in the end of the reduction, $\mathcal{A}_i$ computes an efficiently computable function $f_i(\mathsf{out}, r, \mathcal{Q}, \mathsf{inp}_i)$ on input of $\mathcal{A}^2$'s final answer out, the randomness used to prepare $\mathcal{A}^2$'s challenge input, $\mathcal{A}^2$'s queries $\mathcal{Q}$ and A_i's challenge input inp_i, and gives it to the challenger of primitive P_i.

For convenience, we will refer to the function f_i used by the reduction as *reduction function*.

Remark 3.14 (Reduction Function). *We refer to the above function f_i used by the reduction as* reduction function.

Remark 3.15 (Special Primitives in $\mathcal{T}_K$). *The role of the primitives in the set $\mathcal{T}_K$ may be confusing to the readers at this point. We make some further explanation. As we go into later sections and go into examples (Appendix A), its role will become clear.*

More specifically, $\mathcal{T}_k$ only contains the simulation-based primitives P_ℓ where

1. *there exists an efficient simulator algorithm such that the output of $P_\ell.\mathsf{PubEval}(P_\ell.\mathsf{pk}, \cdot)$ is indistinguishable from the output of the simulator $\mathsf{Sim}(P_\ell.\mathsf{td}, \cdot)$. Their secret key (trapdoor td) will only come up in the security proof for P.*
2. *In the security proof for P, the rest of the hybrid arguments and reduction happen in a hybrid world where we have already invoked the above security for P_ℓ.*

The one example primitive in the set $\mathcal{T}_K$ we use within this work is a NIZK scheme with a trapdoor. Its "keys" (the common reference string CRS, and trapdoor td) are generated in the main key generation algorithm, but the trapdoor is not used in any of $\mathsf{SecEval}$, $\mathsf{PubEval}$, and only in the security proof.

4 Watermarking Composition Framework

4.1 Definition: Watermarkable Implementation of a Cryptographic Primitive

In this section, we will define what we call a "watermarkable implementation" of a cryptographic primitive P. We distinguish it from the usual naming ("watermarkable P") because of some differences in syntax and definition. But we will sometimes use watermarkable P in our work for convenience. Please note that all "watermarkable P" used in the technical part of this work refers to a watermarkable implementation of a cryptographic primitive P defined below.

On a high-level, a watermarkable implementation of P differs from most existing watermarking definition in two aspects:

1. Unremovability security: A pirate circuit produced by the adversary in the unremovability security game is considered to function successfully as long as it can break the security game G_P.
2. Extraction algorithm syntax: the extraction key used to extract a watermark is able to simulate the game G_P for the underlying "plain security" of P.

In more detail, a watermarkable implementation of a cryptographic primitive has the following syntax and properties.

Watermarkable Primitive Syntax A watermarkable primitive WP for a cryptographic primitive P with a security game G_P consists of the following algorithms:

$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk}, \mathsf{mk}, \mathsf{xk})$: on security parameter, outputs a secret key sk, public key pk, marking key mk, extraction key xk.

$\mathsf{wSecEval}(\mathsf{sk}, \mathsf{pk}, x)$: takes in the secret information sk, public information pk and some input x from input space $\mathcal{X}$, and output a value $y \in \mathcal{Y}_s$.

$\mathsf{wPubEval}(\mathsf{pk}, x \in \mathcal{X})$: takes in the public information pk and some input x from input space $\mathcal{X}$, and output a value $y \in \mathcal{Y}_p$.

$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau \in \mathcal{M}_{\mathsf{Mark}}) \to \mathsf{sk}_\tau$: takes in marking key mk, a secret key sk and a message $\tau \in \mathcal{M}_{\mathsf{Mark}}$; output a marked key sk_τ[7].

$\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C) \to \tau \in \mathcal{M}_{\mathsf{Mark}}/\vec{\tau} \in \mathcal{M}^k_{\mathsf{Mark}}/\bot$: on input extraction key sk, public key pk, auxiliary information aux and circuit C, outputs a mark message $\tau \in \mathcal{M}_\tau$ or a tuple of marked messages $\vec{\tau} \in \mathcal{M}^k_\tau$ where k is a constant/small polynomial parameter related to the concrete watermarking construction.

Remark 4.1. *The* Extract *algorithm is allowed to output a tuple of marks when working on adversarial programs, when the watermarkable implementation is one that comes from composing underlying watermarking implementations. More details to be discussed later in Section 4.2.*

Remark 4.2. *The extraction algorithm takes in the public key* pk *(which is not a limitation because it's public) and an auxiliary input* aux. *As we will see in the concrete examples, depending on the security game of the primitives we watermark, we may need* Extract *to take in some* aux *or may not.*

When it is clear from the context that there is no public key or auxiliary information for Extract *to take, we will omit them from the input parameters for notation cleanness.*

Remark 4.3. *There can be several different (constant number of)* wSecEval *and* wPubEval. *As aforementioned, we view all secret algorithms as one algorithm* $\mathsf{SecEval}(\mathsf{sk}, \mathsf{pk}, \cdot)$ *that will take in an index $i \in [\ell]$ to decide which mode to use, similarly for public algorithms. We thus also use* sk *to denote* $(\{\mathsf{sk}_i\}_{i\in[\ell]}$.

Let us denote $\mathsf{Eval} = (\mathsf{wSecEval}, \mathsf{wPubEval})$.

Correctness The construction should satisfy "unmarked" correctness for unmarked keys: a watermarkable implementation of P is said to be correct with respect to predicate F_R if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}, x \in \mathcal{X}$:

$$\Pr[F_R(\mathsf{Eval}, \mathsf{pk}, \mathsf{sk}, x, r) = 1 : (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{WMSetup}(1^\lambda), r \leftarrow D_r] \geq 1 - \mathsf{negl}(\lambda).$$

where the probability is taken over randomness used in WMSetup and D_r. Recall that randomness r is used in checking whether the predicate $F_R(\mathsf{Eval}, \mathsf{pk}, \mathsf{sk}, x, r)$ is satisfied Definition 3.1 and D_r can be simply taken to be the uniform distribution.

Functionality Preserving vs. Exact Functionality Preserving We present both definitions of functionality-preserving for the sake of comprehensiveness because they have both appeared in watermarking literatures. Before [GKM+19], the watermarking literature mainly used exact functionality preserving only. While it is not particularly vital to the contributions in this work, we would like to mention that exact functionality preserving is a stronger notion. For some of the underlying building blocks we use, only constructions under the relaxed functionality-preserving definition are known, for example the watermarkable signature scheme in [GKM+19].

[7] In defintions in the watermarking literature, the output of Mark is the program $\mathsf{wSecEval}(\mathsf{sk}_\tau, \mathsf{pk}, \cdot)$. Since conventionally by Kerkhoff's principle, the evaluation algorithm itself is public, giving out only the marked secret key is equivalent to giving out the program, we give out the key for convenience that come up later. There are watermarking schemes where the evaluation algorithm may be different when running on a marked key; in that case we can consider wSecEval to have two modes, one for unmarked keys one for marked keys.

Functionality Preserving The functionality-preserving property says: if for a predicate F_R, the underlying primitive P satisfies he correctness in Definition 3.2, then there exists a negligible function $\mathsf{negl}(\lambda)$ ssuch that for all $\lambda \in \mathbb{N}, x \in \mathcal{X}$:

$$\Pr\left[F_R(\mathsf{Eval}, \mathsf{pk}, \tilde{\mathsf{sk}}, x, r) = 1 : \begin{array}{c} (\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ \widetilde{\mathsf{sk}} \leftarrow \mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau), r \leftarrow D_r \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Exact Functionality Preserving The exact functionality preserving is a stronger property that: there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}, x \in \mathcal{X}, \tau \in \mathcal{M}_\tau$:

$$\Pr\left[\mathsf{wSecEval}(\tilde{\mathsf{sk}}, \mathsf{pk}, x) = \mathsf{SecEval}(\mathsf{sk}, \mathsf{pk}, x) : \begin{array}{c} (\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ \widetilde{\mathsf{sk}} \leftarrow \mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Correctness of Extraction There exists a publicly known efficient algorithm T and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$, for all $\tau \in \mathcal{M}_\tau$:

$$\Pr\left[\mathsf{Extract}(\mathsf{xk}, T(\mathsf{sk}_\tau, \cdot)) = \tau \ : \begin{array}{c} (\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ \widetilde{\mathsf{sk}} \leftarrow \mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Security with Security Game G_P The watermarkable primitive WP satisfies the same security defined by the security game G_P for P, except that the subroutines used in G_P, $\mathsf{KeyGen}, \mathsf{SecEval}, \mathsf{PubEval}$ are replaced with $\mathsf{WMSetup}, \mathsf{wSecEval}, \mathsf{wPubEval}$ respctively. The $\mathsf{Extract}, \mathsf{Mark}$ algorithms and keys are ignored in the context of these games.

Watermarking Security Compatible with A Security Game Now we present the unremovability security of the watermarking implementation for primitive P.

Informally, the security guarantees that: any PPT adversary $\mathcal{A}$, when given the watermarked key(s), generates a pirate circuit C^*; if C^* can win the security game G_P with some non-negligible advantage, then we must be able to extract a (previously queried) watermark from C^*.

Definition 4.4 (γ-Unremovability with Game G_P)**.** *We say a watermarkable implementation of P is γ-unremovable if:*

For every stateful γ-unremovable admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, the following holds:*

$$\Pr\left[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C^*) \notin \mathcal{Q} : \begin{array}{c} (\mathsf{sk}, \mathsf{pk}, \mathsf{mk}, \mathsf{xk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ (\mathsf{aux}, C^*) \leftarrow \mathcal{A}^{\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)}(1^\lambda, \mathsf{pk}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

$\mathcal{Q}$ denotes the set of f marks queried by $\mathcal{A}$ to the marking oracle $\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)$*, G_P^1 is the stage 1 of a security game G_P, as defined in Definition 3.4.*

We call a PPT adversary $\mathcal{A}$ as γ-unremovable admissible if $\mathcal{A}$'s output C^* is an admissible adversary in the stage-2 *security game G_P^2 and is γ-good, where γ-good is defined as:*

$$\Pr[G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, C^*) = 1] \geq \eta + \gamma$$

Here, η is the trivial success probability for any admissible adversary in G_P.

The randomness over testing whether C^* is γ-good is the randomness used answering any oracle queries from C^* and preparing the challenge for C^*, in the stage-2 security game G_P^2.

To match our syntax, we denote $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C^*) \notin \mathcal{Q}$ *to mean that:*

1. *If* $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C^*)$ *outputs a single mark* $\tau \in \mathcal{M}_\tau / \tau = \bot$*, it is considered to be not in the query set* $\mathcal{Q}$ *if and only if* $\tau \notin \mathcal{Q}$*.*
2. *If* $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C^*)$ *outputs a tuple of marks* $\vec{\tau} = (\tau_1, \cdots, \tau_k)$*, where each* $\tau_i \in \mathcal{M}_\tau / \tau_i = \bot$*, it is considered not in the query set* $\mathcal{Q}$ *if and only if for all* i*,* $\tau_i \notin \mathcal{Q}$*.*

Remark 4.5. *The* Extract *algorithm will only output a tuple of marks when used on adversarial circuits* C*. Because of composition, the adversary might be able to acquire a composed watermarked circuit where each "component" has a different watermark, but this does not formulate an attack as long as we can still extract any valid (previously queried) watermarks from the circuit. We will discuss this aspect in more details in Section 4.2.*

Remark 4.6 (Discussions on the γ-Unremovability Definition)**.** *We give the above definition with a parameter* γ *and it helps illustrate how this definition corresponds to the 2-stage security game in Definition 3.4:* $\mathcal{A}$ *corresponds to the "stage-1" adversary and the circuit* C^* *corresponds to stage-2 adversary.*

If we want a watermarking scheme's γ *to be any non-negligible function (the strongest notion here) in terms of the security parameter, we can simply require it to satisfy* γ*-unremovability for any non-negligible* γ *(defined below as "strong unremovability").*

In our paper, all our underlying watermarking building blocks used for composition satisfy γ*-unremovability for any non-negligible gamma. So they are* not *sensitive to* γ*. Our statements about the watermarking compiler also only consider this case (see Lemma 4.21).*

Besides, we also inherit this γ*-removability definition partially from [GKM*$^+$*19]. In some other watermarking literatures, it may be also meaningful to consider a weaker security where gamma is not necessarily negligible.*

Definition 4.7 (Strong Unremovability)**.** *We say a watermarking implementation of* P *satisfies strong unremovability if it satisfies* γ*-unremovability for any non-negligible* γ*.*

Remark 4.8 (Statefulness of the Adversarial Program)**.** *Since the adversarial circuit in our security game is assumed to be interactive and stateful, one may ask how stateful it can be.*

This issue with stateful pirates is almost always present in the traitor tracing and watermarking literature. For this reason, the vast majority of works operate in a stateless program model. This requires an assumption that the pirate program can be reset by the tracer to its initial conditions, which makes sense if, say, the program is given as software. In our work, we allow the pirate program to be stateful, in the sense that we allow it to play an interactive security game. However, we always assume between runs of the game that the program is reset to its initial conditions. This notion of statefulness generalizes the existing watermarking/traitor tracing definitions from non-interactive games to interactive games, while also avoiding the issue of self-destruction. Like in the existing models, our model makes sense if, say, the program is given as software.

We leave more remark on the unremovability definition at the end of this section Remark 4.12.

Security Game Simulation Property of the Extraction Key We additionally require the extraction key Extract to be able to simulate the (stage-2) security game G_P^2 for the primitive P to be watermarked.

Consider the security game G_P for the underlying primitive P: $G_P = (G_P^1, G_P^2)$ is an interactive game between a challenger and admissible $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Recall that in the stage-2 Definition 3.4 G_P^2 can output a view $\mathsf{view}(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2)$ (Definition 3.8).

Definition 4.9 (Extraction key simulation property). *The extraction key simulation property of the extraction key says that: given* $(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}) \leftarrow G_P^1(\mathcal{A}_1, 1^\lambda)$ *where we use* $\mathsf{WMSetup}(1^\lambda)$ *to obtain* $(\mathsf{sk}, \mathsf{pk}, \mathsf{xk})$ *in* G_P^1*, there exists a PPT simulator* $\mathsf{Sim}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2)$ *(where* Sim *interacts with* $\mathcal{A}_2$ *black-boxly) that outputs a simulated view* $\mathsf{view}_{\mathsf{Sim}}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2)$*, such that for any* $\lambda \in \mathbb{N}$*, for any admissible* $\mathcal{A}$ *in* G_P*, the distributions* $\mathsf{view}_{G_P^2} = \{\mathsf{view}(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2)\}$ *and* $\mathsf{view}_{\mathsf{Sim}} = \{\mathsf{view}_{\mathsf{Sim}}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2)\}$ *are perfectly/statistically/computationally indistinguishable.*

Extraction Syntax: Simulation of Security Game G_P inside $\mathsf{Extract}$ Following the above property, we require that the Extract algorithm in a watermarkable implementation of a primitive P follows a specific format: it simulates the stage-2 security game G_P^2 for any input circuit, while trying to extract a mark.

$\mathsf{Extract}$ **Algorithm**

On input $(\mathsf{xk}, \mathsf{pk}, \mathsf{aux})$ and a program C:

- Run an algorithm E where E uses the following subroutine (for possibly $\mathsf{poly}(\lambda)$) many times):
  - Use $(\mathsf{xk}, \mathsf{pk}, \mathsf{aux})$ to simulate the stage-2 game $G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, C)$ for primitive P, running C black-boxly by treating C as the stage-2 adversary $\mathcal{A}_2$ defined in Definition 3.4.
  - If C is non-interactive (i.e. C does not make any queries or respond to interaction in the challenge phase of G_P^2), then E samples challenge inputs on its own and runs C on them.
- E can take any of C's outputs, including C's queries made during the above simulated game, as inputs to compute the extraction algorithm's final output.
- Output a watermark $\tau \in \mathcal{M}_\tau$ or $\perp$

Examples of Extraction Algorithm that Simulates a Security Game It is not hard to create an extraction algorithm with the above syntax and simulation capability. In fact, some existing works have already built watermarking schemes that have Extract algorithm with such a format.

- To enable the extraction key to simulate the security game, a naive solution in the private extraction case is to simply let xk contain the secret key sk.
  One example is the extraction algorithm in a watermarkable signature scheme (see Appendix D). To answer the signature queries for the pirate program, the extraction key is the signing key.
- In some other scenarios, we don't need the secret key to simulate the security game and one can even have public extraction.
  For example, a CPA secure PKE scheme. Another example is when we can sample from the input-output space of the evaluation oracles without having the actual key: in the weak PRF setting (with non-adaptive queries), we can simply answer its queries by sampling. In [GKWW21], one can use indistinguishability obfuscation to build such a sampler, so that we have watermarkable weak PRF with public extraction.

Remark 4.10 (Extraction of watermarks from honestly generated circuit). *Note that even though our extraction algorithm simulates a possibly interactive game for any input program, it does not require or assume the input program to be interactive or even stateful. As we have described, when the input circuit is not interactive, then the extraction algorithm simply samples challenge inputs and run C on them to get C's outputs.*

For example, we can also correctly extract from an honestly watermarked circuit which is neither stateful nor interactive.

Remark 4.11 (Computational Simulation). *In most of our applications, the extraction key can simulate the security game's view with perfect indistinguishability. But whether the simulation quality is perfect/statistical/computational does not affect our watermarking composition theorem and applications, as long as the difference is negligible.*

We will encounter computational simulation only when there are primitives in the set $\mathcal{T}_K$: the extraction algorithm will simulate a "hybrid version" of the security game, after invoking the security of the primitive in $\mathcal{T}_K$, instead of the original security game. See more discussions Remark 4.16.

The one example provided in our paper is the use of NIZK proof with trapdoors in Appendix A.

Meaningfulness The meaningfulness property says that an honestly generated key should not have watermarks.

A watermarking implementation of primitive P scheme satisfies the meaningfulness property if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathsf{Extract}(\mathsf{xk}, \mathsf{sk}) \neq \bot : \ (\mathsf{sk}, \mathsf{pk}, \mathsf{mk}, \mathsf{xk}) \leftarrow \mathsf{WMSetup}(1^\lambda)\ \right] \leq \mathsf{negl}(\lambda)$$

Private Extraction vs. Public Extraction A watermakable implementation of a cryptographic primitive is a scheme with public-extraction if the extraction key xk can be made public while the above properties still hold. Otherwise it has private extraction.

Single-key vs. Collusion Resistant A watermakable implementation of a cryptographic primitive is collusion resistant if in the unremovability security game, $\mathcal{A}$ is allowed to query the marking oracle $\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)$ for arbitrarily many times and q-bounded collusion resistance if it is allowed to query for a-priori bounded polynomial q-times. the scheme is single-key secure if $q = 1$.

Remark 4.12 (Defining stages in the unremovability game). *As discussed briefly in Definition 3.4: an example of subtlety when defining the staged unremovability game is: in the original security game for P, $\mathcal{A}$ needs to commit to some challenge messages that will be used by the challenger to prepare the final challenge for $\mathcal{A}$. The question is whether we make $\mathcal{A}$ commit to such challenge messages in stage 1, or let the stage-2 adversary, i.e. the circuit C output by $\mathcal{A}$ choose their challenge messages later in stage 2.*

In summary, if admissibility of queries made in stage 1 will be dependent on the challenge, then we must make $\mathcal{A}$ commit to its own part on the challenge (as some auxiliary information aux*) before it produces C , and we run C with these prefixed* aux *in stage-2 game $G_P^2(\mathsf{pk}, \mathsf{xk}, \mathsf{aux}, C)$.*

Let's take a "flexible" example, where we can let the stage-2 adversary $\mathcal{A}_2$ do such commitment, instead of forcing it to commit in stage 1. Accordingly, we can let the watermarking adversary's output program C selects its own commitment when we run C: for example, in CPA/CCA encryption games, we can allow $\mathcal{A}$ to submit its challenge messages (m_0, m_1) either together/before it outputs C or output a C that will choose *its own* (m_0, m_1)*. This flexibility comes from the fact that whether the queries made by* $\mathcal{A}$ *and* C *are valid does* not *depend on the choice of* (m_0, m_1)*.*

A non-flexible example would be the challenge attribute/identity in ABE/IBE. We must let $\mathcal{A}$ *choose its challenge attribute/identity before/at the same time with outputting* C*.* C *is simulated in a second stage security game with this pre-selected attribute/identity as its input. Otherwise* $\mathcal{A}$ *can easily cheat because the extraction algorithm(stage-2 game) does not get to check if the previous key generation queries made by* $\mathcal{A}$ *are valid with respect to challenge attribute/identity.*

4.2 Watermarking Composition: Target Primitive from Input Primitives

In this section, we show that if primtivive P is built from $P_1, \cdots, P_k$ where the security can be shown via a watermarking-compatible reduction, we can construct a watermarkable implementation of P, named WP to satisfy definition in 4.1, from existing watermarkable implementations of $P_1, \cdots, P_k$ called $\mathsf{WP}_1, \mathsf{WP}_2, \cdots, \mathsf{WP}_k$, satisfying definition in 4.1. We will still refer to P as the target primitive and $P_1, \cdots, P_k$ as input primitives.

Outline and Intuition On a high level, the watermarking scheme of the target primitive P simply follows the construction of plain P from the plain underlying building blocks in terms of evaluation algorithms $\mathsf{SecEval}, \mathsf{PubEval}$. Correctness and functionality-preserving compose in a relatively natural way.

To mark a key of P, the marking algorithm concatenates the marked keys of all underlying P_i that need to be marked. To extract a mark, we attempt to run the extraction algorithm of all underlying P_i on the input circuit. If any valid mark is extracted, then the circuit is considered marked.

In particular, the extraction algorithm will treat the circuit as an adversary in the stage-2 security game of P and turns it black-boxly into a reduction for the security game of each underlying P_i, one by one and then run the underlying extraction algorithm of P_i on it.

In order to remove a mark from a P's key, the adversary $\mathcal{A}$ must remove all marks from each P_i's key. Meanwhile, the pirate program still needs to win the security game of P, then we must be able to use to break the unremovability of at least one underlying P_i. In more generic scenarios, the pirate program made by $\mathcal{A}$ may not break any unremovability security of watermarkable building blocks, but get around the task of removing marks by breaking the security of some unwatermarked building blocks. By similar means, we can use the pirate program to build our reduction to the security of these unwatermarked building blocks. By the properties of the watermarking-compatible reduction (Section 3.2) that P's construction satisfies, the above analysis is exhaustive.

Construction of WP Similar to the description of construction in Section 3.2, we recall the following notations:

- Let $\mathcal{S} \subset [k]$ be a fixed set used in P's construction from $P_1, \cdots, P_k$, where the primitives P_i with $i \in \mathcal{S}$'s keys will be generated in the KeyGen algorithm of P. Without loss of generality, we let the first $|\mathcal{S}|$ number of P_i's be those corresponding to the set $\mathcal{S}$.
- Let $\mathcal{T}_S \in [k]$ denote the set of indices i where P_i's keys will be generated during $\mathsf{SecEval}$; Let $\mathcal{T}_P \in [k]$ denote the set of indices i where P_i's keys will be generated during $\mathsf{PubEval}$. Let $\mathcal{T}_K$ denote the set of indices i where P_i's secret key (we call trapdoor td) will be generated during $\mathsf{KeyGen}(1^\lambda)$ but will not be used in P's algorithms.

- Without loss of generality, we sometimes assume a numbering on the primitives so that the first $|\mathcal{S}|$ primitives are in the set $\mathcal{S}$.

$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk})$:

1. compute $(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{xk}_i, \mathsf{mk}_i) \leftarrow \mathsf{WP}_i.\mathsf{WMSetup}(1^\lambda)$ for all $i \in \mathcal{S}$.
2. compute $(\mathsf{pk}_\ell, \mathsf{td}_\ell) \leftarrow P_\ell.\mathsf{KeyGen}(1^\lambda)$ for all $\ell \in \mathcal{T}_K$;
3. output $\mathsf{sk} = (\mathsf{sk}_{j_1}, \cdots, \mathsf{sk}_{|\mathcal{S}|})$; $\mathsf{pk} = (\mathsf{pk}_1, \cdots, \mathsf{pk}_{|\mathcal{S}|)}, \{\mathsf{pk}_\ell\}_{\ell \in \mathcal{T}_k}))$;
   $\mathsf{xk} = (\mathsf{xk}_1, \cdots, \mathsf{xk}_{|\mathcal{S}|}, \{\mathsf{td}_\ell\}_{\ell \in \mathcal{T}_k})$; $\mathsf{mk} = (\mathsf{mk}_1, \cdots, \mathsf{mk}_{|\mathcal{S}|})$

$\mathsf{wSecEval}(\mathsf{sk}, \mathsf{pk}, x)$:

1. parse input $\mathsf{sk} = (\mathsf{sk}_1, \cdots, \mathsf{sk}_{|\mathcal{S}|})$; $\mathsf{pk} = (\mathsf{pk}_1, \cdots, \mathsf{pk}_{|\mathcal{S}|}, \{\mathsf{pk}_\ell\}_{\ell \in \mathcal{T}_k}))$.
2. $\mathsf{wSecEval}(\mathsf{sk}, \mathsf{pk},)$ is the same algorithm as $P.\mathsf{SecEval}(\mathsf{sk}, \mathsf{pk}, \cdot)$ in the construction of P from $P_1, \cdots, P_k$, except that $P_i.\mathsf{SecEval}(\mathsf{sk}_i, \cdot)$ is replaced with $\mathsf{WP}_i.\mathsf{wSecEval}(\mathsf{sk}_i, \cdot)$ for $i \in \mathcal{S}$. Overall, $\mathsf{wSecEval}(\mathsf{sk}, \mathsf{pk},)$ is an algorithm that:
   (a) uses $\mathsf{WP}_i.\mathsf{wSecEval}(\mathsf{sk}_i, \mathsf{pk}_i, \cdot), i \in \mathcal{S}$ as subroutines.
   (b) uses $\mathsf{WP}_i.\mathsf{wPubEval}(\mathsf{pk}_i, \cdot), i \in \mathcal{S} \cup \mathcal{T}_k$ as subroutines.
   (c) computes $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow P_j.\mathsf{KeyGen}(1^\lambda)$ for some $j \in \mathcal{T}_S$ (and may include these pk_j generated as part of the output).

$\mathsf{wPubEval}(\mathsf{pk}, x)$

1. parse input $\mathsf{pk} = (\mathsf{pk}_1, \cdots, \mathsf{pk}_{|\mathcal{S}|}, \{\mathsf{pk}_\ell\}_{\ell \in \mathcal{T}_k})$.
2. $\mathsf{wPubEval}(\mathsf{pk},)$ is the same algorithm as $P.\mathsf{PubEval}(\mathsf{pk}, \cdot)$ in the construction of P from $P_1, \cdots, P_k$, except that $P_i.\mathsf{PubEval}(\mathsf{pk}_i, \cdot)$ is replaced with $\mathsf{WP}_i.\mathsf{wPubEval}(\mathsf{pk}_i, \cdot)$ for $i \in \mathcal{S} \cup \mathcal{T}_k$. Overall, $\mathsf{wPubEval}(\mathsf{sk}, \mathsf{pk}, \cdot)$ is an algorithm that:
   (a) uses $\mathsf{WP}_i.\mathsf{wPubEval}(\mathsf{pk}_i, \cdot), i \in \mathcal{S}$ as subroutines.
   (b) uses $\mathsf{WP}_i.\mathsf{wPubEval}(\mathsf{pk}_i, \cdot), i \in \mathcal{S}$ as subroutines.
   (c) computes $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow P_j.\mathsf{KeyGen}(1^\lambda)$ for some $j \in \mathcal{T}_P$ (and may include these pk_j generated as part of the output).

$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau \in \mathcal{M}_{\mathsf{Mark}}) \to \mathsf{sk}_\tau$

1. parse $\mathsf{mk} = (\mathsf{mk}_1, \cdots, \mathsf{mk}_{|\mathcal{S}|})$; $\mathsf{sk} = (\mathsf{sk}_1, \cdots, \mathsf{sk}_{|\mathcal{S}|})$; $\mathsf{pk} = (\mathsf{pk}_1, \cdots, \mathsf{pk}_{|\mathcal{S}|}, \{\mathsf{pk}_\ell\}_{\ell \in \mathcal{T}_k})$.
2. Compute $\mathsf{sk}_{i,\tau} \leftarrow \mathsf{WP}_i.\mathsf{Mark}(\mathsf{mk}_i, \mathsf{sk}_i, \tau)$ for all $i \in \mathcal{S}$.
3. output $\widetilde{\mathsf{sk}} = (\mathsf{sk}_{1,\tau}, \cdots, \mathsf{sk}_{|\mathcal{S}|,\tau})$

$\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C) \to \tau \in \mathcal{M}_{\mathsf{Mark}} / \vec{\tau} \in \mathcal{M}_{\mathsf{Mark}}^{|\mathcal{S}|} / \bot$:

1. parse $\mathsf{xk} = (\mathsf{xk}_1, \cdots, \mathsf{xk}_{|\mathcal{S}|}, \{\mathsf{td}_\ell\}_{\ell \in \mathcal{T}_K})$; $\mathsf{aux} = (\mathsf{aux}_1, \cdots, \mathsf{aux}_{|\mathcal{S}|})$; $\mathsf{pk} = (\mathsf{pk}_1, \cdots, \mathsf{pk}_{|\mathcal{S}|}, \{\mathsf{pk}_\ell\}_{\ell \in \mathcal{T}_k})$.
2. Initialize an empty set $\vec{\tau}$.
3. For each $i \in \mathcal{S}$:
   (a) prepare the following circuit C_i using black-box access to C.
      i. C_i uses $(\{\mathsf{xk}_j\}_{j \in \mathcal{S}, j \neq i}, \{\mathsf{td}_\ell\}_{\ell \in \mathcal{T}_K})$ and external queries to simulate security game $G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, C)$ for C [8]:
         - For any (admissible) queries in the stage-2 game $G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, C)$, if the oracles provided in security game $G_{P_i}(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{aux}_i, \cdot)$ is required to compute

[8] C_i can also have xk_i if the watermarkable implementation WP_i has a public extraction key. In this case, C_i also does not need to make external queries.

the answer for the query, C_i can will make a query to an external challenger. To answer the entire query, C_i may finish the rest of the computation using $(\{\mathsf{xk}_j\}_{j\neq i}, \{\mathsf{td}_\ell\}_{\ell\in\mathcal{T}_k})$.
   - If only $\{\mathsf{xk}_j\}_{j\neq i}$ are needed to answer a query, C_i answers it by itself.

  ii. C_i records queries from C into a set $\mathcal{Q}_C$.
  iii. C_i receives its challenge input inp_i from the interaction with an external challenger, samples a random string r, and prepares a challenge input inp for C using inp_i and r.
  iv. If there is a query phase after the challenge phase, C_i continues to simulate the oracles required using $\{\mathsf{xk}_j\}_{j\neq i}$ and external queries to a challenger.
  v. After C makes its final output out, C_i computes the function $f_i(\mathsf{out}, r, \mathcal{Q}_c, \mathsf{inp})$ where f_i is the reduction function (Remark 3.14) used in watermarking-compatible reduction of P to P_i . Output the result of this computation.

(b) compute $\tau_i/\bot \leftarrow \mathsf{WP}_i.\mathsf{Extract}(\mathsf{xk}_i, C_i)$.
(c) add τ_i(or $\bot$) to the tuple $\vec{\tau}$, and go to step 2 with $i := i + 1$.

4. Output
   - $\vec{\tau} = (\tau_1, \cdots, \tau_{|\mathcal{S}|})$ if $\exists i, j$ where $\tau_i \neq \tau_j$;
   - else if $\tau_i = \tau_j = \tau$(or $\bot$) for all i, j, output τ (or $\bot$ resp.).

Remark 4.13. *We also give a more systematic description of Extract is using the language of watermarking-compatible reduction:*

The Extract *algorithm:*

1. *On input circuit C and* xk, pk, aux*; for $i \in \mathcal{S}$:*
   (a) *Create program C_i (C_i has black-box access to C). C_i treats C as the stage-2 adversary $\mathcal{A}^2$ in the stage-2 game $G^2_P(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \mathcal{A}_2)$.*
   (b) *C acts as a stage-2 reduction $\mathcal{A}^2_i$ in the watermarking-compatible reduction from primitive P to P_i, with the difference that C uses $\{\mathsf{xk}_j\}_{j\neq i}$ (instead of $\{\mathsf{sk}_j\}_{j\neq i}$) and external queries to challenger to simulate the game.*
   (c) *compute $\tau/\bot \leftarrow \mathsf{WP}_i.\mathsf{Extract}(\mathsf{xk}_i, C_i)$; add it to the extracted mark tuple.*
2. *Output the tuple of marks extracted.*

Remark 4.14. *As discussed previously, for the convenience of discussions in some later parts, our* Mark *algorithm outputs a marked key instead of circuits since the* wSecEval *algorithm itself is public.*

Remark 4.15 (Outputting a tuple of marks)**.** *The* Extract *algorithm will only output a tuple of marks (with $|\mathcal{S}|$ number of them at most) when used on adversarial circuits C. For honestly generated circuit C, the extraction will always output only one marked message.*

First this is clearly not a limitation because $|\mathcal{S}|$ is the number of input primitives and usually a small constant/polynomial. Second, the need for the extraction algorithm to output a tuple of marks is essential for the composability of watermarking schemes, otherwise we cannot capture the following trivial "attack": the adversary receives an honestly marked output primitive key, which consists of k marked secret keys of the input primitives. It can simply destroy all the keys except one and this leftover marked key suffices for breaking the security of P. Such an "attack" should not be considered as a successful attack because we can still extract a valid watermark from the entire program. Therefore, we need the extraction to output a *tuple of marks so that we can check if the adversary has succeeded at removing/replacing marks from each underlying functionality.*

Remark 4.16. *As previously discussed in Remark 4.11, when we have a primitive in the set $\mathcal{T}_K$, the* $\mathsf{Extract}$ *algorithm needs to use the trapdoor(s)* td *of the primitives in set $\mathcal{T}_K$ to simulate a hybrid version of the security game G_P. This simulation is supposedly computationally indistinguishable from the original game, by the security of the primitive(s) in $\mathcal{T}_K$.*

In more detail, $\mathcal{T}_k$ only contains primitives P_ℓ where

1. *There exists an efficient simulator algorithm such that the output of $P_\ell.\mathsf{PubEval}(P_\ell.\mathsf{pk}, \cdot)$ is indistinguishable from the output of the simulator $\mathsf{Sim}(P_\ell.\mathsf{td}, \cdot)$. Their secret key (trapdoor* td*) will only come up in the security proof for P.*
2. *In the security proof for P, the rest of the hybrid arguments and reduction happen in a hybrid game where we have already invoked the above security for P_ℓ.*

Due to the above reasons, the $\mathsf{Extract}$ *algorithm has to simulate hybrid version of game G_P where the security of $P_\ell \in \mathcal{T}_K$ is already invoked, in order for the reduction (and thus extraction) for the rest of the primitives to go through.*

Essentially, we simply need the plain security of primitives in the set $\mathcal{T}_K$ to realize the watermarking security of P, like other primitives in $\bar{\mathcal{S}}$.

Such examples are rare. The only example in this work is a NIZK proof system with a trapdoor used to simulate proofs (Appendix A).

We now state our main theorem:

Theorem 4.17 (Watermarking Composition)**.** *Suppose the input primitives in the set $\mathcal{S}$, $\{P_i\}_{i\in\mathcal{S}}$ have watermarkable implementations $\{\mathsf{WP}_i\}_{i\in\mathcal{S}}$ which satisfy the definitions in 4.1, the primitives outside the set $\mathcal{S}$ have secure constructions (defined by their corresponding correctness and security in Section 3.1), and the construction of P is watermarking-compatible by the definitions in 3.2, then the output watermarkable implementation* WP *for primitive P will satisfy the properties in Definition 4.1.*

To prove Theorem 4.17, we show that all the properties from Section 4.1 hold.

Functionality-Preserving

Lemma 4.18 (Functionality-Preserving and Exact Functionality-Preserving)**.** *Suppose the above construction of P from $P_1, \cdots, P_k$ satisfies correctness of construction, and constructions $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ satisfy the functionality-preserving properties in Definition 4.1, then* WP *satisfies correctness defined by predicate F_R.*

If all $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ satisfy the exact functionality-preserving properties, then WP *satisfies exact functionality-preserving.*

Proof. Since $\mathsf{WP}_1, \cdots, \mathsf{WP}_k$ satisfy the functionality-preserving properties in Definition 4.1, the correctness properties of $P_1, \cdots, P_k$ are preserved by their watermarkable implementations. Since P's correctness follows from the correctness properties of $P_1, \cdots, P_k$ and moreover, WP's evaluation algorithms are constructed the same way as they are in P (even though the $\mathsf{WMSetup}(1^\lambda)$ algorithm differs from $\mathsf{KeyGen}(1^\lambda)$, we can ignore the watermarking-related components xk, mk in this context). The constructions for primitives $P_i, i \notin \mathcal{S}$ also do not affect functionality preserving since we use their plain, unwatermarked keys during evaluations. WP's correctness property thereby follows.

It is easy to see that if all WP_i satisfies exact functionality preserving, then WP satisfies exact functionality preserving. □

Correctness of Extraction Our correctness property is a computational correctness that relies on the unremovability of the input watermarkable implementations. Since our Extract algorithm interacts with any input circuit black-boxly by simulating the security game G_P, the only way to extract a watermark is through the circuit's input-output behavior in this game. We use the guarantee that "if the circuit can win the game, then we can extract a watermark" to show correctness on a circuit embedded with an honestly watermarked key (which can win the game with probability 1 naturally). Also note that we do not need our honestly watermarked circuit to be stateful/interactive as in Remark 4.10.

Lemma 4.19 (Correctness of Extraction)**.** *Suppose the above construction of P from $P_1, \cdots, P_k$ has a watermarking-compatible reduction, and all watermarkable implementation of input primitives in set $\mathcal{S}$,* $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ *satisfy strong unremovability and the extraction syntax, then the watermarkable implementation of target primitive* WP *satisfies correctness of extraction.*

Proof. Given an honestly marked key $\mathsf{sk}_\tau = (\mathsf{sk}_{1,\tau}, \cdots, \mathsf{sk}_{|\mathcal{S}|,\tau}) \leftarrow \mathsf{WP.Mark}(\mathsf{mk}, \mathsf{sk})$: we can observe that there exists a public algorithm T that once given sk_τ, can win the game $G_P(\mathsf{sk}, \mathsf{pk}, \cdot)$ with probability $(1 - \mathsf{negl}(\lambda))$. We can sample a random $i \leftarrow \mathcal{S}$ and design a circuit T to work as follows: T on any challenge input, uses a strategy associated with the key sk_i to compute the challenge given in game G_P^2.

Take the concrete example of the watermarkable implementation WP for a CCA2 encryption scheme, used in our technical overview section (Section 1.4). The watermarked key is a concatenation of a watermarked decryption key/weak PRF key $\mathsf{sk}_{1,\tau}$ and a watermarked MAC signing key $\mathsf{sk}_{2,\tau}$. Then the algorithm T can choose the weak PRF key as its functionality: on any challenge ciphertext, T simply decrypts the ciphertext using the watermarked key $\mathsf{sk}_{1,\tau}$ and outputs the correct answer.

We then look into what happens when we compute $\mathsf{Extract}(\mathsf{xk}, C)$: C is treated as a stage-2 adversary in the watermarking-compatible reduction from the security of P to the security of P_i and C_i with black box access to C is a stage-2 reduction algorithm. C, for each $i \in \mathcal{S}$. We input $T(\mathsf{sk}_\tau, \cdot)$ into the extraction algorithm. By the properties of watermarking-compatible reduction, such a $T(\mathsf{sk}_\tau, \cdot)$ will be used to build C_i that is γ_i-good for some noticeable γ_i for each $i \in \mathcal{S}$. When it comes to the index i that we choose to be our functionality in T, then the created circuit C_i using T will naturally help win the security game $G_{P_i}^2$.

If we view the honest user as an adversary that makes one marking query on some message τ, then by the unremovability security of WP_i, we must have $\Pr[\mathsf{WP}_i.\mathsf{Extract}(\mathsf{xk}_i, \mathsf{pk}_i, \mathsf{aux}, C_i = \tau] \geq 1 - \mathsf{negl}(\lambda)$ for all i. Therefore $\Pr[\mathsf{WP.Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, (\mathsf{sk}_\tau, \cdot)) = \tau] \geq 1 - \mathsf{negl}(\lambda)$. Since extracting one valid watermark suffices, we can conclude the correctness of extraction.

Going back to the CCA2 encryption example: the circuit T can answer any challenge by decrypting using the PRF key; therefore, when it is used by the circuit C_1 created in Extract algorithm, its output helps C_1 distinguish between pseudorandom input and real random input. By the unremovability security of the watermarkable weak PRF, we must be able to extract the queried watermark τ from C_1.

□

Security with G_P We show that the (plain) security of P with game G_P holds.

Lemma 4.20 (Security with G_P)**.** *Suppose the above construction of P from $P_1, \cdots, P_k$ has a provable black-box security, and watermarkable implementations* $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ *as well as the (unwatermarked) constructions for $\{P_i\}_{i \notin \mathcal{S}}$ satisfy the security with the security defined by game $G_{P_i}, i \in [k]$ respectively, then* WP *satisfies security defined by game G_P.*

Proof. Since $\mathsf{WP}_1, \cdots, \mathsf{WP}_k$ satisfy the security property with game G_{P_i} in Definition 4.1, the security properties of $P_1, \cdots, P_k$ are preserved by their watermarkable implementations. P's security with respect to G_P follows from the security properties of $P_1, \cdots, P_k$ with respect to $\{G_{P_i}\}_{i \in [k]}$. Moreover, WP's evaluation algorithms are constructed the same way as they are in P and even though the $\mathsf{WMSetup}(1^\lambda)$ algorithm differs from $\mathsf{KeyGen}(1^\lambda)$, we can ignore the watermarking-related components xk, mk in this context. Therefore, WP's security property follows from the security of $\mathsf{WP}_1, \cdots, \mathsf{WP}_k$ as P's security property follows from the security of $P_1, \cdots, P_k$. □

Unremovability We now show that the construction satisfies unremovability if all building blocks satisfy unremovability (or plain security, for unwatermarked building blocks) and the properties on extraction key/algorithm defined in section 4.1.

Lemma 4.21 (Unremovability)**.** *Suppose the above construction of P from $P_1, \cdots, P_k$ has a watermarking-compatible reduction, the watermarkable implementations* $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ *satisfy strong unremovability (Definition 4.7), the constructions for $\{P_j\}_{j \notin \mathcal{S}}$ satisfy security defined by their game $\{G_{P_j}\}_{j \notin \mathcal{S}}$ respectively, and the extraction algorithms of* $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ *satisfy the extraction syntax in Section 4.1, then the watermarkable implementation* WP *satisfies strong unremovability.*

Proof. If there exists an adversary that breaks γ-unremovability with game G_P as defined in Definition 4.4, then we We have $\Pr[\mathsf{Extract}(\mathsf{xk}, C) \notin \mathcal{Q} \land C \text{ is } \gamma\text{-good}] \geq \epsilon$ for some non-negligible ϵ where C is γ-good in the sense that $\Pr[G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, C) = 1] \geq \eta + \gamma$. Here the overall winning probability of $\mathcal{A}$ is taken over the randomness used in KeyGen and the unremovability game; the probability for $G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, C) = 1$ is taken over the randomness used in the stage-2 game G_P^2.

By the design of our Extract algorithm, it must be the case that, for all $i \in \mathcal{S}$, $\mathsf{WP}_i.\Pr[\mathsf{Extract}(\mathsf{xk}_i, C_i) \notin Q] \geq \epsilon$, where C_i is the circuit made black-boxly from C in WP.Extract algorithm for index i.

Now we divide our analysis into cases. First, we consider that there is no input primitive in the set $\mathcal{T}_k$, i.e. no trapdoor td used in the extraction procedure and the extraction key can perfectly simulate the security game for G_P.

For any $\mathcal{A}$ producing a circuit C such that $\Pr[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C) \notin \mathcal{Q} \land C \text{ is } \gamma\text{-good}] \geq \epsilon$ (where the probability is taken over the randomness in KeyGen and randomness used throughout the game), one of the following cases must hold:

- **Case 1: For some non-negligible probability ϵ_i, during the execution of** $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C)$**, there exists some $i^* \in \mathcal{S}$ such that C_{i^*} is a γ_{i^*}-good program in $G_{P_i}^2(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{aux}_i, C_i)$ for some non-negligible γ_{i^*}.**
  Then, there exists a reduction to break the γ_{i^*}-unremovability of WP_{i^*} for some non-negligible γ_{i^*}, using $\mathcal{A}$. Let $\mathcal{A}_{i^*}$ be the reduction and adversary in γ_{i^*}-unremovability game of WP_{i^*}.

- $\mathcal{A}_{i^*}$ receives pk_{i^*} from the challenger and samples $\{\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{xk}_i, \mathsf{mk}_i\}_{i \in \mathcal{S}, i \neq i^*}$ from $\mathsf{WP}_{i^*}.\mathsf{WMSetup}$. It gives $\{\mathsf{pk}_i\}_{i \in \mathcal{S}, i \neq i^*}$ to $\mathcal{A}$.
- When $\mathcal{A}$ makes a marking query on some symbol τ, $\mathcal{A}_{i^*}$ responds as follows:
  * compute $\mathsf{sk}_{i,\tau} \leftarrow \mathsf{Mark}(\mathsf{mk}_i, \mathsf{sk}_i, \tau)$ for $i \in \mathcal{S}, i \neq i^*$.
  * query the WP_{i^*} challenger on oracle $\mathsf{Mark}(\mathsf{mk}_{i^*}, \mathsf{sk}_{i^*}, \tau)$ to obtain $\mathsf{sk}_{i^*,\tau}$.
  * Let $\mathsf{sk}_\tau = \{\mathsf{sk}_{i,\tau}\}_{i \in \mathcal{S}}$; send sk_τ to $\mathcal{A}$.
- Next, $\mathcal{A}$ outputs circuit polynomial-size C.
- $\mathcal{A}_{i^*}$ creates the following circuit C'_{i^*} using C as a black-box:
  * C'_{i^*} is hardcoded with $\{\mathsf{sk}_i\}_{i \in \mathcal{S}, i \neq i^*}, \{\mathsf{pk}_i\}_{i \in \mathcal{S}}$.
  * When C makes a query to oracles in game G_P, C'_{i^*} responds as follows: if sk_{i^*} is required to answer the query, C_{i^*} queries the external challenger on the oracles provided in game $G_{P_{i^*}}$; otherwise, C'_{i^*} answers the query using $\{\mathsf{sk}_i\}_{i \in \mathcal{S}, i \neq i^*}$.
  * C'_{i^*} records all queries from C, denoted as $\mathcal{Q}_C$.
  * In the challenge phase, C'_{i^*} receives challenge input inp_i from the interaction with an external challenger, samples a random string r, and prepares a challenge input inp for C using inp_i and r.
  * When C outputs its final answer out, C_{i^*} computes the function $f_{i^*}(\mathsf{out}, \mathcal{Q}_C, \mathsf{inp})$ according to the reduction function f_{i^*} in the watermarking-compatible reduction from P to P_{i^*}.
- In short, C is treated as a stage adversary $\mathcal{A}_2$ in security game G_P; circuit C_{i^*} is a stage-2 reduction algorithm in the watermarking-compatible reduction from target primitive P to input primitive P_{i^*}, with black box access to C.

By the design of the $\mathsf{WP}.\mathsf{Extract}$ algorithm, it using extraction keys $\{\mathsf{xk}_i\}_{i \in \mathcal{S}}$, will treat the input circuit as an adversary in stage-2 game G_P^2 and simulate G_P^2 for it.
Given any adversarial circuit C that is a γ-good adversary in the security game G_P, following our extraction algorithm syntax requirement, the extraction algorithm running at step $i = i^*$ will create a circuit C_{i^*} that has exactly the same functionality as the circuit C'_{i^*} created in the above reduction [9]. Therefore, if C_{i^*} created by $\mathsf{Extract}$ is a γ_{i^*}-good in breaking $G_{P_{i^*}}$, then so is C'_{i^*} created by the above procedure; since we have $\mathsf{WP}_{i^*}. \Pr[\mathsf{Extract}(\mathsf{xk}_{i^*}, C_{i^*}) \notin Q] \geq \epsilon$ and the set Q is the same for $\mathcal{A}$ and $\mathcal{A}_{i^*}$, then we also have $\mathsf{WP}_i. \Pr[\mathsf{Extract}(\mathsf{xk}_i, C'_{i^*}) \notin Q] \geq \epsilon$.

- **Case 2: With overwhelming probability, during the execution of** $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C)$**, there exists *no*** $i \in \mathcal{S}$ **such that** C_i **is a** γ_i**-good adversary in** $G_{P_i}^2(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{aux}_i, C_i)$ **for some non-negligible** γ_i**.**
  In this case, it must be that there exists some $i^* \notin \mathcal{S}$ such that we can use the unremovability adversary $\mathcal{A}$ for WP to break the *security* of P_{i^*}, i.e. winning game $G_{P_i}, i \notin \mathcal{S}$ with some non-negligible advantage.
  By the properties of watermarking-compatible reduction, if (with non-negligible probability), the circuit C produced by $\mathcal{A}$ is a γ-good adversary in G_P, then there must exist some $i \in [k]$ such that using the watermarking-compatible reduction, one can create a γ_i-good adversary for some non-negligible γ_i. By the above analysis, C_i supposedly performs exactly the reduction for P_i and if there is no good C_i for $i \in \mathcal{S}$, then C can only be used to build a

[9] Note that the external queries made by C_{i^*} will be answered by $\mathsf{WP}_{i^*}.\mathsf{Extract}(\mathsf{xk}_{i^*}, C_{i^*})$ during its execution, where by our requirement on the extraction syntax, $\mathsf{WP}_{i^*}.\mathsf{Extract}(\mathsf{xk}_{i^*}, C_{i^*})$ will simulate the game $G_{P_{i^*}}$ for C_{i^*}, answering its queries.

reduction to break the security of some $P_i, i \notin \mathcal{S}$.

The reduction is as follows. Let $\mathcal{A}_{i^*}$ be the reduction and adversary in the security game $G_{P_{i^*}}$.

- $\mathcal{A}_{i^*}$ receives pk_{i^*} from the challenger and samples $\{\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{xk}_i, \mathsf{mk}_i\}_{i\in\mathcal{S}}$ from $\mathsf{WP}_{i^*}.\mathsf{WMSetup}$. It gives $\{\mathsf{pk}_i\}_{i\in\mathcal{S}}$ to $\mathcal{A}$.
- When $\mathcal{A}$ makes a marking query on some symbol τ, $\mathcal{A}_{i^*}$ answers the query on its own because it has all the $\{\mathsf{sk}_i, \mathsf{mk}_i\}_{i\in\mathcal{S}}$.
- Besides answering marking queries, $\mathcal{A}_{i^*}$ will treat $\mathcal{A}$ as a stage-1 adversary in the watermarking-compatible reduction Type 2 (Section 3.2): when $\mathcal{A}$ makes a query to oracles in game G_P, $\mathcal{A}_{i^*}$ responds as follows: if sk_{i^*} is required to answer the query, $\mathcal{A}_{i^*}$ queries the challenger in $G_{P_{i^*}}$ security game; otherwise, $\mathcal{A}_{i^*}$ answers the query using $\{\mathsf{sk}_i\}_{i\in\mathcal{S}}$.
  For the secret keys of $\{\mathsf{WP}_i\}_{i\notin\mathcal{S}, i\neq i^*}$, they are all sampled freshly upon every query of $\mathsf{wSecEval}(\mathsf{sk}, \mathsf{pk}, \cdot)$ (or sampled freshly in $\mathsf{wPubEval}(\mathsf{pk}, \cdot)$ which $\mathcal{A}$ can do it on its own).
- Next, $\mathcal{A}$ outputs circuit polynomial-size C.
- $\mathcal{A}_{i^*}$ creates the following circuit C^* using C as a black-box:
  * C^* is hardcoded with $\{\mathsf{sk}_i, \mathsf{pk}_i\}_{i\in\mathcal{S}}$.
  * When C makes a query to oracles in game G_P, C^* responds as follows: if sk_{i^*} is required to answer the query, C^* queries the external challenger on the oracles provided in game $G_{P_{i^*}}$; otherwise, C^* answers the query using $\{\mathsf{sk}_i\}_{i\in\mathcal{S}}$.
  * C^* records all queries from C, denoted as $\mathcal{Q}_C$.
  * In the challenge phase, C^* receives challenge input inp_i from the interaction with an external challenger, samples a random string r, and prepares a challenge input inp for C using inp_i and r.
  * When C outputs its final answer out, C^* computes the function $f_{i^*}(\mathsf{out}, \mathcal{Q}_C, \mathsf{inp})$ according to the reduction function f_{i^*} in the watermarking-compatible reduction from P to P_{i^*}.
- In short, C is treated as a stage-2 adversary $\mathcal{A}_2$ in security game G_P; circuit C^* is a stage-2 reduction algorithm in the watermarking-compatible reduction Type 2 from target primitive P to input primitive P_{i^*}, with black box access to C. By the property of the reduction, C^* should win the security game $G_{P_{i^*}}$ with non-negligible advantage.

Finally, we discuss the rare case when the extraction key simulates the security game for C statistically/computationally close to a real game.

In the setting where $(\{\mathsf{xk}_i\}_{i\in\mathcal{S}}, \{\mathsf{td}_\ell\}_{\ell\in\mathcal{T}_K})$ can only simulate G_P^2 for C statistically close by some negligible distance, then if C is a γ-good circuit in a real $G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \cdot)$ game, then C is a $(\gamma - \mathsf{negl}(\lambda))$-good circuit in the game simulated by the $\mathsf{Extract}$ algorithm and thus by the reduction. Afterwards, we can go back to use C in the above Case 1 and Case 2.

Coming to the computational setting, we will see that they will also be transformed into Case 1 or Case 2.

Note that within the scope of this work, this case corresponds precisely to having primitives in the set $\mathcal{T}_k$ and the computational property comes from primitives in the set $\mathcal{T}_k$. Recall that $\mathcal{T}_k$ only contains the primitives P_ℓ where there exists a simulator algorithm such that the output of $P_\ell.\mathsf{PubEval}(P_\ell.\mathsf{pk}, \cdot)$ is indistinguishable from the output of the simulator $\mathsf{Sim}(P_\ell.\mathsf{td}, \cdot)$ and in the security proof for P, the rest of the hybrid arguments and reduction happen in a hybrid world where we have already invoked the security for P_ℓ. (see discussions in Re-

mark 3.15, Definition 4.9 and Remark 4.11, Remark 4.16).
We provide one such example where we need to use a NIZK proof and provide simulated proofs to the adversarial circuits in the extraction algorithm. See the example in Appendix A for details.
First consider the case where we only have one primitive P_ℓ in set $\mathcal{T}_K$ in our construction (this is the case in our example Appendix A). By our premise, using $(\{\mathsf{xk}_i\}_{i\in\mathcal{S}}, \mathsf{td}_\ell\})$ where $\mathsf{td}_\ell \leftarrow P_\ell.\mathsf{KeyGen}(1^\lambda)$, one can simulate G_P^2 for C computationally close by some negligible amount. If with some non-negligible probability ϵ, we have C is a γ-good circuit in a real $G_P^2(\mathsf{sk}, \mathsf{pk}, \mathsf{aux}, \cdot)$ game. Then with $(\epsilon - \mathsf{negl}(\lambda))$ probability, C is a $(\gamma - \mathsf{negl}(\lambda))$-good circuit in the game simulated by the Extract algorithm and the reductions, otherwise there exists a PPT distinguisher that can use C to distinguish between the real game and simulated one to break the security of a primitive in set $\mathcal{T}_k$: a reduction $\mathcal{B}$ can samples all $\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk}$ on its own and receive the public parameters from the challenger of some G_{P_ℓ}. The rest of simulation is the same as in Case 2 of the above analysis. At the end, $\mathcal{B}$ tests if the circuit output by $\mathcal{A}$ is a γ-good one, if yes, output guess "real view; else output guess "simulated view".
If there are more than one$P_\ell \in \mathcal{T}_K$, we can then use a hybrid to apply the above to each of them, following the hybrid order in the security proof for the plain security of P.
Therefore, by the security of P_ℓ, we have that C is a $(\gamma - \mathsf{negl}(\lambda))$-good circuit in the game simulated by the Extract algorithm. Since the drop in C's advantage is negligible, we can go back to Case 1 and Case 2.

□

Additional Properties Finally, we discuss some additional properties realized by the watermarking composition.

Private vs. Public Extraction Firstly, we have a relatively straightforward observation that if all underlying constructions satisfy public extraction, then the target construction also does.

Lemma 4.22 (Public Extraction)**.** *Suppose a watermakable implementation* WP *is constructed from watermarkable implementations* $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$*, if* $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ *all have public extraction and there exists no primitive in the set* $\mathcal{T}_K$*, then* WP *has public extraction.*

Proof. It is easy to observe that if all WP_i have public extraction, then all xk_i can be made public and the extraction algorithm of WP can be made public. During the extraction procedure, since the keys in the primitives in the set $\mathcal{T}_S, \mathcal{T}_P$ are sampled online, even a public procedure can sample them. But if we have a primitive in the set $\mathcal{T}_K$, then we need a trapdoor during extraction, so we have to rule out this case. □

Meaningfulness The meaningfulness of WP follows from the meaningfulness of each WP_i construction naturally.

Collusion Resistance Finally, we show that collusion resistance also composes. Informally, suppose a watermakable implementation WP is constructed from watermarkable implementations $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$, if all of $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ are collusion resistant, then WP is collusion resistant. If at least one of them is q-bounded collusion resistant, then WP is q-collusion resistant q (by the smallest $q_i, i \in |\mathcal{S}|$).

Lemma 4.23 (Collusion Resistance)**.** *Suppose the above construction of P from $P_1, \cdots, P_k$ has a watermarking-compatible reduction, the watermarkable implementations $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ all satisfy strong unremovability (Definition 4.7) with q-collusion-resistance, the constructions for $\{P_j\}_{j \notin \mathcal{S}}$ satisfy security defined by their game $\{G_{P_j}\}_{j \notin \mathcal{S}}$ respectively, and the extraction algorithms of $\mathsf{WP}_1, \cdots, \mathsf{WP}_{|\mathcal{S}|}$ satisfy the extraction syntax in Section 4.1, then he watermarkable implementation WP satisfies strong unremovability with q-collusion resistance.*

Proof. Since each marking query of WP requires querying each underlying marking oracle $\mathsf{WP}_i.\mathsf{Mark}$ once on the same marking message, the number of queries allowed for the reduction from WP's unremovability is bounded by the input primitive WP_i's query bound. If all underlying schemes are q-collusion resistant for some q, then all reductions can query q times and the unremovability proof for Lemma 4.21 stays the same. WP is also q-collusion resistant.

□

Extraction Simulation Property and Extraction Syntax Having this additional property in the composed watermarkable implementation scheme allows us to further compose watermarking schemes from input schemes that come from composition themselves.

It is easy to see that the ability of the extraction algorithm to simulate the security game also composes.

We can observe that if all watermarkable implementations of input primitives satisfy the extraction syntax (Section 4.1,and thus key simulation property Definition 4.9) , then the combined keys $(\{\mathsf{xk}_i\}_{i\in\mathcal{S}}, \{\mathsf{pk}_i\}_{i\in\mathcal{S}})$ can be used to simulate the game G_P because all oracles used in G_P can all be built from oracles used in $\{G_{P_i}\}_{i\in\mathcal{S}}$ according to the watermarking-compatible reduction.

Since all the $\mathsf{WP}_i.\mathsf{Extract}(\mathsf{xk}_i, \mathsf{pk}_i, \mathsf{aux}_i, \cdot)$ algorithm perfectly(statistically/computationally, resp.) simulates the security game for G_{P_i} stage 2 (as a subroutine of extraction), when each C_i created from input program C is executed inside the procedure $\mathsf{WP}_i.\mathsf{Extract}(\mathsf{xk}_i, \mathsf{pk}_i, \mathsf{aux}_i, C_i)$: C_i will simulate the game G_P stage 2 for C using external oracle queries when interacting with the algorithm $\mathsf{WP}_i.\mathsf{Extract}(\mathsf{xk}_i, \mathsf{pk}_i, \mathsf{aux}_i, C_i)$ and the other extraction keys $\{\mathsf{xk}_i\}_{i\in\mathcal{S}}$. The entire Extract algorithm consists of subroutines where one simulates G_P perfectly(statistically/computationally, resp.) stage 2 for C when running C.

When we have input primitives in $\mathcal{T}_k$, as we have discussed in several remarks: in the extract trapdoor(s) td of primitives in $\mathcal{T}_k$ to simulate part of the security game, which results in a computationally close simulation from the original game. One concrete example of such primitive in $\mathcal{T}_k$ is NIZK, which we will use in Appendix A and Appendix G.

5 Two Simple Examples: Watermarkable CPA and CCA-secure SKE

In this section, we give two relatively simple examples through the following two steps:

- First we give their "plain" construction and show that their reduction is watermarking-compatible.
- Then give their watermarking implementation of the target primitive in the watermarking-compatible reduction language.

For the remaining examples we show in this work, we will integrate the above two discussions for brevity.

5.1 Watermarkable CPA-secure SKE from Watermarkable weak PRF

In this section, we show a simple example of obtaining a watermarkable implementation of a CPA secure secret-key encryption scheme from a watermarkable implementation of weak PRF. The security of watermarkable weak PRF satisfying our requirements in Section 4.1 can be built from LWE (with private extraction) or iO (with public extraction) [GKWW21]. The following example was also discussed in [GKWW21].

5.2 Definition: Watermarkable Implementation of PRF

We give the functionality-preserving unremovability security definition for PRF and CPA-secure SKE. The other definitions (pseudorandomness, CPA-security, correctness, functionality preserving) follow naturally from the correctness and security defintions of PRF and SKE respectively.

A watermarkable implementation of a PRF scheme consists of the following algorithms $(\mathsf{WMSetup}, \mathsf{Eval}, \mathsf{Mark},$ and satisfies the following properties:

$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{sk}, \mathsf{mk}, \mathsf{xk})$: on input security parameter, outputs a secret key sk, marking key mk, extraction key xk.

$\mathsf{Eval}(\mathsf{sk}, x \in \{0,1\}^\ell) \to y \in \{0,1\}^\ell$: a deterministic evaluation algorithm that on input sk, x, outputs y.

$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau)$: on marking key mk, secret key sk, a message τ, output a marked key sk_τ.

$\mathsf{Extract}(\mathsf{xk}, C)$: on extraction key sk and program C, output a mark $\tau \in \mathcal{M}_\tau$ or $\perp$.

Functionality-Preserving A watermarkable PRF is functionality preserving if then there exists a negligible function $\mathsf{negl}(\lambda)$ ssuch that for all $\lambda \in \mathbb{N}, , x \in \mathcal{X}, \tau \in \mathcal{M}_\tau$:

$$\Pr\left[\mathsf{Eval}(\tilde{\mathsf{sk}}, x) = \mathsf{Eval}(\mathsf{sk}, x) : \begin{array}{c} (\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ \widetilde{\mathsf{sk}} \leftarrow \mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Weak Pseudorandomness A watermarkable PRF satisfies weak pseudorandomness if for all PPT $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathcal{A}^{\mathcal{O}(\mathsf{sk},\cdot)}(x, y_b) = b : \; x \leftarrow \{0,1\}^\ell, y_0 = \mathsf{Eval}(\mathsf{sk}, x), y_1 \leftarrow \{0,1\}^\ell, b \leftarrow \{0,1\} \; \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

where the oracle $\mathcal{O}(\mathsf{sk}, \cdot)$ samples a uniformly random $x \leftarrow \{0,1\}^\ell$ upon every query and outputs $(x, \mathsf{Eval}(\mathsf{sk}, \cdot))$.

We can also consider the equivalent notion where $\mathcal{A}$ is given an oracle $\mathcal{O}(\mathsf{sk}, \cdot)$ which either computes $\mathsf{PRF}.\mathsf{Eval}(\mathsf{sk}, \cdot)$ on random inputs or computes a real random function that samples random input together with a random output. There will be no challenge input and $\mathcal{A}$ outputs a bit indicating which oracle it is given. Similarly for the security below.

Variant of Weak Pseudorandomness Game To be better compatible with our unremovability definition, we consider the following variant of the above weak PRF game, described in a 2-stage fasion: in stage 1, $\mathcal{A}^1$ is allowed to query *adaptively* $\mathcal{O}(\mathsf{sk}, \cdot)$ on any input of its own choice. Entering stage 2, it can only query $\mathcal{O}(\mathsf{sk}, \cdot)$ in a way that each input is sampled at random. Then $\mathcal{A}^2$ is fed with challenge input (x, y_b) where $x \leftarrow \{0,1\}^\ell, y_0 = \mathsf{Eval}(\mathsf{sk}, x), y_1 \leftarrow \{0,1\}^\ell, b \leftarrow \{0,1\}$ and needs to output the correct b.

γ-Unremovability of weak PRF The γ-Unremovability for a watermarkable implementation of a weak PRF scheme says, for all PPT admissible stateful adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathsf{Extract}(\mathsf{xk}, C) \notin \mathcal{Q} \wedge C \text{ is } \gamma\text{-good} \;:\; \begin{array}{c} (\mathsf{sk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ C \leftarrow \mathcal{A}^{\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)}(1^\lambda) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

$\mathcal{O}(\mathsf{sk}, \cdot)$ samples a uniformly random $x \leftarrow \{0,1\}^\ell$ upon every query and outputs $(x, \mathsf{Eval}(\mathsf{sk}, \cdot))$.

$\mathcal{Q}$ is the set of marks queried by $\mathcal{A}$ and C is a PPT admissible, stateful γ-good adversary in the security game $G_{\mathsf{PRF}}(\mathsf{sk}, \cdot)$ if:

$$\Pr\left[C^{\mathcal{O}(\mathsf{sk}, \cdot)}(x, y_b) = b : \; x \leftarrow \{0,1\}^\ell, y_0 = \mathsf{Eval}(\mathsf{sk}, x), y_1 \leftarrow \{0,1\}^\ell, b \leftarrow \{0,1\} \right] \geq \frac{1}{2} + \gamma.$$

Remark 5.1. *In the above setting, the oracle $\mathcal{O}(\cdot)$ will sample a random r and output $r, \mathcal{O}(r)$.*

We can consider an even slightly stronger notion of weak PRF: in the "fully adaptive-query" setting, $\mathcal{A}, C$ are allowed to make any query to the real Eval *oracle; only the challenge input x is sampled uniformly at random. Then C is asked o distinguish between $y_0 = \mathsf{Eval}(\mathsf{sk}, x), y_1 \leftarrow \{0,1\}^\ell$.*

Note that [GKWW21] consider the first notion so that they can realize public extraction only by letting the extraction key sample input-output pairs. This notion also suffices to prove watermarkable CPA security. Both cann be constructed in [GKWW21] .

5.3 Definition: Watermarkable Implementation of CPA-secure SKE

A watermarkable implementation of a SKE scheme consists of the following algorithms:

$\mathsf{WMSetup}(1^\lambda) \rightarrow (\mathsf{sk}, \mathsf{mk}, \mathsf{xk})$: on input security parameter, outputs a secret key sk, marking key mk, extraction key xk.

$\mathsf{Enc}(\mathsf{sk}, m) \rightarrow \mathsf{ct}$: on input secret key sk and message m, outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \rightarrow m$: on input secret key sk and ciphertext ct outputs a message m.

$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau)$: on marking key mk, secret key sk, a message τ, output a marked key sk_τ.

$\mathsf{Extract}(\mathsf{xk}, C)$: on extraction key sk and program C, output a mark $\tau \in \mathcal{M}_\tau$ or $\perp$.

The correctness property is the same as the correctness property of a plain SKE scheme and we combine it with the functionality preserving property it due to it being standard.

Correctness and Functionality Preserving A watermarkable CPA-secure SKE is functionality preserving if then there exists a negligible function $\mathsf{negl}(\lambda)$ ssuch that for all $\lambda \in \mathbb{N}, , m \in \mathcal{M}, \tau \in \mathcal{M}_\tau$:

$$\Pr\left[\mathsf{Dec}(\tilde{\mathsf{sk}}, \mathsf{ct}') = m \wedge \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ \tilde{\mathsf{sk}} \leftarrow \mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, m), \mathsf{ct}' \leftarrow \mathsf{Enc}(\mathsf{sk}_\tau, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

CPA-security A watermarkable CPA secure SKE scheme satisfies CPA security if for all PPT admissible stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathcal{A}_2^{\mathsf{Enc}(\mathsf{sk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c} (m_0, m_1) \leftarrow \mathcal{A}_1^{\mathsf{Enc}(\mathsf{sk},\cdot)} \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{sk}, m_b), b \leftarrow \{0,1\} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

γ-Unremovability of CPA-secure SKE The γ-Unremovability for a watermarkable CPA secure SKE scheme says, for all PPT admissible stateful adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathsf{Extract}(\mathsf{xk}, C) \notin \mathcal{Q} \wedge C \text{ is } \gamma\text{-good} : \begin{array}{c} (\mathsf{sk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ C \leftarrow \mathcal{A}^{\mathsf{Mark}(\mathsf{mk},\mathsf{sk},\cdot)}(1^\lambda) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

where $\mathcal{Q}$ is the set of marks queried by $\mathcal{A}$ and $C = (C_1, C_2)$ is said to be a (PPT admissible, stateful) γ-good program in the security game $G^2_{CCA}(\mathsf{sk}, \cdot)$, more specifically:

$$\Pr\left[C_2^{\mathsf{Enc}(\mathsf{sk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c} (m_0, m_1) \leftarrow C_1^{\mathsf{Enc}(\mathsf{sk},\cdot)} \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{sk}, m_b), b \leftarrow \{0,1\} \end{array}\right] \geq \frac{1}{2} + \gamma.$$

Remark 5.2. *The above definition is equivalent to letting A output (m_0, m_1) and making (m_0, m_1) as the auxiliary input* aux *for G_{CCA} and* $\mathsf{Extract}$. *Because we can view any distribution over (m_0, m_1) used by C as a convex combination of different message pairs $\{(m_0, m_1)_i\})_i$ and its corresponding strategy such that the overall winning probability is $1/2 + \gamma$. Thus, we can let $\mathcal{A}$ pick the message-pair and corresponding strategy with the largest winning probability and hardcode them into C instead. The other way of implication is easy to see.*

This also applies to the CCA security game.

5.4 Watermarking-Compatible Reduction

We show that the textbook construction of CPA encryption from weak PRF is watermarking-compatible. Given a PRF scheme that satisfies weak PRF security $\mathsf{PRF} = (\mathsf{PRF.KeyGen}, \mathsf{PRF.Eval})$, where input and output lengths are ℓ and key length is n.

The syntax of PRF and CPA-security is standard and we refer them to the watermarkable definitions Section 5.3, Section 5.3, by replace $\mathsf{WMSetup}$ with KeyGen and removing $\mathsf{Extract}$, Mark algorithms.

Plain Construction of CPA-secure Encryption from weak PRF

- $\mathsf{KeyGen}(1^\lambda)$: $\mathsf{sk} \leftarrow \mathsf{PRF.KeyGen}(\lambda)$
- $\mathsf{Enc}(\mathsf{sk}, m \in \{0,1\}^\ell) \rightarrow \mathsf{ct}$: samples $r \leftarrow \{0,1\}^\ell$; output $\mathsf{ct} \leftarrow (r, \mathsf{PRF.Eval}(\mathsf{sk}, r) \oplus m)$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: parse $\mathsf{ct} := (r, \mathsf{ct}')$; compute $m := \mathsf{PRF.Eval}(\mathsf{sk}, r) \oplus \mathsf{ct}'$.

Watermarking-Compatible Reduction We briefly recall the textbook reduction for the above construction, in the language of watermarking-compatible.

Claim 5.3. *The CPA-secure SKE to weak PRF pseudorandomness reduction is a watermarking compatible reduction.*

PRF pseudorandomness adversary $\mathcal{A}_{\mathsf{PRF}}$ simulates CPA game for adversary $\mathcal{A}$ as follows:

- There is no public key and public evaluation algorithm $\mathsf{PubEval}$ in the scheme, so $\mathcal{A}_{\mathsf{PRF}}$ and $\mathcal{A}$ will not receive information regarding public key pk.
- First consider stage 1: For any encryption query from $\mathcal{A}^1$, $\mathcal{A}^1_{\mathsf{PRF}}$ will simulate the response as follows: it will query the oracle $\mathcal{O}(\mathsf{sk}, \cdot)$ in the security game G_{PRF}, where $\mathcal{O}(\cdot)$ outputs $\mathsf{PRF.Eval}(\mathsf{sk}, x)$ on any query x.
- After entering stage 2, for any encryption query from $\mathcal{A}^2$, $\mathcal{A}^2_{\mathsf{PRF}}$ will still simulate the response as the above. Recall that $\mathcal{A}^2_{\mathsf{PRF}}$ will not get to see any queries made in stage 1.
- Note that $\mathcal{A}^2_{\mathsf{PRF}}$'s access to $\mathcal{O}(\cdot)$ is changed to non-adaptive. $\mathcal{O}(\cdot)$ performs the following: upon every query, sample a fresh r, output $\mathsf{PRF.Eval}(\mathsf{sk}, r)$.
- In the challenge phase, $\mathcal{A}^2_{\mathsf{PRF}}$ will use the challenge input from the weak PRF challenger: $r^* \leftarrow \{0,1\}^\ell, y^* \in \{0,1\}^\ell$ where y^* is either $\mathsf{PRF.Eval}(\mathsf{sk}, r^*)$ or uniformly random.
- $\mathcal{A}^2$ sends in challenge messages (m_0, m_1); $\mathcal{A}^2_{\mathsf{PRF}}$ flips a coin $b \leftarrow \{0,1\}$ and sends $\mathsf{ct} = (r^*, y^* \oplus m_b)$ to $\mathcal{A}^2$.
- If $\mathcal{A}^2$ guesses the correct b, then $\mathcal{A}^2_{\mathsf{PRF}}$ output 0, for y^* is $\mathsf{PRF.Eval}(\mathsf{sk}, r^*)$", else $\mathcal{A}^2_{\mathsf{PRF}}$ output 1, for "y^* is uniformly random".

In the above reduction, $\mathcal{A}^2_{\mathsf{PRF}}$'s reduction function $f(\mathsf{out}, b, \mathcal{Q}, \mathsf{inp})$ is: ignore $\mathcal{Q}, \mathsf{inp}$, check if $\mathcal{A}^2$'s output out is equal to the randomness b used in preparing the challenge input, if yes output 0 else 1.

We denote the security game for weak PRF as G_{PRF} and for CPA encryption as G_{CPA}; we also denote the advantage of $\mathcal{A}$ in G_{CPA} as Adv_{CPA} and advantage of $\mathcal{A}_{\mathsf{PRF}}$ as $\mathsf{Adv}_{\mathsf{PRF}}$. We can observe that $\mathsf{Adv}_{\mathsf{PRF}} \geq \mathsf{Adv}_{CPA}/2 - \frac{q}{2^\ell}$ where q is the number of queries made by $\mathcal{A}$.

5.4.1 Security of Watermarkable Implementation for CPA-secure SKE

Theorem 5.4. *Assuming that LWE is secure, then there exists secure single-key watermarkable implementation of CPA-secure SKE with private tracing.*

Assuming that there exists indistinguishability obfuscation, then there exists secure collusion-resistant watermarkable implementation of CPA-secure SKE with public tracing.

Theorem 5.5. *[[GKWW21, MW22]] Assuming that LWE is secure, then there exists secure collusion-resistant watermarkable implementation of weak PRF with private extraction.*

Assuming that there exists indistinguishability obfuscation, then there exists secure collusion-resistant watermarkable implementation of weak PRF with public extraction.

Note that both schemes above satisfy the properties we require in Section 4.1, including: extraction key of the scheme can be used to simulate the weak PRF pseudorandomness security game *perfectly*; the extraction procedure satisfies the extraction syntax [10].

Remark 5.6. *In [GKWW21, MW22], the watermarkable implementation of weak PRF is called traceable PRF. Their exact extraction procedure does not satisfy our requirement that the adversarial circuit C can*

[10]For conveniece, in the rest of this work, when we cite an existing watermarking scheme as a "watermarking implementation of a primitive", then the scheme satisfies all properties defined in Section 4.1.

make $\mathsf{PRF.Eval}(\mathsf{sk}, \cdot)$ *queries during the extraction/tracing procedure. But it can be easily modified to satisfy this property: their tracing key has the capability to sample from the input-output space of the PRF, and thus can answer weak PRF queries.*

Lemma 5.7. *Assuming that there exists collusion-resistant (resp. single-key) secure watermarkable implementation of weak PRF with public(resp. private) extraction, then there exists secure collusion-resistant (resp. single-key) watermarkable implementation of CPA-secure SKE with public(resp. private)extraction.*

To prove Lemma 5.7, we present the construction as below.

Watermarking Implementation of CPA-secure Encryption Given a watermarkable implementation of weak PRF $(\mathsf{wPRF.WMSetup}, \mathsf{wPRF.Eval}, \mathsf{wPRF.Mark}, \mathsf{wPRF.Extract})$, we can construct a watermarkable implementation of CPA-secure SKE as follows:

$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{sk}, \mathsf{xk}, \mathsf{mk})$: compute $(\mathsf{sk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{wPRF.WMSetup}(1^\lambda)$
$\mathsf{Enc}(\mathsf{sk}, m \in \{0,1\}^\ell)$: samples $r \leftarrow \{0,1\}^\ell$; output $\mathsf{ct} \leftarrow (r, \mathsf{wPRF.Eval}(\mathsf{sk}, r) \oplus m)$.
$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: parse $\mathsf{ct} := (r, \mathsf{ct}')$; compute $m := \mathsf{wPRF.Eval}(\mathsf{sk}, r) \oplus \mathsf{ct}'$.
$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau \in \mathcal{M}_{\mathsf{Mark}})$:

1. parse $\mathsf{mk} = \mathsf{wPRF.mk}$
2. output $\mathsf{sk}_\tau \leftarrow \mathsf{wPRF.Mark}(\mathsf{mk}, \mathsf{sk}, \tau)$;

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

1. On input circuit C and $\mathsf{xk} = \mathsf{wPRF.xk}$;
2. Treat C as the stage-2 adversary $\mathcal{A}_2$ in the watermarking-compatible reduction from CPA-security to weak PRF security to create circuit C_{PRF} (C_{PRF} has black-box access to C) as a stage-2 reduction $\mathcal{A}^2_{\mathsf{PRF}}$ algorithm from CPA-security to the weak PRF security.
3. compute $\tau/\bot \leftarrow \mathsf{wPRF.Extract}(\mathsf{xk}, C_{\mathsf{PRF}})$; if the mark extracted is not $\bot$, abort and output the mark.

Proof. We describe the reduction in the watermarking-compatible reduction language.

Suppose there exists a PPT adversary $\mathcal{A}_{\mathsf{Mark}}$ for the γ-unremovability security of the above watermarkable CPA encryption, we can consider the adversary $\mathcal{A}_{\mathsf{Mark}}$ to be the stage-1 adversary $\mathcal{A}^1$ in the CPA-security game: $\mathcal{A}_{\mathsf{Mark}}$ gets to query the marking oracle $\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)$. The reduction $\mathcal{B}$ which is an adversary in the strong unremovability security game of the watermarkable implementation of weak PRF, can simulate both the answers to the marking queries by querying the PRF's marking oracle.

Then $\mathcal{A}_{\mathsf{Mark}}$ outputs a circuit C; during the $\mathsf{Extract}(\mathsf{xk}, \cdot)$ algorithm. C will be treated as a stage-2 adversary in the CPA-security game; with black-box access to C, circuit C_{PRF} will act like a stage-2 reduction from CPA-security to weak PRF security. Since $\mathsf{wPRF.Extract}(\mathsf{wPRF.sk}, \cdot)$ algorithm will perfectly simulate the weak PRF security game on its input circuit, C_{PRF} will be able to answer C's queries and use C's output to finish the reduction.

The reduction B will create a circuit C' with black-box use of C the same way as the above C^C_{PRF} and output C' in the $\gamma/2$-unremovability security game of the watermarkable implementation of weak PRF.

By our assumption, C should satisfy that $\Pr[G_{CPA}(\mathsf{sk}, C) = 1] \geq \frac{1}{2} + \gamma$ and $\Pr[\mathsf{Extract}(\mathsf{xk}, C) \notin \mathcal{Q}] \geq \epsilon$ for some non-negligible ϵ. By the design of our Extract algorithm, it must be the case that

$\Pr[\mathsf{wPRF.Extract}(\mathsf{wPRF.xk}, C_{\mathsf{PRF}}) \notin \mathcal{Q}] \geq \epsilon$. Since the circuit C' created by $\mathcal{B}$ is exactly the same as C_{PRF} and the set of marking queries $\mathcal{Q}$ made by $\mathcal{B}$ to the marking oracle is the set as $\mathcal{A}_{\mathsf{Mark}}$'s, we must have that $\Pr[\mathsf{wPRF.Extract}(\mathsf{wPRF.xk}, C') \notin \mathcal{Q}] \geq \epsilon$.

Meanwhile, by the property of the reduction, we have that

$$\Pr[G_{\mathsf{PRF}}(\mathsf{wPRF.sk}, C') = 1] \geq \frac{1}{2} + \gamma/3$$

. Therefore, $\mathcal{B}$ breaks $\gamma/3$-unremovability for watermarkable implementation of weak PRF for some non-negligible γ.

□

5.5 Watermarkable CCA-secure Encryption from Watermarkable MAC and PRF

In this section, we show an example of obtaining a watermarkable implementation of a CCA secure secret-key encryption scheme from a watermarkable implementation of weak PRF and a watermarkable implementation of MAC(or signature).

The security of watermarkable implementation of signatures (and therefore MAC) satisfying our requirements in Section 4.1 can be built from a modification of the watermarkable signature scheme in [GKM+19].

5.5.1 Watermaking-Compatible Reduction of CCA Security to MAC and weak PRF

We first give the "plain" construction from weak PRF and MAC to CCA-secure SKE and show that the reduction is watermarking-compatible.

Given a PRF scheme that satisfies weak PRF security $\mathsf{PRF} = (\mathsf{PRF.KeyGen}, \mathsf{PRF.Eval})$, where input and output lengths are ℓ and key length is n and a MAC scheme $\mathsf{MAC} = (\mathsf{MAC.KeyGen}, \mathsf{MAC.Sign}, \mathsf{MAC.Verify})$.

Construction

- $\mathsf{KeyGen}(1^\lambda)$: $\mathsf{PRF.sk} \leftarrow \mathsf{PRF.KeyGen}(\lambda)$, $\mathsf{MAC.sk} \leftarrow \mathsf{MAC.KeyGen}(\lambda)$; output $\mathsf{sk} = (\mathsf{PRF.sk}, \mathsf{MAC.sk})$.
- $\mathsf{Enc}(\mathsf{sk}, m \in \{0,1\}^\ell) \rightarrow \mathsf{ct}$:
  - parse $\mathsf{sk} = (\mathsf{PRF.sk}, \mathsf{MAC.sk})$;
  - samples $r \leftarrow \{0,1\}^\ell$;
  - compute $\mathsf{ct}_0 \leftarrow (r, \mathsf{PRF.Eval}(\mathsf{sk}, r) \oplus m)$;
  - compute $\sigma = \mathsf{MAC.Sign}(\mathsf{MAC.sk}, \mathsf{ct}_0)$
  - output $\mathsf{ct} = (\mathsf{ct}_0, \sigma)$,
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$:
  - parse $\mathsf{ct} = (\mathsf{ct}_0, \sigma)$ and $\mathsf{sk} = (\mathsf{PRF.sk}, \mathsf{MAC.sk})$;
  - If $\mathsf{MAC.Verify}(\mathsf{MAC.sk}, \mathsf{ct}_0, \sigma) = 1$ continue, else abort and output $\perp$.
  - parse $\mathsf{ct}_0 := (r, \mathsf{ct}')$; compute $m := \mathsf{PRF.Eval}(\mathsf{PRF.sk}, r) \oplus \mathsf{ct}'$.
  - output m.

Claim 5.8. *The CCA-secure SKE to weak PRF pseudorandomnes's reduction and the CCA-secure SKE to EUF-CMA-security of MACs reduction are both watermarking compatible reductions.*

Watermarking-Compatible Reduction We describe the security reduction of the above construction in the language of watermarking-compatible.

First consider a reduction for MAC security, $\mathcal{A}_{\mathsf{MAC}}$ simulates CPA game for adversary $\mathcal{A}$ as follows:

- There is no public key and public evaluation algorithm $\mathsf{PubEval}$ in the scheme, so $\mathcal{A}_{\mathsf{PRF}}$ and $\mathcal{A}$ will not receive information regarding public key pk.
- $\mathcal{A}_{\mathsf{MAC}}$ samples its own PRF secret key $\mathsf{PRF.sk}$.
- For stage 1: For any encryption query from $\mathcal{A}^1$, $\mathcal{A}^1_{\mathsf{MAC}}$ will simulate the response as follows: it will compute the ct_0 part of ciphertext and then query the signing oracle $\mathsf{MAC.Sign}(\mathsf{MAC.sk}, \cdot)$ in the security game G_{MAC}. For any decryption query, $\mathcal{A}_{\mathsf{MAC}}$ will query the verification oracle $\mathsf{MAC.Verify}(\mathsf{MAC.sk}, \cdot)$.
- After entering stage 2, for any encryption or decryption query from $\mathcal{A}^2$, $\mathcal{A}^2_{\mathsf{MAC}}$ will still simulate the response as the above. Recall that $\mathcal{A}^2_{\mathsf{MAC}}$ will not get to see any queries made in stage 1 and $\mathcal{A}^2_{\mathsf{MAC}}$ will record all queries from $\mathcal{A}_2$.
- In the challenge phase, $\mathcal{A}^2$ sends in challenge messages (m_0, m_1); $\mathcal{A}^2_{\mathsf{MAC}}$ flips a coin $b \leftarrow \{0, 1\}$ and sends the encryption of m_b to $\mathcal{A}^2$.
- A^2_{MAC} continues to simulate the encryption and decryption oracles for $\mathcal{A}^2$.
- In the end, $\mathcal{A}^2_{\mathsf{MAC}}$ will look up $\mathcal{A}^2$'s queries: find a decryption query $\mathsf{ct} = (\mathsf{ct}_0, \sigma)$ such that it has never been the output of an encryption query and the decryption oracle did not output $\perp$ on this query. $\mathcal{A}^2_{\mathsf{MAC}}$ output (ct_0, σ) as its forgery.

If such a decryption query exists in the end, then $\mathcal{A}_{\mathsf{MAC}}$ break the EUF-CMA-security of MAC (Otherwise, we will do a reduction to weak PRF).

In the above reduction, $\mathcal{A}^2_{\mathsf{MAC}}$'s reduction function $f(\mathsf{out}, r, \mathcal{Q}, \mathsf{inp})$ is: ignore $\mathsf{out}, \mathsf{inp}, r$, check if $\mathcal{A}^2$'s queries $\mathcal{Q}$ contain a decryption query where the input of this query is not from the output of an encryption query; if found, output this query.

Since such a ciphertext pair was never output by the encryption oracle, then it means that the reduction $\mathcal{A}_{\mathsf{MAC}}$ never queried the corresponding message-signature out of the decryption oracle. Thus $\mathcal{A}_{\mathsf{MAC}}$ can use it to break EUF-CMA-security of MAC.

If no such a decryption query exists, we can build a reduction to break the weak pseudorandomness of PRF. The reduction to weak PRF security will be similar to Section 5.1, except that the reduction will sample its own MAC keys to simulate the oracles. Since we have discussed this example in sufficient detail in the technical overview, we omit repeating the details here.

5.5.2 Definition: Watermarkable Implementation of CCA-secure SKE

We give the CCA2 security and unremovability security definition for CCA2-secure SKE. The other definitions (correctness, functionality preserving) follow naturally from the correctness and previously discussed CPA secure SKE.

CCA2-security A watermarkable CPA secure SKE scheme satisfies CPA security if for all PPT admissible stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathcal{A}_2^{\mathsf{Enc}(\mathsf{sk},\cdot),\mathsf{Dec}(\mathsf{sk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c}(m_0, m_1) \leftarrow \mathcal{A}_1^{\mathsf{Enc}(\mathsf{sk},\cdot),\mathsf{Dec}(\mathsf{sk},\cdot)} \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{sk}, m_b), b \leftarrow \{0,1\}\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

where $\mathcal{A}_2$ can only make decryption queries on $\mathsf{ct} \neq \mathsf{ct}_b$.

γ-Unremovability of CCA2-secure SKE The γ-Unremovability for a watermarkable CCA2 secure SKE scheme says, for all PPT admissible stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathsf{Extract}(\mathsf{xk}, C) \notin \mathcal{Q} \wedge C \text{ is } \gamma\text{-good} : \begin{array}{c} (\mathsf{sk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ C \leftarrow \mathcal{A}^{\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)}(1^\lambda) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

where $\mathcal{Q}$ is the set of marks queried by $\mathcal{A}$ and $C = (C_1, C_2)$ is a PPT admissible, stateful γ-good adversary in the security game $G_{CCA}(\mathsf{sk}, \cdot)$, more specifically:

$$\Pr\left[C_2^{\mathsf{Enc}(\mathsf{sk},\cdot),\mathsf{Dec}(\mathsf{sk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c} (m_0, m_1) \leftarrow C_1^{\mathsf{Enc}(\mathsf{sk},\cdot),\mathsf{Dec}(\mathsf{sk},\cdot)} \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{sk}, m_b), b \leftarrow \{0,1\} \end{array}\right] \geq \frac{1}{2} + \gamma.$$

C is admissible if it only make queries to the oracle on $\mathsf{ct} \neq \mathsf{ct}_b$.

We refer the watermarkable MAC (and signatures) definition to Appendix D.2.

5.5.3 Security of Watermarkable Implementation for CCA-secure SKE

Theorem 5.9. *Assuming LWE, then there exists secure bounded collusion-resistant watermarkable implementation of CCA-secure SKE with private extraction.*

Assuming that there exists indistinguishability obfuscation, then there exists secure bounded collusion-resistant watermarkable implementation of CCA-secure SKE with private tracing.

Tha above theorem is based on Theorem 5.5 and the following:

Theorem 5.10. *[Revision of [GKM+19]] Assuming that there exists secure OWFs, then there exists secure watermarkable implementation of digital signatures (and MAC) with private extraction and bounded collusion resistance.*

Remark 5.11. *The construction for the watermarkable digital signature scheme in the original [GKM+19] does not exactly satisfy our requirements in Section 4.1. But we will show that a minor modification of the scheme will suffice.*

Lemma 5.12. *Assuming that there exists collusion-resistant (resp. single-key) secure watermarkable implementation of weak PRF and MAC, then there exists secure collusion-resistant (resp. single-key) watermarkable implementation of CCA-secure SKE with private extraction.*

Remark 5.13. *[GKWW21, MW22]can give us a watermarkable weak PRF scheme that suffices for our use, though their definition of weak PRF allows only non-adaptive queries of random r and its evalaution. We need the weak PRF to allow adaptive queries to be used in CCA construction. But their prvate-key watermarkable PRF scheme already satisfies this stronger definition, by having the PRF evaluation key included in the tracing key.*

To prove Lemma 5.12, we present the construction as below.

Watermarking Implementation of CCA-secure Encryption Given a watermarkable implementation of weak PRF (wPRF.WMSetup, wPRF.Eval, wPRF.Mark, wPRF.Extract) and one of MAC (wMAC.WMSetup, wMAC.Sign, wMAC.Verify, wMAC.Mark, wMAC.Extract), we can construct a watermarkable implementation of CCA-secure SKE as follows:

```
$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{sk}, \mathsf{xk}, \mathsf{mk})$:
    1. compute $(\mathsf{wPRF.sk}, \mathsf{wPRF.xk}, \mathsf{wPRF.mk}) \leftarrow \mathsf{wPRF.WMSetup}(1^\lambda)$; compute $(\mathsf{wMAC.sk}, \mathsf{wMAC.xk}, \mathsf{MAC.mk}) \leftarrow \mathsf{wMAC.WMSetup}(1^\lambda)$
    2. output $\mathsf{sk} = (\mathsf{wPRF.sk}, \mathsf{wMAC.sk})$; $\mathsf{mk} = (\mathsf{wPRF.mk}, \mathsf{wMAC.mk})$; $\mathsf{xk} = (\mathsf{wPRF.xk}, \mathsf{wMAC.xk})$
$\mathsf{Enc}(\mathsf{sk}, m \in \{0,1\}^\ell)$:
    1. parse $\mathsf{sk} = (\mathsf{wPRF.sk}, \mathsf{wMAC.sk})$;
    2. samples $r \leftarrow \{0,1\}^\ell$;
    3. compute $\mathsf{ct}_0 \leftarrow (r, \mathsf{wPRF.Eval}(\mathsf{wPRF.sk}, r) \oplus m)$;
    4. compute $\sigma = \mathsf{wMAC.Sign}(\mathsf{wMAC.sk}, \mathsf{ct}_0)$
    5. output $\mathsf{ct} = (\mathsf{ct}_0, \sigma)$,
$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$:
    1. parse $\mathsf{ct} = (\mathsf{ct}_0, \sigma)$ and $\mathsf{sk} = (\mathsf{wPRF.sk}, \mathsf{wMAC.sk})$;;
    2. If $\mathsf{wMAC.Verify}(\mathsf{wMAC.sk}, \mathsf{ct}_0, \sigma) = 1$ continue, else abort and output $\perp$.
    3. parse $\mathsf{ct}_0 := (r, \mathsf{ct}')$; compute $m := \mathsf{wPRF.Eval}(\mathsf{wPRF.sk}, r) \oplus \mathsf{ct}'$.
    4. output $m$.
$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau \in \mathcal{M}_{\mathsf{Mark}})$:
    1. parse $\mathsf{mk} = (\mathsf{wPRF.mk}, \mathsf{wMAC.mk})$
    2. compute $\mathsf{sk}_{\mathsf{PRF},\tau} \leftarrow \mathsf{wPRF.Mark}(\mathsf{wPRF.mk}, \mathsf{wPRF.sk}, \tau)$;
       $\mathsf{sk}_{\mathsf{MAC},\tau} \leftarrow \mathsf{wMAC.Mark}(\mathsf{wMAC.mk}, \mathsf{wMAC.sk}, \tau)$;
    3. output $(\mathsf{sk}_{\mathsf{PRF},\tau}, \mathsf{sk}_{\mathsf{MAC},\tau})$
$\mathsf{Extract}(\mathsf{xk}, C)$ :
    1. On input circuit $C$ and $\mathsf{xk} = (\mathsf{wPRF.xk}, \mathsf{wMAC.xk})$;
    2. Initialize empty tuple $\vec{\tau}$.
    3. Let $P_1 := \mathsf{PRF}$ and $P_2 := \mathsf{MAC}$; For $i = 1, 2$:
        (a) Treat $C$ as the stage-2 adversary $\mathcal{A}_2$ in the watermarking-compatible reduction from CCA-security to $P_i$ security; create a circuit $C_{P_i}^C$ (i.e. $C_{P_i}$ has black-box access to $C$) as a stage-2 reduction algorithm $\mathcal{A}_i^2$ from CCA-security game to $P_i$'s security game.
        (b) compute $\tau/\perp \leftarrow \mathsf{wP_i.Extract}(\mathsf{wP_i.xk}, C_{P_i}^C)$; if the mark extracted is not $\perp$, add it to $\vec{\tau}$.
    4. output $\vec{\tau}$
```

Now we prove the unremovability security of the watermarkable implementation of the CCA secure encryption scheme.

Proof. We describe the reduction in the watermarking-compatible reduction language.

Suppose there exists a PPT adversary $\mathcal{A}_{\mathsf{Mark}}$ for the γ-unremovability security of the above watermarkable CCA encryption, we can consider the adversary $\mathcal{A}_{\mathsf{Mark}}$ to be the stage-1 adversary

$\mathcal{A}^1$ in the CCA-security game: $\mathcal{A}_{\mathsf{Mark}}$ gets to query the marking oracle $\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)$, which is a leakage on the secret key $\{\mathsf{wPRF.sk}, \mathsf{wMAC.sk}\}$.

Let us denote $P_1 = \mathsf{PRF}; P_2 = \mathsf{wMAC}$. The reduction $\mathcal{B}_i$ which is an adversary in the $\gamma/3$-unremovability security game of the watermarkable implementation of primitive P_i, can simulate both the answers to the marking queries by querying the challenger.

Then $\mathcal{A}_{\mathsf{Mark}}$ outputs a circuit C; during the $\mathsf{Extract}(\mathsf{xk}, \cdot)$ algorithm. C will be treated as a stage-2 adversary in the CCA-security game; with black-box access to C. circuit C_{P_i} will act like a stage-2 reduction from CCA-security to P_i's security. Since $\mathsf{wP_i.Extract}(\mathsf{wP_i.xk}, \cdot)$ algorithm will perfectly simulate the security game G_{P_i} on its input circuit, C_{P_i} will be able to answer C's queries and use C's output to finish the reduction.

Since we are assuming that $\Pr[\mathsf{Extract}(\mathsf{xk}, C) \notin \mathcal{Q}] \geq \epsilon$, for some non-negligible ϵ. By the design of our $\mathsf{Extract}$ algorithm, it must be that for both $i \in \{1, 2\}$, $\Pr[\mathsf{wP_i.Extract}(\mathsf{wP_i.xk}, C_i^C) \notin \mathcal{Q}] \geq \epsilon$.

We now consider two cases. For any $\mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot),\mathsf{Dec}(\mathsf{sk},\cdot),\mathsf{Mark}(\mathsf{mk},\mathsf{sk},\cdot)}(\mathsf{pk})$ producing a γ-good C such that $\Pr[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C) \notin \mathcal{Q}] \geq \epsilon$, one of the following cases must hold:

- **Case 1**: In case 1, with some non-negligible probability, C_1 is γ_1-good in the game $G_{\mathsf{PRF}}(\mathsf{wPRF.sk}, \cdot)$ for some non-negligible γ_1. In other words, during the execution of $\mathsf{Extract}(\mathsf{xk}, C)$, C makes decryption queries on ciphertexts that all come from the encryption queries.
  The reduction B_1 will create a circuit C_1' with black-box use of C the same way as the circuit $C_{P_1}^C$ created in the extraction algorithm and output C_1' in the γ_1-unremovability security game of the watermarkable implementation of weak PRF.
  Since the circuit C' created by $\mathcal{B}_1$ is exactly the same as C_{PRF}^C and the set of marking queries $\mathcal{Q}$ made by $\mathcal{B}$ to the marking oracle is the set as $\mathcal{A}_{\mathsf{Mark}}$'s, we must have that $\Pr[\mathsf{wPRF.Extract}(\mathsf{wPRF.xk}, C_1') \notin \mathcal{Q}] \geq \epsilon$.
  Meanwhile, by the property of the reduction, we have that $\Pr[G_{\mathsf{PRF}}(\mathsf{wPRF.sk}, C_1') = 1] \geq \frac{1}{2} + \gamma_1$ for some non-negligible γ_1. Therefore, $\mathcal{B}$ breaks γ_1-unremovability for watermarkable implementation of weak PRF.
- **Case 2**: In case 2, with some non-negligible probability, C_2 is γ_2-good in the game $G_{\mathsf{MAC}}(\mathsf{wMAC.sk}, \cdot)$ for some non-negligible γ_2.
  In other words, during the execution of $\mathsf{Extract}(\mathsf{xk}, C)$, C makes decryption queries on ciphertexts that not only come from the encryption queries.
  The reduction B_2 will create a circuit C_2' with black-box use of C the same way as the circuit C_{MAC}^C created in the extraction algorithm and output C_2' in the γ_2-unremovability security game of the watermarkable implementation of weak PRF.
  Since the circuit C' created by $\mathcal{B}_2$ is exactly the same as C_{MAC} and the set of marking queries $\mathcal{Q}$ made by $\mathcal{B}_2$ to the marking oracle is the set as $\mathcal{A}_{\mathsf{Mark}}$'s, we must have that $\Pr[\mathsf{wMAC.Extract}(\mathsf{wMAC.xk}, C_2') \notin \mathcal{Q}] \geq \epsilon$.
  Meanwhile, by the property of the reduction, we have that $\Pr[G_{\mathsf{MAC}}(\mathsf{wMAC.sk}, C_2') = 1] \geq \frac{1}{2} + \gamma_2$ for some non-negligible γ_2. Therefore, $\mathcal{B}_2$ breaks γ_2-unremovability for watermarkable implementation of MAC.

By the property of the watermarking-compatible reduction (shown in Section 5.5.1), one of the above two cases must hold. $\square$

6 Watermarkable CCA-secure PKE from Watermarkable Identity-based Encryption and Strong One-Time Signature

In this section, we give an example of how to achieve watermarkable implementation of CCA-secure PKE, where type-2 reduction is used. The watermarkable CCA-secure PKE scheme is based on the construction from selectively secure IBE and strong one-time signatures ([BCHK07])

6.1 Definition: Watermarkable CCA2-secure PKE

A watermarkable implementation of a SKE scheme consists of the following algorithms:

$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk}, \mathsf{mk}, \mathsf{xk})$: on input security parameter, outputs a secret key sk, public key pk, marking key mk, extraction key xk.
$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$: on input public key pk and message m, outputs a ciphertext ct.
$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$: on input secret key sk and ciphertext ct outputs a message m.
$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau)$: on marking key mk, secret key sk, a message τ, output a marked key sk_τ.
$\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C)$: on extraction key sk, public key pk and program C, output a mark $\tau \in \mathcal{M}_\tau$ or $\perp$.

The correctness property is the same as the correctness property of a plain PKE scheme and we combine it with the functionality preserving property it due to it being standard.

Correctness and Functionality Preserving A watermarkable PKE is functionality preserving if then there exists a negligible function $\mathsf{negl}(\lambda)$ ssuch that for all $\lambda \in \mathbb{N}, , m \in \mathcal{M}, \tau \in \mathcal{M}_\tau$:

$$\Pr\left[\mathsf{Dec}(\tilde{\mathsf{sk}}, \mathsf{ct}) = m \land \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ \widetilde{\mathsf{sk}} \leftarrow \mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

CCA2-security A watermarkable PKE scheme satisfies CCA2 security if for all PPT admissible stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathcal{A}_2^{\mathsf{Dec}(\mathsf{sk}, \cdot)}(\mathsf{st}, \mathsf{ct}_b) = b : \begin{array}{c} (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathsf{Dec}(\mathsf{sk}, \cdot)}(\mathsf{pk}) \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b), b \leftarrow \{0, 1\} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

$\mathcal{A}$ is admissible if it only make queries to the oracle on $\mathsf{ct} \neq \mathsf{ct}_b$.

γ-Unremovability of CCA2-secure PKE The γ-Unremovability for a watermarkable CCA2 secure PKE scheme says, for all PPT admissible stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C) \notin \mathcal{Q} \land C \text{ is } \gamma\text{-good} : \begin{array}{c} (\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ C \leftarrow \mathcal{A}^{\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)}(1^\lambda, \mathsf{pk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

where $\mathcal{Q}$ is the set of marks queried by $\mathcal{A}$ and $C = (C_1, C_2)$ is a PPT admissible, stateful γ-good adversary in the security game $G_{CCA}(\mathsf{sk}, \mathsf{pk}, \cdot)$, more specifically:

$$\Pr\left[C_2^{\mathsf{Dec}(\mathsf{sk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c} (m_0, m_1) \leftarrow C_1^{\mathsf{Dec}(\mathsf{sk},\cdot)}(\mathsf{pk}, 1^\lambda) \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b), b \leftarrow \{0,1\} \end{array}\right] \geq \frac{1}{2} + \gamma.$$

C is admissible if it only make queries to the oracle on $\mathsf{ct} \neq \mathsf{ct}_b$.

6.1.1 Preliminaries

Definition: Watermarkable Implementation of Identity-Based Encryption For clarity, we first give the definition for watermarkable implementation of a selectively secure identity-based encryption wIBE where one can mark the master secret key, though it is easy to see that they match the definition in Section 4.1. It consists of the following algorithms.

$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{msk}, \mathsf{mpk}, \mathsf{mk}, \mathsf{xk})$: on input security parameter, outputs a master secret key msk, master public key mpk, marking key mk, extraction key xk.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to (\mathsf{sk}_{\mathsf{id}})$: on input mster secret key, and $\mathsf{id} \in \mathcal{ID}$, output a key $\mathsf{sk}_{\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{mpk}, m, \mathsf{id}) \to \mathsf{ct}_{\mathsf{id}}$: on input public key pk, message m and $\mathsf{id} \in \mathcal{ID}$, outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) \to m$: on input secret key $\mathsf{sk}_{\mathsf{id}}$ and ciphertext ct outputs a message m.

$\mathsf{Mark}(\mathsf{mk}, \mathsf{msk}, \tau) \to \mathsf{msk}_\tau$: on marking key mk, secret key msk, a message τ, output a marked key msk_τ.

$\mathsf{Extract}(\mathsf{xk}, \mathsf{mpk}, \mathsf{aux}, C) \to \tau/\perp$: on extraction key sk, public key mpk auxiliary input aux and program C, output a mark $\tau \in \mathcal{M}_\tau$ or $\perp$.

Note that for our applications, we only consider marking a master secret key. But a scheme where one marks an ID-embedded key can also be applicable in certain compositions. Both types of watermarkable IBE can be consructed from modifications of the watermarkable ABE scheme in [GKM+19].

Correctness The functionality-preserving property says, for all $m \in \mathcal{M}, \mathsf{id} \in ID$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\Pr\left[\begin{array}{c} (\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) = m) \wedge \\ (\mathsf{Dec}(\mathsf{msk}, \mathsf{ct}) = m) \end{array} : \begin{array}{c} (\mathsf{msk}, \mathsf{mpk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ \mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Functionality-Preserving The functionality-preserving property says, for all $m \in \mathcal{M}, \mathsf{id} \in ID$ and all $\tau \in \mathcal{M}_\tau$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\Pr\left[\begin{array}{c} (\mathsf{Dec}(\mathsf{sk}_{\mathsf{id},\tau}, \mathsf{ct}) = m) \wedge \\ (\mathsf{Dec}(\mathsf{msk}_\tau, \mathsf{ct}) = m) \end{array} : \begin{array}{c} (\mathsf{msk}, \mathsf{mpk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ \mathsf{msk}_\tau \leftarrow \mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \\ \mathsf{sk}_{\mathsf{id},\tau} \leftarrow \mathsf{KeyGen}(\mathsf{msk}_\tau, \mathsf{id}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Selective-ID Security The selective-ID CPA security property says, for all $m \in \mathcal{M}, \mathsf{id} \in ID$ and all $\tau \in \mathcal{M}_\tau$, and for all PPT admissible stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\Pr\left[\mathcal{A}_3^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c} \mathsf{id}^* \leftarrow \mathcal{A}_1(1^\lambda) \\ (\mathsf{msk}, \mathsf{mpk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ (m_0, m_1) \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}^*, m_b), b \leftarrow \{0,1\} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

where $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ is admissible if $\mathcal{A}_2, \mathcal{A}_3$ only makes queries to the oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ on $\mathsf{id} \neq \mathsf{id}^*$.

Additionally, we denote the above security game after $\mathsf{id}^*, msk, \mathsf{mpk}$ are fixed as $G_{\mathsf{IBE}}(\mathsf{msk}, \mathsf{mpk}, \mathsf{id}^*, \cdot)$.

Note that the stages in the above defintion do not exactly match the stage in the watermarking-compatible reduction, as shown below.

γ-Unremovability The γ-Unremovability for a watermarkable selective-ID secure IBE scheme says, for all $m \in \mathcal{M}, \mathsf{id} \in ID$ and all $\tau \in \mathcal{M}_\tau$, and for all PPT admissible stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\Pr\left[\begin{array}{c} \mathsf{Extract}(\mathsf{xk}, \mathsf{mpk}, \mathsf{aux} = \mathsf{id}^*, C) \notin \mathcal{Q} \\ \wedge C \text{ is } \gamma\text{-good} \end{array} : \begin{array}{c} \mathsf{id}^* \leftarrow \mathcal{A}_1(1^\lambda) \\ (\mathsf{msk}, \mathsf{mpk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ C \leftarrow \mathcal{A}_2^{\mathsf{Mark}(\mathsf{mk},\mathsf{msk},\cdot)}(\mathsf{mpk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

where $\mathcal{Q}$ is the set of marks queried by $\mathcal{A}_2$ and $C = (C_1, C_2)$ is a PPT admissible, stateful γ-good adversary in the security game $G_{\mathsf{IBE}}(\mathsf{msk}, \mathsf{mpk}, \mathsf{id}^*, \cdot)$, more specifically:

$$\Pr\left[C_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c} (m_0, m_1) \leftarrow C_1^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk}) \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}^*, m_b), b \leftarrow \{0,1\} \end{array}\right] \geq \frac{1}{2} + \gamma.$$

C is admissible if and only if C only make queries to the oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ on $\mathsf{id} \neq \mathsf{id}^*$.

6.2 Watermarkable CCA-secure PKE Construction

In this section, we prove Theorem A.1with an alternative approach, based on [BCHK07].

Theorem 6.1. *Assuming the hardness of LWE, there exists secure watermarkable implementation of identity-based encryption with selective-ID security, with private extraction and collusion-resistant security.*

We prove this above theorem in Appendix C by making a modification of [GKM+19].

Lemma 6.2. *Assuming there exists secure watermarkable implementation of (private-extraction, collusion resistant) identity-based encryption with selective-ID security and strongly unforgeable one-time signatures, then there exists secure (private-extraction, collusion resistant) watermarkable implementation of CCA-secure PKE*

Now we describe the construction for watermarkable implementation of CCA-secure PKE from a watermarkable implementation of selective-ID secure IBE scheme $\mathsf{wIBE} = (\mathsf{WMSetup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Mark}, \mathsf{Extract})$ and a one-time strongly unforgeable signature scheme $\mathsf{OTS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ (without watermarking).

$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{msk}, \mathsf{mpk}, \mathsf{mk}, \mathsf{xk})$: compute $(\mathsf{wIBE.msk}, \mathsf{wIBE.mpk}, \mathsf{wIBE.mk}, \mathsf{wIBE.xk}) \leftarrow \mathsf{wIBE.WMSetup}(1^\lambda)$; output $\mathsf{sk} = \mathsf{wIBE.msk}$; $\mathsf{pk} = \mathsf{wIBE.mpk}$; $\mathsf{mk} = \mathsf{wIBE.mk}$; $\mathsf{xk} = (\mathsf{wIBE.xk}, \mathsf{wIBE.mpk})$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$:

1. compute $(\mathsf{OTS.vk}, \mathsf{OTS.sk}) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$;
2. compute $\mathsf{ct}_{\mathsf{vk}} \leftarrow \mathsf{wIBE.Enc}(\mathsf{pk}, \mathsf{id} = \mathsf{vk}, m)$;
3. compute $\sigma \leftarrow \mathsf{OTS.Sign}(\mathsf{OTS.sk}, \mathsf{ct}_{\mathsf{vk}})$;
4. output $\mathsf{ct} = (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig}, \mathsf{vk})$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$:

1. parse $\mathsf{ct} := (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig}, \mathsf{vk}), \mathsf{sk} = \mathsf{wIBE.msk}$;
2. if $\mathsf{OTS.Verify}(\mathsf{vk}, (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig})) = 1$, continue; else abort and output $\perp$.
3. compute $\mathsf{sk}_{\mathsf{vk}} \leftarrow \mathsf{wIBE.KeyGen}(\mathsf{wIBE.msk}, \mathsf{id} = \mathsf{vk})$;
4. Output $m \leftarrow \mathsf{wIBE.Dec}(\mathsf{sk}_{\mathsf{vk}}, \mathsf{ct}_{\mathsf{vk}})$.

$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \to \mathsf{sk}_\tau$: compute $\mathsf{sk}_\tau \leftarrow \mathsf{wIBE.Mark}(\mathsf{mk}, \mathsf{sk} = \mathsf{wIBE.msk}, \tau)$.

$\mathsf{Extract}(\mathsf{xk}, \mathsf{aux} = \perp, C) \to \tau/\perp$:

1. parse $\mathsf{xk} := \mathsf{wIBE.xk}, \mathsf{wIBE.mpk}$. Samples $(\mathsf{OTS.vk}^*, \mathsf{OTS.sk}^*) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$.
2. Create a circuit C_{IBE} that has black-box access to C.
   - C_{IBE} is hardcoded with challenge $\mathsf{id}^* = \mathsf{OTS.vk}^*$ and $\mathsf{OTS.sk}^*$, $\mathsf{wIBE.mpk}$
   - C_{IBE} simulates the CCA PKE security game for C by simulating $\mathsf{Dec}(\mathsf{sk}, \cdot)$ oracles as follows:
     (a) when C makes a query $\mathsf{ct} = (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig}, \mathsf{vk})$; first check if $\mathsf{OTS.Verify}(\mathsf{vk}, (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig})) = 1$, if yes continue; else output $\perp$.
     (b) make an external query to a $\mathsf{wIBE.KeyGen}(\mathsf{msk}, \cdot)$ oracle for $\mathsf{sk}_{\mathsf{vk}} \leftarrow \mathsf{wIBE.KeyGen}(\mathsf{msk}, \mathsf{vk})$. Output $m \leftarrow \mathsf{wIBE.Dec}(\mathsf{sk}_{\mathsf{vk}}, \mathsf{ct}_{\mathsf{vk}})$.
   - In the challenge phase, C outputs (m_0, m_1); then C_{IBE} submits (m_0, m_1) to the external challenger and receives challenger ciphertext $\mathsf{ct}_{b,\mathsf{vk}^*} = \mathsf{Enc}(\mathsf{mpk}, \mathsf{OTS.vk}^*, m_b), b \leftarrow \{0, 1\}$;
   - feed C with $\mathsf{ct}_b^* = (\mathsf{ct}_{b,\mathsf{vk}^*}, \mathsf{sig}^* = \mathsf{OTS.Sign}(\mathsf{OTS.sk}^*, \mathsf{ct}_{b,\mathsf{vk}^*}), \mathsf{OTS.vk}^*)$.
   - Continue to simulate $\mathsf{Dec}(\mathsf{sk}, \cdot)$ oracle for C and only answer queries $\mathsf{ct} \neq \mathsf{ct}_b^*$; finally C outputs a bit b' and C_{IBE} outputs the same.
3. Output $\tau/\perp \leftarrow \mathsf{wIBE.Extract}(\mathsf{wIBE.xk}, \mathsf{aux} = \mathsf{id}^*, C_{\mathsf{IBE}})$;

Proof. We devide our analysis into cases.

Suppose there exists adversary $\mathcal{A}$ that breaks the γ-unremovability of watermarkable CCA-secure PKE for some non-negligible γ, i.e. $\mathcal{A}^{\mathsf{Dec}(\mathsf{sk},\cdot),\mathsf{Mark}(\mathsf{mk},\mathsf{sk},\cdot)}(\mathsf{pk})$ produces some program C such that $\Pr[G_{CCA}(\mathsf{sk}, \mathsf{pk}, C) = 1] \geq \frac{1}{2} + \gamma$ (for G_{CCA} see Section 5.5.2) and $\Pr[\mathsf{Extract}(\mathsf{xk}, C) \in \mathcal{Q}] \geq \epsilon$ for some non-negligible ϵ.

For any $(\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk})$ generated by $\mathsf{WMSetup}$ and any $\mathcal{A}^{\mathsf{Mark}(\mathsf{mk},\mathsf{sk},\cdot)}(\mathsf{pk})$ producing a γ-good C such that $\Pr[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C) \notin \mathcal{Q}] \geq \epsilon$, one of the following cases must hold:

- **Case 1: during the execution of** $\mathsf{Extract}(\mathsf{xk}, C)$**, the program** C **produced by the adversary** $\mathcal{A}$ **will only make queries that have the format** $\mathsf{ct} = (\mathsf{ct}_{\mathsf{vk}}, \mathsf{vk}, \mathsf{sig})$**, where** $\mathsf{vk} \neq \mathsf{vk}^*$**,** $\mathsf{vk}*$ **is the** OTS**'s verification key part in the challenge ciphertext** $\mathsf{ct}_b^* = (\mathsf{ct}_{b,\mathsf{vk}^*}, \mathsf{sig}^*, \mathsf{OTS.vk}^*)$.
  In this case, we show that we can use $\mathcal{A}$ to break the γ-unremovability of the watermarkable IBE.
  The reduction $\mathcal{B}_{\mathsf{wIBE}}$ interacts with $\mathcal{A}$ as follows:

- $\mathcal{B}$ samples $(\mathsf{vk}^*, \mathsf{sk}^*) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$ and submits vk^* as the challenge id^* to the challenger;
- $\mathcal{B}_{\mathsf{wIBE}}$ receives $\mathsf{wIBE.mpk}$ from the challenger and gives it as the public key to $\mathcal{A}$. For $\mathcal{A}$'s marking queries, $\mathcal{B}$ queries the marking oracle $\mathsf{wIBE.Mark}(\mathsf{wIBE.mk}, \mathsf{wIBE.msk}, \cdot)$;

Next, $\mathcal{A}$ outputs a program C. $\mathcal{B}$ creates a circuit C_{wIBE} with black-box access to C. C is treated as a stage-2 adversary in the reduction from CCA-secure PKE to selective-ID secure IBE:

- C_{wIBE} is hardcoded with the challenge $\mathsf{id}^* = \mathsf{vk}^*$ and its corresponding signing key sk^* and $\mathsf{wIBE.mpk}$. Note that C_{wIBE} does not receive any queries $\mathcal{B}$ has answered/queried when answering $\mathcal{A}$s queries in the previous stage.
- C_{wIBE} simulates answers to Cs decryption queries: when C makes a query $\mathsf{ct} = (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig}, \mathsf{vk})$; first check if $\mathsf{OTS.Verify}(\mathsf{vk}, (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig})) = 1$, if yes continue; else output $\perp$. Make an external query to the $\mathsf{wIBE.KeyGen}(\mathsf{msk}, \cdot)$ oracle for $\mathsf{sk}_{\mathsf{vk}} \leftarrow \mathsf{wIBE.KeyGen}(\mathsf{msk}, \mathsf{vk})$. Output $m \leftarrow \mathsf{wIBE.Dec}(\mathsf{sk}_{\mathsf{vk}}, \mathsf{ct}_{\mathsf{vk}})$. Since all the vk queried are not equal to vk^*, these queries arre all valid.
- In the challenge phase, C outputs (m_0, m_1); then C_{wIBE} submits (m_0, m_1) to the external challenger and receives challenger ciphertext $\mathsf{ct}_{b,\mathsf{vk}^*} = \mathsf{Enc}(\mathsf{mpk}, \mathsf{vk}^*, m_b), b \leftarrow \{0,1\}$;
- feed C with $\mathsf{ct}_b^* = (\mathsf{ct}_{b,\mathsf{vk}^*}, \mathsf{sig}^* = \mathsf{OTS.Sign}(\mathsf{sk}^*, \mathsf{ct}_{b,\mathsf{vk}^*}), \mathsf{vk}^*)$.
- Continue to simulate $\mathsf{Dec}(\mathsf{sk}, \cdot)$ oracle for C and only answer queries $\mathsf{ct} \neq \mathsf{ct}_b^*$; finally C outputs a bit b' and C_{wIBE} outputs the same.

In the execution of the extraction $\mathsf{Extract}(\mathsf{xk}, C)$, C is used in the program C_{IBE} as a black box and by our design, it must hold that $\Pr[\mathsf{wIBE.Extract}(\mathsf{xk}, \mathsf{mpk}, C_{\mathsf{IBE}}) \in \mathcal{Q} \wedge C \text{ is } \gamma\text{-good}] \geq \epsilon$ for some non-negligible ϵ. Note that C_{IBE} created by $\mathsf{Extract}(\mathsf{xk}, C)$ differs from C_{wIBE} created by $\mathcal{B}$ in that: $\mathsf{Extract}(\mathsf{xk}, C)$ samples a fresh $(\mathsf{vk}^*, \mathsf{sk}^*) \leftarrow \mathsf{OTS.KeyGen}(1^\lambda)$; C_{wIBE} is hardcoded with a $(\mathsf{vk}^*, \mathsf{sk}^*)$ sampled by $\mathcal{B}$ previously. However, the challenge id's in both settings are sampled freshly at random, independent of the $\mathsf{wIBE.KeyGen}(\mathsf{wIBE.msk}, \cdot)$(i.e. $\mathsf{Dec}(\mathsf{sk}, \cdot)$) queries made in the stage of $\mathcal{B}$ interacting with $\mathcal{A}$ before $\mathcal{A}$ outputs C (i.e. stage-1 in the watermarking-compatible reduction). Therefore, the output of $C_{\mathsf{wIBE}}, C_{\mathsf{IBE}}$ are the same unless in the real CCA PKE security game, $\mathcal{A}$ has made a query on some ciphertext $\mathsf{ct} = (\mathsf{ct}_{\mathsf{vk}^*} = \mathsf{wIBE.Enc}(\mathsf{mpk}, \mathsf{vk}^*, m_b), \mathsf{vk}^*, \mathsf{sig})$ for the same vk^* that is later sampled by $\mathsf{Extract}(\mathsf{xk}, C)$ and for one of the challenge messages $m_b \in \{m_0.m_1\}$. But this happens with negligible probability since vk^* is sampled at random.
Since we have $\Pr[\mathsf{wIBE.Extract}(\mathsf{xk}, \mathsf{aux} = \mathsf{vk}^*, C_{\mathsf{IBE}}) \in \mathcal{Q}] \geq \epsilon$ and given our assumption in case 1, we also have $\Pr[G_{\mathsf{IBE}}(\mathsf{wIBE.mpk}, \mathsf{wIBE.msk}, \mathsf{id}^* = \mathsf{vk}^*, C_{\mathsf{IBE}}) = 1] \geq 1/2 + \gamma$, we should deduce that the program C_{wIBE} produced by the reduction $\mathcal{B}_{\mathsf{wIBE}}$ from C performs the same with all but negligible difference: with probability $\epsilon - \mathsf{negl}(\lambda)$, we still have $\mathsf{wIBE.Extract}(\mathsf{xk}, \mathsf{aux} = \mathsf{id}^* C_{\mathsf{wIBE}}) \in \mathcal{Q}]$ and $\Pr[G_{\mathsf{IBE}}(\mathsf{wIBE.mpk}, \mathsf{wIBE.msk}, \mathsf{aux} = \mathsf{id}^*, C_{\mathsf{wIBE}}) = 1] \geq 1/2 + \gamma$.

- **Case 2: during the execution of** $\mathsf{Extract}(\mathsf{xk}, C)$**, the program** C **produced by the adversary** $\mathcal{A}$ **will make at least one query that have the format** $\mathsf{ct} = (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig}, \mathsf{vk})$**, where** $\mathsf{vk} = \mathsf{OTS.vk}^*$**,** $\mathsf{OTS.vk}^*$ **is the** OTS**'s verification key part in the challenge ciphertext** ct^***; further more, the query** $\mathsf{ct} = (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig}, \mathsf{vk})$ **satisfies** $\mathsf{OTS.Verify}(\mathsf{OTS.vk}^*, \mathsf{ct}_{\mathsf{vk}}, \mathsf{sig}) = 1$ **and** $\mathsf{ct}^* \neq \mathsf{ct}$
In this case, we show a Type-2 watermarking compatible reduction to the strongly unforgeability security of the one-time signature scheme.

In stage-1 of the watermarking-compatible reduction, $\mathcal{B}^1_{\mathsf{OTS}}$ operates as follows:

- $\mathcal{B}^1_{\mathsf{OTS}}$ receives vk^* from the challenger. It samples $\mathsf{wIBE.xk}, \mathsf{msk}, \mathsf{mpk}, \mathsf{mk}$ on its own. $\mathcal{B}^1_{\mathsf{OTS}}$ can thus answer all the marking queries from $\mathcal{A}$.

After $\mathcal{A}$ outputs program C, since we are given that $\Pr[G_{CCA}(\mathsf{sk}, \mathsf{pk}, C) = 1] \geq 1/2 + \gamma$ and C makes decryption queries on valid ciphertexts with the vk^*. $\mathcal{B}_{\mathsf{OTS}}$ can do the following.
Entering stage-2 of the reduction, $\mathcal{B}^2_{\mathsf{OTS}}$ receives all stateful information from $\mathcal{B}^1_{\mathsf{OTS}}$ except for the queries made by $\mathcal{A}$.
$\mathcal{B}^2_{\mathsf{OTS}}$ runs C and simulates the CCA-security game, $\mathsf{Dec}(\mathsf{sk}, \cdot)$ oracles for C as follows:

- For decryption queries o the format $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{vk}, \mathsf{sig})$, first checks if $\mathsf{OTS.Verify}(\mathsf{vk}, (\mathsf{ct}_{\mathsf{vk}}, \mathsf{sig})) = 1$, if yes continue; else output $\perp$.
  Then decrypt ct directly using $\mathsf{sk}_{\mathsf{vk}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{vk})$ and $m \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{vk}}, \mathsf{ct}_{\mathsf{vk}})$.

In the challenge phase, C submits 2 messages (m_0, m_1) and $\mathcal{B}^2_{\mathsf{OTS}}$ outputs challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}_{\mathsf{vk}^*,b} = \mathsf{wIBE.Enc}(\mathsf{mpk}, \mathsf{vk}^*, m_b \leftarrow (m_0, m_1)), \mathsf{vk}^*, \mathsf{sig}^*)$ by making one query to the signing oracle $\mathsf{OTS.Sign}(\mathsf{sk}^*, \cdot)$ on message $\mathsf{ct}_{\mathsf{vk}^*,b}$.
Then the query phase continues and C is only allowed to query on $\mathsf{ct} \neq \mathsf{ct}^*$. Since in the end, there will be one query ct such that $\mathsf{ct} = (\mathsf{ct}', \mathsf{vk}^*, \mathsf{sig}')$ where $\mathsf{ct} \neq \mathsf{ct}^*$ but $\mathsf{OTS.Verify}(\mathsf{vk}^*, \mathsf{ct}', \mathsf{sig}') = 1$. Thus $\mathcal{B}^2_{\mathsf{OTS}}$ can output $(\mathsf{ct}', \mathsf{sig}')$ as a forgery.

□

7 References

[BCHK07] Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *SIAM Journal on Computing*, 36(5):1301–1328, 2007.

[CFNP00] B. Chor, A. Fiat, M. Naor, and B. Pinkas. Tracing traitors. *IEEE Transactions on Information Theory*, 46(3):893–910, 2000.

[CHN+16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 1115–1127, New York, NY, USA, 2016. Association for Computing Machinery.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 542–552, 1991.

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

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

[GKM+19] Rishab Goyal, Sam Kim, Nathan Manohar, Brent Waters, and David J Wu. Watermarking public-key cryptographic primitives. In *Advances in Cryptology–CRYPTO*

2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39, pages 367–398. Springer, 2019.

[GKP$^+$13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 555–564, 2013.

[GKWW21] Rishab Goyal, Sam Kim, Brent Waters, and David J Wu. Beyond software watermarking: traitor-tracing for pseudorandom functions. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27*, pages 250–280. Springer, 2021.

[KN22] Fuyuki Kitagawa and Ryo Nishimaki. Watermarking prfs against quantum adversaries. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 488–518, Cham, 2022. Springer International Publishing.

[KW17] Sam Kim and David J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 503–536, Cham, 2017. Springer International Publishing.

[MW22] Sarasij Maitra and David J. Wu. Traceable prfs: Full collusion resistance and active security. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography – PKC 2022*, pages 439–469, Cham, 2022. Springer International Publishing.

[Nis13] Ryo Nishimaki. How to watermark cryptographic functions. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 111–125, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Nis20] Ryo Nishimaki. Equipping public-key cryptographic primitives with watermarking (or: A hole is to watermark). In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography*, pages 179–209, Cham, 2020. Springer International Publishing.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, 1990.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for np from (plain) learning with errors. In *Annual International Cryptology Conference*, pages 89–114. Springer, 2019.

[Wat24] Brent Waters. A new approach for non-interactive zero-knowledge from learning with errors. Cryptology ePrint Archive, Paper 2024/340, 2024. `https://eprint.iacr.org/2024/340`.

A Watermarkable CCA-secure PKE from Watermarkable CPA-secure PKE and Statistically Simulation-Sound NIZK

In this section we present a watermarkable CCA2-secure PKE, based on a modification of the [NY90] scheme.

A.1 Preliminaries: Statistically Simulation Sound NIZK Proof for NP

A statistically simulation sound NIZK proof for language $L \in \mathsf{NP}$ with relation R_L consists of the following efficient algorithms.

- $\mathsf{NIZK.Setup}(1^\lambda) \to (\mathsf{CRS}, \mathsf{td})$: On input the security parameter 1^λ, the setup returns a common reference string CRS and a trapdoor td.
- $\mathsf{NIZK.Prove}(\mathsf{CRS}, w, x) \to \pi$: On input a common reference string CRS, a classical witness w, and a statement x, the proving algorithm returns a proof π.
- $\mathsf{NIZK.Ver}(\mathsf{CRS}, \pi, x) \to 0/1$: On input a common reference string CRS, a proof π, and a statement x, the verification algorithm returns a bit in $\{0, 1\}$.

A SSS-NIZK for NP scheme should satisfy the following properties:

Correctness A NIZK proof ($\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Ver}$) is correct if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all $x \in L$, and all $w \in \mathcal{R}_L(x)$ it holds that

$$\Pr[\mathsf{NIZK.Ver}(\mathsf{CRS}, \mathsf{NIZK.Prove}(\mathsf{CRS}, w, x), x) = 1] = 1 - \mathsf{negl}(\lambda)$$

where $\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$.

Statistical Soundness A NIZK proof ($\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Ver}$) is computationally sound if there exist a negligible function $\mathsf{negl}(\cdot)$ such that for all unbounded adversaries $\mathcal{A}$ and all $x \notin L$, it holds that:

$$\Pr[\mathsf{NIZK.Ver}(\mathsf{CRS}, \pi \leftarrow \mathcal{A}(\mathsf{CRS}, x), x) = 1] = \mathsf{negl}(\lambda)$$

where $\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$.

Computational Zero Knowledge A NIZK proof ($\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Ver}$) is computationally zero-knowledge if there exists a PPT simulator Sim such that for all non-uniform PPT adversaries, all statements $x \in L$ and all witnesses $w \in \mathcal{R}_L(x)$, it holds that $\mathsf{Sim}(1^\lambda, \mathsf{CRS}, \mathsf{td}, , x) \approx_c \mathsf{NIZK.Prove}(\mathsf{CRS}, w, x)$ where $\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$.

Statistically Simulation Soundness A NIZK proof satisfies statistically simulation soundness if it is infeasible to convince an honest verifier of a false statement even when the adversary itself is provided with a simulated proof.

Formally, for all statements x and all (even unbounded) adversaries $\mathcal{A} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$, there exists a negligible function $\mathsf{negl}(\cdot)$:

$$\Pr\left[(\mathsf{CRS}, \mathsf{td}) \leftarrow \mathsf{Sim}_1(1^\lambda, x), \pi \leftarrow \mathsf{Sim}_2(x, \mathsf{CRS}, \mathsf{td}) : \substack{\exists (x', \pi') \wedge x' \neq x \wedge x' \in L_{no} \\ \text{and} \\ V(\mathsf{CRS}, x', \pi') = 1}\right] \leq \mathsf{negl}(\cdot)$$

where td is a trapdoor generated by the simulator along with the simulated CRS.

Instantiation A SSS-NIZK proof system can be realized from a standard NIZK proof system [GGH+16], which can be built on the hardness of LWE [PS19, Wat24].

A.2 Construction and Security

Theorem A.1. *Assuming the hardness of LWE, there exists secure watermarkable implementation of CCA-secure PKE with private extraction and collusion resistance.*

Lemma A.2. *Assuming the security of watermarkable implementation of a CPA-secure PKE* $\mathsf{wPKE} = (\mathsf{WMSetup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Mark}, \mathsf{Extract})$ *and statistically simulation sound NIZK scheme* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *(without watermarking) , there exists a secure watermarkable implementation of CCA(2)-secure PKE.*

The two building blocks: watermarkable implementation of a CPA-secure PKE $\mathsf{wPKE} = (\mathsf{WMSetup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Mark}, \mathsf{Extract})$ and a SSS-NIZK scheme $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ can both be obtained from LWE.

Construction Given a watermarkable implementation of a CPA-secure PKE $\mathsf{wPKE} = (\mathsf{WMSetup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Mark}, \mathsf{Extract})$ and a SSS-NIZK scheme $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, the construction of watermarkable implementation of CCA-secure PKE is as follows.

$\mathsf{KeyGen}(1^\lambda, 1^n)$: Compute $(\mathsf{wPKE.pk}_1, \mathsf{wPKE.sk}_1, \mathsf{wPKE.xk}_1, \mathsf{wPKE.mk}_1) \leftarrow \mathsf{wPKE.WMSetup}(\lambda)$; $(\mathsf{wPKE.pk}_2, \mathsf{wPKE.sk}_2, \mathsf{wPKE.xk}_2, \mathsf{wPKE.mk}_2) \leftarrow \mathsf{wPKE.WMSetup}(\lambda)$;
Compute $(\mathsf{CRS}, \mathsf{td}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$; Output $\mathsf{sk} = (\mathsf{wPKE.sk}_i)_{i\{1,2\}}$; $\mathsf{pk} = (\{\mathsf{wPKE.pk}_i\}_{i\in\{1,2\}}, \mathsf{CRS})$; $\mathsf{xk} = (\{\mathsf{wPKE.xk}_i\}_{i\in\{1,2\}}, \mathsf{td})$; $\mathsf{mk} = \{\mathsf{wPKE.mk}_i\}_{i\in\{1,2\}}$

$\mathsf{Enc}(\mathsf{pk}, m)$:

- parse $\mathsf{pk} := \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{CRS}$;
- compute $\mathsf{ct}_1 \leftarrow \mathsf{wPKE.Enc}(\mathsf{pk}_1, m)$; $\mathsf{ct}_2 \leftarrow \mathsf{wPKE.Enc}(\mathsf{pk}_2, m)$;
- compute $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, (\mathsf{ct}_1, \mathsf{ct}_2), (r_1, r_2, m))$ for the following statement: $\exists$ witness (r_1, r_2, m) such that $\mathsf{ct}_i = \mathsf{wPKE.Enc}(\mathsf{pk}_i, m; r_i)$ for all $i \in [2]$, where r_i is the randomness used in encryption.
- Output $\mathsf{ct} = (\{\mathsf{ct}_i\}_{i\in[2]}, \pi)$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$:

- parse $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \pi)$; $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$;
- if $\mathsf{NIZK.Verify}(\mathsf{CRS}, \pi, (\mathsf{ct}_1, \mathsf{ct}_2)) = 1$, continue; else abort and output $\bot$.
- compute $m \leftarrow \mathsf{wPKE.Dec}(\mathsf{sk}_1, \mathsf{ct}_1)$; output m'.

$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau)$: parse $\mathsf{mk} = (\mathsf{mk}_1, \mathsf{mk}_2)$; $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$; output $(\mathsf{sk}_{\tau,1} \leftarrow \mathsf{wPKE.Mark}(\mathsf{sk}_1, \tau)$; $\mathsf{sk}_{\tau,2} \leftarrow \mathsf{wPKE.Mark}(\mathsf{sk}_2, \tau)$;

$\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux}, C)$:

- parse $\mathsf{xk} := (\mathsf{xk}_1, \mathsf{xk}_2, \mathsf{NIZK.td})$; $\mathsf{pk} := (\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{CRS})$. Initialize an empty tuple $\vec{\tau}$;
  For $i = 1, 2$: Create the following circuit C_i with black-box access to C:
  - C_i is hardcoded with $(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{xk}_{j\neq i}, \mathsf{td}, \mathsf{CRS})$ and simulates the $\mathsf{CCA} - \mathsf{PKE}$ game for C as follows:
  - For C's decryption queries $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \pi)$:
    - First check if $\mathsf{NIZK.Verify}(\mathsf{CRS}, \pi) = 1$, if 0 output $\bot$; else continue;

    * By the extraction key simulation property, since $\mathsf{xk}_{j \neq i}$ can be used to simulate the oracles used in $G_{CPA}(\mathsf{pk}_j, \mathsf{sk}_j, \cdot)$, C_i can simulate the oracle $\mathsf{wPKE.Dec}(\mathsf{sk}_j, \cdot)$ and thus decrypt ct_j in the ciphertext ct to output m.
  - C submits challenge messages (m_0, m_1); C_i submits (m_0, m_1) to the external challenger; C_i receives challenge ciphertext $\mathsf{ct}_i^* = \mathsf{wPKE.Enc}(\mathsf{pk}_i, m_b), b \leftarrow \{0,1\}$ from the challenger.
    C_i prepares the following ciphertext: compute $\mathsf{ct}_j \leftarrow \mathsf{wPKE.Enc}(\mathsf{pk}_j, m_{b_j}), b_j \leftarrow \{0,1\}$;
    compute $\widehat{\pi} \leftarrow \mathsf{Sim}(\mathsf{td}, \mathsf{CRS}, (\mathsf{ct}_1^*, \mathsf{ct}_2^*))$ where Sim is the simulator algorithm for NIZK.
    Then it sends $\mathsf{ct}^* = (\mathsf{ct}_1^*, \mathsf{ct}_2^*, \widehat{\pi})$ to C.
  - C continues to simulate the decryption oracle as above to decrypt only valid ciphertexts $\mathsf{ct} \neq \mathsf{ct}^*$.
  - In the end, C_i outputs the same as C outputs.
  - Add $\tau/\bot \leftarrow \mathsf{wPKE.Extract}(\mathsf{wPKE.xk}_i, \mathsf{wPKE.pk}_i, \mathsf{aux} = \bot, C_i)$ to $\vec{\tau}$.
- Output $\vec{\tau}$.

Proof. We make a few claims for the proof of unremovability.

First, we consider a hybrid game where stage-1 (before the unremovability adversary $\mathcal{A}$ produces program C) is the same, but the stage-2 game for $\mathsf{CCA} - \mathsf{PKE}$ where the program C plays in is different. We call this stage-2 game $G_{\mathsf{CCA},H_1}(\mathsf{sk}, \mathsf{pk}, \cdot)$, defined as follows:

- On input $(\mathsf{sk}, \mathsf{pk}) = (\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{sk}_1, \mathsf{sk}_2, \mathsf{td}, \mathsf{CRS})$; the challenger plays the $\mathsf{CCA} - \mathsf{PKE}$ game for C as follows:
- For C's decryption queries $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \pi)$:
  - First check if $\mathsf{NIZK.Verify}(\mathsf{CRS}, \pi) = 1$, if 0 output $\bot$; else continue; output $m' \leftarrow \mathsf{wPKE.Dec}(\mathsf{sk}_1, \mathsf{ct}_1)$.
- C submits challenge messages (m_0, m_1)
  The challenger prepares the following ciphertext: compute $\mathsf{ct}_i^* \leftarrow \mathsf{wPKE.Enc}(\mathsf{pk}_i, m_b)$ for all $i = 1, 2$, $b \leftarrow \{0,1\}$;
  $\underline{\text{compute } \widehat{\pi} \leftarrow \mathsf{Sim}(\mathsf{td}, \mathsf{CRS}, (\mathsf{ct}_1^*, \mathsf{ct}_2^*)) \text{ where Sim is the simulator algorithm for NIZK.}}$
  Then it sends $\mathsf{ct}^* = (\mathsf{ct}_1^*, \mathsf{ct}_2^*, \widehat{\pi})$ to $\mathcal{A}$ and C outputs a guess b'.
- continues to simulate the decryption oracle as above to decrypt only valid ciphertexts $\mathsf{ct} \neq \mathsf{ct}^*$.

Claim A.3. *Assuming the computational zero-knowledge property of* NIZK, *for any admissible PPT C, if C is a γ-good program in the original game $G_{\mathsf{CCA-PKE}}(\mathsf{sk}, \mathsf{pk}, \cdot)$, then C is a $(\gamma - \mathsf{negl}(\lambda))$-good program in the above $G_{\mathsf{CCA},H_1}(\mathsf{sk}, \mathsf{pk}, \cdot)$ defined.*

Proof. We refer the definition of $G_{\mathsf{CCA-PKE}}(\mathsf{sk}, \mathsf{pk}, C)$ to Section 6.1.

Let $\mathsf{view}_{\mathsf{CCA},H_1}$(see Definition 3.8 for defintion) be the transcript output by the above game.

It is to see that the only difference in $G_{\mathsf{CCA},H_1}(\mathsf{sk}, \mathsf{pk}, C)$ and $G_{\mathsf{CCA-PKE}}(\mathsf{sk}, \mathsf{pk}, C)$ is in the generation of the proof π: $G_{\mathsf{CCA-PKE}}(\mathsf{sk}, \mathsf{pk}, C)$ the proof is generated honestly using NIZK.Prove and in $G_{\mathsf{CCA},H_1}(\mathsf{sk}, \mathsf{pk}, C)$ it is generated by $\mathsf{Sim}(\mathsf{td}, \cdot)$. By the computational zero knowledge property of NIZK, $\mathsf{view}_{G_{\mathsf{CCA-PKE}}}$ and $\mathsf{view}_{\mathsf{CCA},H_1}$ are computationally indistinguishable. Otherwise we can build a distinguisher, given NIZK.CRS, that samples WMSetup on its own and use C to break the computational zero knowledge property.

Therefore, the output distributions of any admissible PPT $\mathcal{A}$ in $G_{\mathsf{CCA},H_1}(\mathsf{sk},\mathsf{pk},C)$ and $G_{\mathsf{CCA-PKE}}(\mathsf{sk},\mathsf{pk},C)$ must be computationally indistinguishable. Thus, any γ-good C in $G_{\mathsf{CCA},H_1}(\mathsf{sk},\mathsf{pk},\cdot)$ must be $(\gamma-\mathsf{negl}(\lambda))$-good in $G_{\mathsf{CCA-PKE}}(\mathsf{sk},\mathsf{pk},\cdot)$

□

Next, we consider a next stage-2 hybrid game $G_{\mathsf{CCA},H_2}(\mathsf{sk},\mathsf{pk},\cdot)$ (note that the stage before $\mathcal{A}$ outputs C is still the same:

- On input $(\mathsf{sk},\mathsf{pk})=(\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,\mathsf{sk}_2,\mathsf{td},\mathsf{CRS})$;
- For C's decryption queries $\mathsf{ct}=(\mathsf{ct}_1,\mathsf{ct}_2,\pi)$:
  - First check if $\mathsf{NIZK.Verify}(\mathsf{CRS},\pi)=1$, if 0 output $\bot$; else continue; output $m'\leftarrow\mathsf{wPKE.Dec}(\mathsf{sk}_1,\mathsf{ct}_1)$.
- C submits challenge messages (m_0,m_1)
  The challenger prepares the following ciphertext: compute $\underline{\mathsf{ct}_1^*\leftarrow\mathsf{wPKE.Enc}(\mathsf{pk}_1,m_{b_1}),}$ $\underline{b_1\leftarrow\{0,1\}\text{; compute }\mathsf{ct}_2^*\leftarrow\mathsf{wPKE.Enc}(\mathsf{pk}_2,m_{b_2})\ b_2\leftarrow\{0,1\}}$
  compute $\widehat{\pi}\leftarrow\mathsf{Sim}(\mathsf{td},\mathsf{CRS},(\mathsf{ct}_1^*,\mathsf{ct}_2^*))$ where Sim is the simulator algorithm for NIZK.
  Then it sends $\mathsf{ct}^*=(\mathsf{ct}_1^*,\mathsf{ct}_2^*,\widehat{\pi})$ to $\mathcal{A}$ and C outputs a guess b'.
- continues to simulate the decryption oracle as above to decrypt only valid ciphertexts $\mathsf{ct}\neq\mathsf{ct}^*$.

Claim A.4. *For any admissible PPT C, if C is a γ-good program in the game $G_{\mathsf{CCA},H_1}(\mathsf{sk},\mathsf{pk},\cdot)$, then C is a $\gamma/2$-good program in the above $G_{\mathsf{CCA},H_2}(\mathsf{sk},\mathsf{pk},\cdot)$ defined.*

Proof. The only difference between $G_{\mathsf{CCA},H_1}(\mathsf{sk},\mathsf{pk},\cdot)$ and $G_{\mathsf{CCA},H_2}(\mathsf{sk},\mathsf{pk},\cdot)$ is that the bits b_i used to decide which of m_0,m_1 to encrypt for the challenge ciphertext is the same for $G_{\mathsf{CCA},H_1}(\mathsf{sk},\mathsf{pk},\cdot)$ and independent for $G_{\mathsf{CCA},H_2}(\mathsf{sk},\mathsf{pk},\cdot)$. When $b_1=b_2$, the two games are the same. Therefore, we have that C is a $\gamma/2$-good program in the above $G_{\mathsf{CCA},H_2}(\mathsf{sk},\mathsf{pk},\cdot)$ given that C is a γ-good program in the above $G_{\mathsf{CCA},H_1}(\mathsf{sk},\mathsf{pk},\cdot)$.

□

Next we show that for any γ-unremovability adversary $\mathcal{A}$ where the output program C is γ-good in $G_{\mathsf{CCA},H_2}(\mathsf{sk},\mathsf{pk},\cdot)$, $\mathcal{A}$ can be used to build a reduction to γ-unremovability of CPA-security of for wPKE.

Claim A.5. *For any γ-unremovability adversary $\mathcal{A}$ where the output program C is γ-good in $G_{\mathsf{CCA-PKE}}(\mathsf{sk},\mathsf{pk},\cdot)$, $\mathcal{A}$ can be used to build a reduction to $(\gamma/2-\mathsf{negl}(\lambda))$-unremovability of CPA-security of for* wPKE.

Proof. By Claim A.3 and Claim A.4, we know that if there is a γ-unremovability adversary $\mathcal{A}$ for Section 6.1, then if we put its output circuit C into our $\mathsf{Extract}(\mathsf{xk},\mathsf{pk},\mathsf{aux},C)$ procedure, the game simulated by Extract, when $i=2$ is exactly the same as $G_{\mathsf{CCA},H_2}(\mathsf{sk},\mathsf{pk},\cdot)$, i.e. the NIZK proof in the challenge ciphertext is generated by simulator and the bits for encryption are independent.

Thus, for any C produced by a winning $\mathcal{A}$, if C is γ-good by definition in $G_{\mathsf{CCA-PKE}}(\mathsf{sk},\mathsf{pk},\cdot)$, C will be a $(\gamma-\mathsf{negl}(\lambda))/2$-good program during the execution of $\mathsf{Extract}(\mathsf{xk},\mathsf{pk},C)$ for $i=2$.

Also, by the statistical soundness and statistical simulation soundness of NIZK, no adversary $\mathcal{A}$ and its output program C will make a query on a ct that contains a false proof π', that will pass the verification, even after C has seen the simulated proof $\widehat{\pi}$, except with negligible probability.

Therefore we can say that with overwhelming probability, all queries made in the unremovability game (including execution of Extract contain valid proofs.

Now we can create a reduction $\mathcal{B}_2$ to break the $\gamma/2 - \mathsf{negl}(\lambda)$-unremovability of CPA-secure PKE (of sk_2). $\mathcal{B}_2$ simulates the marking oracle for $\mathcal{A}$ by querying the marking oracle for $\mathsf{wPKE.sk}_1$. $\mathcal{B}_2$ also samples $\mathsf{wPKE.sk}_1$ on its own.

After $\mathcal{A}$ outputs C, $\mathcal{B}_2$ created program C_2' that works exactly as the circuit C_2 in our $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C)$ algorithm when $i = 2$, except that C_2' can hardcode the secret key sk_1 sampled by $\mathcal{B}_2$ to simulate the decryption oracle.

Also, since we have that $\Pr[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C) \notin \mathcal{Q}] \geq \epsilon$ for some non-negligible ϵ, we must have that $\Pr[\mathsf{wPKE.Extract}(\mathsf{xk}_2, \mathsf{pk}_2, C_2') \notin \mathcal{Q}] \geq \epsilon$ for some non-negligible ϵ by the design of our Extract algorithm.

□

We next analyze a symmetric case, consider the following game $G_{\mathsf{CCA},H_3}(\mathsf{sk}, \mathsf{pk}, \cdot)$.

First, before $\mathcal{A}$ outputs program C, we switch to answering $\mathcal{A}$'s decryption queries using $\mathsf{wPKE.Dec}(\mathsf{sk}_2, \cdot)$.

Note that the underlined differences are between *game* $G_{\mathsf{CCA},H_1}(\mathsf{sk}, \mathsf{pk}, \cdot)$ *and game* $G_{\mathsf{CCA},H_3}(\mathsf{sk}, \mathsf{pk}, \cdot)$.

- On input $(\mathsf{sk}, \mathsf{pk}) = (\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{sk}_1, \mathsf{sk}_2, \mathsf{td}, \mathsf{CRS})$;
- For C's decryption queries $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \pi)$:
  - First check if $\mathsf{NIZK.Verify}(\mathsf{CRS}, \pi) = 1$, if 0 output $\perp$; else continue; $\underline{\text{output } m' \leftarrow \mathsf{wPKE.Dec}(\mathsf{sk}_2, \mathsf{ct}_1)}$.
- C submits challenge messages (m_0, m_1)
  The challenger prepares the following ciphertext: compute $\underline{\mathsf{ct}_1^* \leftarrow \mathsf{wPKE.Enc}(\mathsf{pk}_1, m_{b_1})}$, $\underline{b_1 \leftarrow \{0,1\}; \text{compute } \mathsf{ct}_2^* \leftarrow \mathsf{wPKE.Enc}(\mathsf{pk}_2, m_{b_2})\ b_2 \leftarrow \{0,1\}}$
  $\underline{\text{compute } \widehat{\pi} \leftarrow \mathsf{Sim}(\mathsf{td}, \mathsf{CRS}, (\mathsf{ct}_1^*, \mathsf{ct}_2^*)) \text{ where Sim is the simul}}$ator algorithm for NIZK.
  Then it sends $\mathsf{ct}^* = (\mathsf{ct}_1^*, \mathsf{ct}_2^*, \widehat{\pi})$ to $\mathcal{A}$ and C outputs a guess b'.
- continues to simulate the decryption oracle as above to decrypt only valid ciphertexts $\mathsf{ct} \neq \mathsf{ct}^*$.

Note that the only difference is that we now answer decryption queries of C using sk_2. This results in no difference on the adversary's view because by the statistical soundness of NIZK, all ciphertexts submitted to the decryption oracle have their $\mathsf{ct}_1, \mathsf{ct}_2$ encrypt the same message or will result in $\perp$ replies.

By the exact (but symmetric) analysis as Claim A.5, we can do a reduction to break the $(\gamma/2 - \mathsf{negl}(\lambda))$-unremovability of wPKE where the reduction is challenged with CPA-security game under $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{mk}_1, \mathsf{xk}_1)$ and samples $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{mk}_2, \mathsf{xk}_2)$ on its own.

□

B Watermarkable Weak PRP from Watermarkable weak PRF

In this section, we present a watermarkable weak PRP built using the two-round Feistel network from a watermarkable weak PRF.

A weak PRP of input-output space $\{0,1\}^\ell$ consists of algorithms:

$\mathsf{KeyGen}(1^\lambda) \rightarrow \mathsf{sk}$: a randomized algorithm that generates a secret key sk.

$\mathsf{Eval}(\mathsf{sk}, x \in \{0,1\}^\ell) \to y \in \{0,1\}^\ell$: a deterministic evaluation algorithm that on input sk, x, outputs y.

The other syntax and definitions (such as correcntess, functionality preserving) are the same as watermarkable weak PRF and we omit them here.

γ-Unremovability of weak PRP The γ-Unremovability for a watermarkable implementation of a weak PRF scheme says, for all PPT admissible stateful adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathsf{Extract}(\mathsf{xk}, C) \notin \mathcal{Q} \wedge C \text{ is } \gamma\text{-good} : \begin{array}{c}(\mathsf{sk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ C \leftarrow \mathcal{A}^{\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)}(1^\lambda)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

$\mathcal{Q}$ is the set of marks queried by $\mathcal{A}$ and C is said to be a PPT admissible, stateful γ-good circuit if:

$$\Pr\left[C^{\mathsf{Eval}(\mathsf{sk}, \cdot)}(x, y_b) = b : \ x \leftarrow \{0,1\}^\ell, y_0 = \mathsf{Eval}(\mathsf{sk}, x), y_1 \leftarrow \{0,1\}^\ell, b \leftarrow \{0,1\}\right] \geq \frac{1}{2} + \gamma.$$

$\mathsf{Eval}(\cdot)$ samples a uniformly random $x \leftarrow \{0,1\}^\ell$ upon every query and outputs $(x, \mathsf{Eval}(\mathsf{sk}, x))$.

B.1 Construction and Security

Given a watermarkable implementation of a weak PRF $\mathsf{wPRF} = (\mathsf{WMSetup}, \mathsf{Eval}, \mathsf{Mark}, \mathsf{Extract})$ where the input and output space of Eval is $\{0,1\}^{\ell/2}$, we give the construction based on a 2-round Feistel network as below:

```
$\mathsf{WMSetup}(1^\lambda)$ : compute $(\mathsf{sk}_1, \mathsf{xk}_1, \mathsf{mk}_1) \leftarrow \mathsf{wPRF.WMSetup}(1^\lambda)$; $(\mathsf{sk}_2, \mathsf{xk}_2, \mathsf{mk}_2) \leftarrow \mathsf{wPRF.WMSetup}(1^\lambda)$;
    output $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$; $\mathsf{xk} = (\mathsf{xk}_1, \mathsf{xk}_2)$; $\mathsf{mk} = (\mathsf{mk}_1, \mathsf{mk}_2)$.
$\mathsf{Eval}(\mathsf{sk}, x \in \{0,1\}^\ell)$:
        1. parse $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$; parse $x = x_0 \| x_1$ where $x_0, x_1 \in \{0,1\}^{\ell/2}$.
        2. for $i = 1, 2$:
               $x_{i+1} = \mathsf{wPRF.Eval}(\mathsf{sk}_i, x_i) \oplus x_{i-1}$
        3. output $x_2 \| x_3$
$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau)$ : parse $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$; $\mathsf{mk} = (\mathsf{mk}_1, \mathsf{mk}_2)$;
    Output $(\tilde{\mathsf{sk}}_1 \leftarrow \mathsf{wPRF.Mark}(\mathsf{sk}_1, \tau), \tilde{\mathsf{sk}}_2 \leftarrow \mathsf{wPRF.Mark}(\mathsf{sk}_2, \tau))$.
$\mathsf{Extract}(\mathsf{xk}, C)$:
        1. parse $\mathsf{xk} = (\mathsf{xk}_1, \mathsf{xk}_2)$;
        2. For $i = 1$: create the following circuit $C_1$
            (a) $C_1$ is harcoded with $\mathsf{xk}_2$ has black-box access to $C$. $C_1$ answers $C'$s queries using
                $\mathsf{xk}_2$ and external queries:
                  i. Upon $C'$s query: query external oracle $\mathcal{O}(\mathsf{sk}_1, \cdot)$ which will return $(x_1 \leftarrow \{0,1\}^{\ell/2}$,
                     $\mathsf{wPRF.Eval}(\mathsf{sk}_1, x_1))$;
                 ii. sample $(x_2 \leftarrow \{0,1\}^{\ell/2}, \mathsf{wPRF.Eval}(\mathsf{sk}_2, x_2))$ using $\mathsf{xk}_2$. Let $x_0 := \mathsf{wPRF.Eval}(\mathsf{sk}_1, x_1) \oplus$
                     $x_2$ and $x_3 = \mathsf{wPRF.Eval}(\mathsf{sk}_2, x_2) \oplus x_1$.
                iii. output $(x_0 \| x_1, x_3 \| x_2)$.
```

    (b) In the challenge phase, receive the challenge (x_1^*, y_1^*) from the external challenger; compute challenge input for C: $(x_0^*\|x_1^*, x_3^*\|x_2^*)$ the same way as above queries.
    (c) C outputs a guess b' and C_1 outputs the same.
3. compute $\tau_1/\perp \leftarrow \mathsf{wPRF.Extract}(\mathsf{xk}_1, C_1)$;
4. For $i = 2$: create the following circuit C_2
    (a) C_2 is harcoded with xk_1 has black-box access to C. C_2 answers C's queries using xk_1 and external queries:
        i. Upon C's query: query external oracle $\mathcal{O}(\mathsf{sk}_2, \cdot)$ which will return $(x_2 \leftarrow \{0,1\}^{\ell/2}$, $\mathsf{wPRF.Eval}(\mathsf{sk}_2, x_2))$;
        ii. sample $(x_1 \leftarrow \{0,1\}^{\ell/2}, \mathsf{wPRF.Eval}(\mathsf{sk}_1, x_1))$ using xk_1. Let $x_0 := \mathsf{wPRF.Eval}(\mathsf{sk}_1, x_1) \oplus x_2$ and $x_3 = \mathsf{wPRF.Eval}(\mathsf{sk}_2, x_2) \oplus x_1$.
        iii. output $(x_0\|x_1, x_3\|x_2)$.
    (b) In the challenge phase, receive the challenge (x_2^*, y_2^*) from the external challenger; compute challenge input for C: $(x_0^*\|x_1^*, x_3^*\|x_2^*)$ the same way as above queries.
    (c) C outputs a guess b' and C_2 outputs the same.
5. compute $\tau_2/\perp \leftarrow \mathsf{wPRF.Extract}(\mathsf{xk}_2, C_2)$;
6. output (τ_1, τ_2) (which can include $\perp$'s).

Proof. The unremovability proof is relatively straightforward given the framework of proof in Section 4.2. We will make it brief. Suppose a given C satisfies that $\Pr[\mathsf{Extract}(\mathsf{xk}, C) \notin \mathcal{Q}] \geq \epsilon$, then it must be that $\Pr[\mathsf{wPRF.Extract}(\mathsf{xk}_i, C_i) \notin \mathcal{Q}]$ for both $i = 1, 2$.

Since the extraction keys of the wPRF scheme can be used to simulate the PRF pseudorandomness game perfectly (see Section 5.4.1), the program C_i built in the Extract algorithm is a stage-2 reduction from weak PRP to weak PRF (with key sk_i). Therefore, for each $i = 1, 2$, we can build a reduction $\mathcal{B}_i$ that samples the key $\mathsf{sk}_{j \neq i, j \in [2]}$ and simulates the $\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)$ queries from $\mathcal{A}$ making queries to the unremovability challenger of wPRF with key sk_i.

After $\mathcal{A}$ outputs program C, $\mathcal{B}_i$ makes a program $C_i(\mathsf{sk}_j)'$ that uses black-box access to C: $C_i(\mathsf{sk}_j)'$ behaves the same as C_i built in Extract except using the real secret key sk_j instead of the extraction key xk_j to simulate the game for C.

By the reduction property, either C_1 or C_2 must be a γ_i-good program for winning the weak pseudorandomness game of weak PRP for some non-negligible γ_i: if C_1 has only negligible advantage, then we can replace $\mathsf{wPRF.Eval}(\mathsf{sk}_1, \cdot)$ in the evaluation algorithm with a real random function (or always give the challenge point evaluation $\mathsf{wPRF.Eval}(\mathsf{sk}_1, x_1^*)$ with a random value) and C should still be $(\gamma - \mathsf{negl}(\lambda))$-good. Then we use C in a way where $\mathsf{wPRF.Eval}(\mathsf{sk}_1, \cdot)$ is replaced with random to build a program C_2: if C_2 is also not γ_2-good for any non-negligible γ_2, then we must be able to replace $\mathsf{wPRF.Eval}(\mathsf{sk}_2, \cdot)$ with a random function while C should still be $(\gamma - \mathsf{negl}(\lambda))$-good. But now since all evaluations are random, C should have no advantage. Contradiction.

□

C Watermarkable Implementation of Attribute-based Encryption

In this section, we give a modified version of the watermarkable ABE scheme in [GKM+19] which will imply a watermarkable implementation of the IBE scheme in Section 6.1.1.

C.1 Watermarkable Attribute-based Encryption

Syntax A watermarkable implementation of ABE consists of the following algorithms:

$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{mk}, \mathsf{xk}, \mathsf{msk}, \mathsf{mpk})$: on the security parameter, outputs a master public/secret key, a marking key, an extraction key.

$\mathsf{KeyGen}(\mathsf{msk}, f, \mathsf{mode} \in \{\mathsf{Marked}, \mathsf{Unmarked}\}) \to \mathsf{sk}_f$ on master secret key msk, a policy f, and a mode $\mathsf{mode} \in \{\mathsf{Marked}, \mathsf{Unmarked}\}$ outputs a secret key sk_f.

$\mathsf{Enc}(\mathsf{mpk}, x, m) \to \mathsf{ct}$: on master public key mpk, anttribute $x \in \mathcal{X}$, a message $m \in \mathcal{M}$, outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) \to m'/\bot$: on secret key sk_f and ciphertext ct, output a message $m \in \mathcal{M}$ or $\bot$.

$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau, \mathsf{mode} \in \{\mathsf{MSK}, \mathsf{SKF}\})$: on marking key mk and secret key sk, a message $\tau \in \mathcal{M}_\tau$, and a mode $\mathsf{mode} \in \{\mathsf{MSK}, \mathsf{SKF}\}$, output a marked key sk_τ.

$\mathsf{Extract}(\mathsf{xk}, \mathsf{mpk}, \mathsf{aux}, C)$: on input an extraction key xk, master public key mpk, and program C, output a mark $\tau \in \mathcal{M}_\tau/\bot$.

Remark C.1. *The definition in [GKM+19] only considers running* KeyGen *on an unmarked master secret key and marking the policy-embedded key* sk_f.

For the use of our watermarkable IBE *scheme, we consider marking the master secret key and allowing* KeyGen *to first "mark" a master secret key.*

We additionally allow computing the functional key $\mathsf{sk}_{f,\tau}$ *from a marked master secret key* msk_τ*.When running on* Marked *mode, the* KeyGen *algorithm also takes in a symbol* τ.

Note that not giving out $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ *oracle to* $\mathcal{A}$*, but only* $\mathsf{Mark}(\mathsf{msk}, \cdot)$ *suffices for our use of watermarkable IBE in Section 6. Here we simply prove a slightly stronger security.*

Correctness There exists a negligible funcion $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all $x \in \mathcal{X}$, $f, g \in \mathcal{F}, m \in \mathcal{M}$, when $f(x) = 1$ the following holds:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{WMSetup}(1^\lambda), \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Definition C.2 (ABE security)**.** *The standard notion of security for a KP-ABE scheme is that of full or adaptive security. Specifically, a key-policy attribute-based encryption scheme is said to be fully secure if for every stateful PPT adversary A, there exists a negligible function $negl(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:*

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ ((m_0, m_1), x) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(1^\lambda, \mathsf{mpk}) \\ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, m_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

$\mathcal{A}$ is admissible if all query $f \in \mathcal{F}$ made to oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ *satisfies $f(x) = 0$.*

It is selectively securre if $\mathcal{A}$ needs to output x before seeing mpk.

Definition C.3 (γ-Unremovability)**.** *The γ-Unremovability for a watermarkable ABE scheme says, for all $\lambda \in \mathbb{N}$, and for all PPT admissible stateful adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\Pr\left[\begin{array}{c} \mathsf{Extract}(\mathsf{xk}, \mathsf{mpk}, \mathsf{aux} = (m_0, m_1, x), C) \notin \mathcal{Q} \\ \wedge C \text{ is } \gamma\text{-good} \end{array} : \begin{array}{c} (\mathsf{msk}, \mathsf{mpk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ (\{m_0, m_1\}, x, C) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot), \mathsf{Mark}(\mathsf{mk}, \mathsf{msk}, \cdot)}(1^\lambda, \mathsf{mpk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

where $\mathcal{Q}$ is the set of marks queried by $\mathcal{A}$ and C is a PPT admissible, stateful γ-good adversary in the security game $G_{\mathsf{ABE}}(\mathsf{msk}, \mathsf{mpk}, \mathsf{aux} = (m_0, m_1, x), \cdot)$, more specifically:

$$\Pr\left[C^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \ \mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b, x), b \leftarrow \{0,1\} \ \right] \geq \frac{1}{2} + \gamma.$$

$\mathcal{A}, C$ is admissible if all query $f \in \mathcal{F}$ made to oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ satisfies $f(x) = 0$.

C will be given the KeyGen oracle as $\mathsf{KeyGen}(\mathsf{msk}, \cdot, \mathsf{Unmarked})$ where it only queries on unmarked keys.

A selective variant is requiring $\mathcal{A}$ output x before seeing mpk.

Remark C.4. *As discussed in Section 5.5.2, the above definition is equivalent to letting C output (m_0, m_1) because we can view any distribution over (m_0, m_1) used by C as a convex combination of different message pairs $\{(m_0, m_1)_i\})_i$ and its corresponding strategy such that the overall winning probability is $1/2 + \gamma$. Thus, we can let $\mathcal{A}$ pick the message-pair and corresponding strategy with the largest winning probability and hardcode them into C instead.*

But x needs to be output by $\mathcal{A}$ instead of C, because the $\mathsf{Extract}$ algorithm does not get to see queries of $\mathcal{A}$ and thus cannot make sure if $f(x) = 0$ for all f queried.

Remark C.5 (Modification of [GKM$^+$19] Watermarkable ABE scheme)**.** *In [GKM$^+$19], the unremovability game allows $\mathcal{A}$ to be given only one policy-embedded key sk_f and then allowed to query $\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}_f, \cdot)$ for polynomially many times. In our setting, we let $\mathcal{A}$ query a marked master secret key $\mathsf{Mark}(\mathsf{mk}, \mathsf{msk}, \cdot)$ and the KeyGen algorithm can derive a functional key from a "marked" master secret key. We will show how to adapt [GKM$^+$19]'s construction for our use.*

C.2 Preliminaries: Delegatable ABE and Mixed FE

The [GKM$^+$19]'s watermakable PKE and ABE building blocks are delegatable ABE and mixed FE.

C.2.1 Delegatable ABE

Delegatable ABE has the same syntax with ABE (see Appendix C.1, ignoring the extraction key, marking key and Extract, Mark algorithm) with an additional algorithm Delegate:

$\mathsf{Delegate}(\mathsf{sk}_f, g) \to sk_{f,g}$: on input sk_f and a predicate $g \in \mathcal{F}$, output a delegated key $\mathsf{sk}_{f,g}$.

Correctness of Delegation There exists a negligible funcion $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all $x \in \mathcal{X}$, $f, g \in \mathcal{F}, m \in \mathcal{M}$, when $f(x) = g(x) = 1$ the following holds:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_{f,g}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{WMSetup}(1^\lambda), \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \\ \mathsf{sk}_{f,g} \leftarrow \mathsf{Delegate}(\mathsf{sk}_f, g) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, m) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Delegatable ABE Security The DABE security is the same as ABE security except that the adversary $\mathcal{A}$ queries an oracle $\mathcal{O}(\mathsf{msk}, \cdot)$ that has several modes and takes in a query of the form $(f, \mathsf{index}, \mathsf{mode})$:

- If $\mathsf{mode} = \mathsf{Storekey}$, the challenger generates a new key $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ and store the generated (n, f, sk_f) where n is an index; update $n := n + 1$.

- If $\mathsf{mode} = \mathsf{OutputKey}$: the challenger first checks if there exists a key tuple of the form $(\mathsf{index}, g, sk_g)$. If no such tuple exists or if $g(x) = 1$, it outputs $\perp$. Otherwise, it replies with (index, sk_g).
- If $\mathsf{mode} = \mathsf{DelegateKey}$, then the challenger first checks if there exists a key tuple of the form $(\mathsf{index}, g, sk_g)$. If no such tuple exists or if $g(x) = f(x) = 1$, it outputs $\perp$. Otherwise, it generates $sk_{g,f} \leftarrow \mathsf{Delegate}(\mathsf{sk}_g, f)$ and replies with $(\mathsf{index}, sk_{g,f})$.

C.2.2 Mixed Functional Encryption

An MFE scheme consists of the following algorithms:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$: on input security parameter λ, outputs the public parameters/master public key mpk and master secret key msk.
$\mathsf{Enc}(\mathsf{mpk}) \to \mathsf{ct}$: on master public key, the normal encryption outputs ciphertext ct.
$\mathsf{SKEnc}(\mathsf{msk}, f) \to \mathsf{ct}$: on master secret key msk and fucnction $f \in \mathcal{F}$, outputs a ciphertext ct.
$\mathsf{KeyGen}(\mathsf{msk}, m) \to \mathsf{sk}_m$: on master secret key msk and message m, outputs a key sk_m.
$\mathsf{Dec}(\mathsf{sk}_m, \mathsf{ct}) \to \{0,1\}$: on secret key sk_m and ciphertext ct, outputs a bit.

Correctness A mixed functionl encryption scheme is correct if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}, f \in \mathcal{F}, m \in \mathcal{M}$:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_m, \mathsf{ct}) = 1 : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{sk}_m \leftarrow \mathsf{KeyGen}(\mathsf{msk}, m) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_m, \mathsf{ct}) = f(m) : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{sk}_m \leftarrow \mathsf{KeyGen}(\mathsf{msk}, m) \\ \mathsf{ct} \leftarrow \mathsf{SKEnc}(\mathsf{msk}, f) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

q-Bounded function indistinguishability Let $q = q(\lambda)$ be any fixed polynomial. A mixed functional encryption scheme is said to satisfy q-bounded function indistinguishability security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SKEnc}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (f_0, f_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SKEnc}(\mathsf{msk},\cdot)}(1^\lambda, \mathsf{mpk}) \\ b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{SKEnc}(\mathsf{mpk}, f_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathcal{A}$ can make at most q queries to $\mathsf{SKEnc}(\mathsf{msk}, \cdot)$ oracle; every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must satisfy $f_0(m) = f_1(m)$.

q-Bounded accept indistinguishability Let $q = q(\lambda)$ be any fixed polynomial. A mixed functional encryption scheme is said to satisfy q-bounded accept indistinguishability security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for every $\lambda \in \mathbb{N}$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SKEnc}(\mathsf{msk},\cdot)}(\mathsf{ct}_b) = b : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ f^* \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathsf{SKEnc}(\mathsf{msk},\cdot)}(1^\lambda, \mathsf{mpk}) \\ b \leftarrow \{0,1\}, \mathsf{ct}_1 \leftarrow \mathsf{SKEnc}(\mathsf{msk}, f^*), \mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{mpk}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathcal{A}$ can make at most q queries to $\mathsf{SKEnc}(\mathsf{msk}, \cdot)$ oracle; every secret key query m made by adversary $\mathcal{A}$ to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle must satisfy $f^*(m) = 1$.

QTrace **Jump-Finding Algorithm** The [GKM+19] watermarkable ABE schemes Extract procedure also additionally uses an algorithm QTrace that is widely used in the traitor-tracing literature. QTrace on inpput parameters $(\lambda, N, q, \delta, \gamma)$ and given black box access to a program Q, runs in time $t = \mathsf{poly}(\lambda, \log N, q, 1/\delta)$. Since detailed discussions are not essential to our presentation on the modified scheme, we refer the readers to [GKM+19] Section 4.2.3 for details.

C.3 Construction

Now we give the construction for watermarkable implementation of ABE, which is modified from [GKM+19] Section 4.3.

- Given a delegatable ABE scheme DABE = (DABE.Setup, DABE.KeyGen, DABE.Enc, DABE.Dec, DABE.Delegate) and a MFE scheme (MFE.Setup, MFE.KeyGen, MFE.SKEnc, MFE.Enc, MFE.Dec).
- Let $\mathcal{M}_\tau$ be the marked space, $\mathcal{X} = \{0,1\}^{\ell_2}$ the attribute space, and $\mathcal{C} \subseteq \mathsf{Functions}[\mathcal{X}, \{0,1\}]$ be thepredicate class for the target watermarkable ABE scheme, with parameters ℓ_1, ℓ_2.
- Let $\mathcal{Y} = \{0,1\}^{\ell_2+\kappa}$ and $\mathcal{D} = \mathcal{C} \cup \{\mathsf{MFE.Dec}\} \cup \{\mathsf{MFE.Dec} \wedge C\}_{C \in \mathcal{C}}$.
- the DABE scheme has attribute class $\mathcal{Y}$, message space $\mathcal{M}$, predicate class cD. Elements in $\mathcal{Y}$ hav the following format $(x \in \mathcal{X}, \mathsf{MFE.ct})$.
- Let γ be the unremovability parameter.

```
WMSetup($1^\lambda$) → (mpk, msk, mk, xk):
    1. compute MFE.mpk, MFE.msk ← MFE.Setup($1^\lambda$);
    2. compute (DABE.mpk, DABE.msk) ← DABE.Setup($1^\lambda$);
    3. output mpk = (MFE.mpk, DABE.mpk); msk = (MFE.msk, DABE.msk);
       mk = xk = (MFE.mpk, MFE.msk, DABE.msk).
KeyGen(msk, $f$, mode, $\tau$) → sk$_f$: Let msk = (MFE.msk, DABE.msk)
  If mode = Unmarked:
    1. Let $\tilde{f} : \{0,1\}^{\ell_2+\kappa} \to \{0,1\}$ denote the predicate $\tilde{f}(x,c) = f(x), x \in \{0,1\}^{\ell_2}, c \in \{0,1\}^\kappa$.
    2. output sk$_f$ ← DABE.KeyGen(msk, $\tilde{f}$).
  If mode = Marked:
    1. compute sk$_\tau$ ← MFE.KeyGen(MFE.msk, $\tau$)
    2. Let $g_\tau$ denote the mixed FE decryption circuit with sk$_\tau$ hardwired, i.e. $g_\tau$ = MFE.Dec(sk$_\tau$, $\cdot$);
    3. compute sk$_{\tilde{g_\tau}}$ ← DABE.KeyGen(DABE.msk, $\tilde{g_\tau}$) where $\tilde{g_\tau}(x,c) = g_\tau(c), \forall x \in \{0,1\}^{\ell_2}, c \in$
       $\{0,1\}^\kappa$;
    4. compute and output delegated key sk$_{f,\tilde{g_\tau}}$ ← DABE.Delegate(sk$_{\tilde{g_\tau}}$, $f$)
Enc(mpk, $x$, $m$):
  Let mpk = (MFE.mpk, DABE.mpk); compute ct$_{\mathsf{MFE}}$ ← MFE.Enc(MFE.mpk).
  Next compute ad output ct ← DABE.mpk, ($x$, ct$_{\mathsf{MFE}}$), $m$) where ($x$, ct$_{\mathsf{MFE}}$) is the attribute.
Dec(sk, ct) → $m/\perp$: output $m$ ← DABE.Dec(sk, ct).
Mark(mk, msk, $\tau$):
    1. mk := (MFE.mpk, MFE.msk);
    2. compute sk$_\tau$ ← MFE.KeyGen(MFE.msk)
    3. Let $g_\tau$ denote the mixed FE decryption circuit with sk$_\tau$ hardwired, i.e. $g_\tau$ = MFE.Dec(sk$_\tau$, $\cdot$);
```

4. compute and output $\mathsf{sk}_{\tilde{g_\tau}} \leftarrow \mathsf{DABE.KeyGen}(\mathsf{DABE.msk}, \tilde{g_\tau})$ where $\tilde{g_\tau}(x, c) = g_\tau(c), \forall x \in \{0,1\}^{\ell_2}, c \in \{0,1\}^\kappa$

$\mathsf{Extract}(\mathsf{xk}, \mathsf{mpk}, \mathsf{aux} = (x, m_0, m_1), C, q)$:

Let $\mathsf{xk} = (\mathsf{MFE.mpk}, \mathsf{MFE.msk}, \mathsf{DABE.msk})$; $\mathsf{mpk} = (\mathsf{DABE.mpk}, \mathsf{MFE.mpk})$;
The extraction algorithms runs the QTrace algorithm as $\tau \leftarrow \mathsf{QTrace}^{Q_C}(\lambda, 2^{\ell_1}, q, \delta, \gamma)$ where q is the bounded parameter for the MFE scheme (upper bound on the number of queries to the $\mathsf{SKEnc}(\mathsf{msk}, \cdot)$ oracles), γ is the unremovability paramter and $\delta = \gamma/(5 + 2\ell_1 q)$. Q_C is simulated as follows:

On input $\tau \leftarrow [0, 2^{\ell_1}]$:

- compute MFE ciphertext as $\mathsf{ct}_{\mathsf{MFE}} \leftarrow \mathsf{MFE.SKEnc}(\mathsf{MFE.msk}, \mathsf{comp}_\tau)$, where comp_τ is the comparison function that on input z, output 1 if and only if $z \geq \tau$.
- sample a bit $b \leftarrow \{0,1\}$ and compute $\mathsf{ct}_b \leftarrow \mathsf{DABE.Enc}(\mathsf{DABE.mpk}, (x, \mathsf{ct}_{\mathsf{MFE}}), m_b)$ where $(x, \mathsf{ct}_{\mathsf{MFE}})$ is the attribute input.
- compute $b' \leftarrow C^{\mathsf{KeyGen}(\mathsf{msk},\cdot,\mathsf{Unmarked})}(\mathsf{ct})$ and output 1 if $b' = b$ and 0 otherwise. C has oracle access to $\mathsf{KeyGen}(\mathsf{msk}, \cdot, \mathsf{Unmarked})$, which can be simulated using $\mathsf{DABE.msk}$ provided in xk.

Correctness and Other Properties The correctness of decryption, correctness of extraction, functionality-preserving and ABE security can directly follow the same proof as in [GKM$^+$19] so we omit them here.

We can observe that the above scheme also satisfies the additional extraction key simulation property and extraction syntax also satisfy our requirement on watermarkable implementation: the extraction key has $\mathsf{DABE.msk}$ embedded so enough to simulate the $\mathsf{KeyGen}(\mathsf{msk}, \cdot, \mathsf{Unmarked})$ queries for program C. The entire extraction algorithm follows the format that runs the simulated stage-2 security game for ABE.

Security Proof The security proof will be similar to the proof in [GKM$^+$19]. Overall, our changes are minor. To avoid repetitive work of doing a same proof verbatimly, we omit the parts of our proof which are exactly the same as [GKM$^+$19] and refer the readers to Section 4.3.1 of [GKM$^+$19] "Proof of Theorem 4.15(Unremovability)" part. We only discuss the major changes to make to adjust to our construction:

All discussions and lemmas before Lemma 4.18 will be exactly the same when moved to our case, except some minor changes:

1. We adjust Experiment $\mathsf{GetCircuit}_{\mathcal{A}}(\lambda)$ in Figure 2 of [GKM$^+$19] to fit our unremovability game: $\mathcal{A}$ is allowed to query $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ for arbitrarily polynomial times. We will discuss how this oracle is simulated when doing different reductions.
2. In Lemma 4.16 and 4.17 invokes security of MFE: in our setting, the reduction algorithm will additionally query the $\mathsf{MFE.KeyGen}(\mathsf{msk}, \cdot)$ oracle when dealing with the adversary $\mathcal{A}$s $\mathsf{KeyGen}(\mathsf{msk}, \cdot, \mathsf{Marked}, \cdot)$ queries. But these queries have exactly the same format as when answering the marking queries $\mathsf{Mark}(\mathsf{mk}, \mathsf{msk}, \tau)$. Moreover, in both the q-bounded function indistinguishability and accept indistinguishability games, the number of queries a reduction can make to oracle $\mathsf{MFE.KeyGen}(\mathsf{msk}, \cdot)$ are unbounded polynomial. Therefore, the proof of Lemma 4.16 and 4.17 are unaffected.

We mainly discuss the changes to Lemma 4.18 and take some steps from [GKM+19] verbatimly for completeness: Let $\mathcal{B}$ be the reduction to security of DABE.

1. Adversary $\mathcal{A}$ chooses a challenge attribute x and sends it over to $\mathcal{B}$, since we consider only selective security.
2. The DABE challenger samples a DABE key pair $(\mathsf{DABE.mpk}, \mathsf{DABE.msk}) \leftarrow \mathsf{DABE.Setup}(1^\lambda)$. Algorithm $\mathcal{B}$ samples a mixed FE key pair $(\mathsf{MFE.mpk}, \mathsf{MFE.msk}) \leftarrow \mathsf{MFE.Setup}(1^\lambda)$, and sends $\mathsf{MFE.mpk}$ to $\mathcal{B}$. , and sends the public key $\mathsf{mpk} = (\mathsf{MFE.mpk}, \mathsf{DABE.mpk})$ to $\mathcal{A}$.
3. $\mathcal{B}$ computes $\mathsf{ct}^*_{\mathsf{MFE}} \leftarrow \mathsf{MFE.SKEnc}(\mathsf{MFE.msk}, \mathsf{comp}_{2^\ell_1})$ and commits to the challenge attribute $x^* = (x, \mathsf{ct}^*_{\mathsf{MFE}})$.
4. When $\mathcal{A}$ makes a query to the $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ and $\mathsf{Mark}(\mathsf{mk}, \mathsf{msk}, \cdot)$ oracles, $\mathcal{B}$ simulates the answer as follows:
   (a) If $\mathcal{A}$ makes a marking query on message τ: $\mathcal{B}$ computes the function $g_{\tilde{\mathsf{sk}}_\tau}$ as described in the construction on its own; using $\mathsf{MFE.KeyGen}$; then it sends query $(g_{\tilde{\mathsf{sk}}_\tau}, \perp, \mathsf{StoreKey})$ to $\mathcal{O}(\mathsf{msk}, \cdot)$.
   Note that $\tau \in [0, 2^{\ell_1}]$, so $g_{\tilde{\mathsf{sk}}_\tau}(x^*) = 0$ is satisfied with overwhelming probability.
   $\mathcal{B}$ gets reply in the form $(\mathsf{index}, \perp)$; $\mathcal{B}$ then makes another query $(g_{\tilde{\mathsf{sk}}_\tau}, \mathsf{index}, \mathsf{OutputKey})$ and gets back $(\mathsf{index}, \mathsf{DABE.sk}_{g_{\mathsf{sk}_\tau}})$.
   $\mathcal{B}$ makes a table that stores the following information $(\mathsf{index}, \tau, \mathsf{sk}_{g_{\mathsf{sk}_\tau}})$ in each entry.
   (b) If $\mathcal{A}$ makes KeyGen query of the format $(\mathsf{Marked}, \tau, f)$: $\mathcal{B}$ first checks if an entry containing τ is stored in its table in the format $(\mathsf{index}, \tau, \mathsf{sk}_{g_{\mathsf{sk}_\tau}})$; if so $\mathcal{B}$ prepares $\tilde{f}(x, c) = f(x)$ and queries the $\mathcal{O}(\mathsf{msk}, \cdot)$ oracle on input $(\mathsf{index}, f, \mathsf{DelegateKey})$ and gets back $(\mathsf{index}, \mathsf{DABE.sk}_{g_{\mathsf{sk}_\tau}, \tilde{f}})$. $\mathcal{B}$ sends $\mathsf{DABE.sk}_{g_{\mathsf{sk}_\tau}, \tilde{f}}$ to $\mathcal{A}$.
   If there is no such entry, $\mathcal{B}$ will query $(g_{\tilde{\mathsf{sk}}_{\tau'}}, \perp, \mathsf{StoreKey})$ first and then query $(\mathsf{index}', f, \mathsf{DelegateKey})$ to get the same output.
   (c) If $\mathcal{A}$ makes a KeyGen query in the format $(\mathsf{Unmarked}, f)$, then $\mathcal{B}$ prepares $\tilde{f}(x, c) = f(x)$ and first queries $(\tilde{f}, \perp, \mathsf{StoreKey})$ to get back some $(\mathsf{index}_{\tilde{f}}, \perp)$; then queries again $(\mathsf{index}_{\tilde{f}}, \tilde{f}, \mathsf{OutputKey})$ for $sk_{\tilde{f}}$. Note it is also satisfied that $\tilde{f}(x^*) = f(x) = 0$.
5. At the end of the game, $\mathcal{A}$ outputs a pair of messages (m_0, m_1) and a circuit C^*.
6. $\mathcal{B}$ then sends (m_0, m_1) to DABE challenger and gets a ciphrtext ct^*. $\mathcal{B}$ samples a random bit $\beta \leftarrow \{0, 1\}$ and computes a fresh ciphertext $\mathsf{ct} \leftarrow \mathsf{DABE.Enc}(\mathsf{DABE.mpk}, (x, \mathsf{ct}_{\mathsf{MFE}}), m_\beta)$, where $\mathsf{ct}_{\mathsf{MFE}} \leftarrow \mathsf{MFE.SKEnc}(\mathsf{MFE.msk}, \mathsf{comp}_{2^{\ell_1}})$.
   C^* will continue to make queries but only allowed to query $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{Unmarked}, \cdot)$ oracle. $\mathcal{B}$ can answer these queries by querying $\mathcal{O}(\mathsf{msk}, \cdot)$ with $(\tilde{f}, \perp, \mathsf{StoreKey})$, because $\mathcal{B}$ is allowed to make such queries in DABE game:
   - If C makes a KeyGen query in the format $(\mathsf{Unmarked}, f)$, then $\mathcal{B}$ prepares $\tilde{f}(x, c) = f(x)$ and first queries $(\tilde{f}, \perp, \mathsf{StoreKey})$ to get back some $(\mathsf{index}_{\tilde{f}}, \perp)$; then queries again $(\mathsf{index}_{\tilde{f}}, \tilde{f}, \mathsf{OutputKey})$ for $sk_{\tilde{f}}$. Note it is also satisfied that $\tilde{f}(x^*) = f(x) = 0$.
7. Finally, $\mathcal{B}$ runs the decryption circuit C^* on ct^* and ct, and if $C^{\mathsf{KeyGen}(\mathsf{msk}, \mathsf{Unmarked}, \cdot)}(\mathsf{ct}^*) = C^{\mathsf{KeyGen}(\mathsf{msk}, \mathsf{Unmarked}, \cdot)}(\mathsf{ct})$, it outputs $b' = \beta$. Otherwise, it outputs $b' = 1 - \beta$.

The rest of the analysis is the same as in the original proof.

D Watermarkable Implementation of Digital Signatures

D.1 Preliminaries: Constrained Signatures

The building block of a bounded collusion resistant watermarkable implementation of digital signatures in [GKM+19] relies on the following building block.

Constrained Signatures A constrained signature with message space $\mathcal{M}$ and consraint family $\mathcal{F} \subseteq \mathsf{Function}[\mathcal{M}, \{0,1\}]$ is a tuple of algorithms:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{msk})$. On input the security parameter λ, the setup algorithm outputs the verification key vk and the master secret key msk.

$\mathsf{Sign}(\mathsf{msk}, m) \to \mathsf{sig}$. On input the master signing key msk and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature sig.

$\mathsf{Verify}(\mathsf{vk}, m, \mathsf{sig}) \to b$. On input the verification key vk, a message $m \in \mathcal{M}$, and a signature sig, the verification algorithm outputs a bit $b \in \{0,1\}$.

$\mathsf{Constrain}(\mathsf{msk}, f) \to \mathsf{sk}_f$: On input the master signing key msk and a function $f \in \mathcal{F}$, the constrain algorithm outputs a constrained key sk_f.

$\mathsf{ConstrainSign}(\mathsf{sk}_f, m) \to \mathsf{sig}$. On input a constrained key sk_f and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature sig.

Correctness A constrained signature scheme is correct if for all messages $m \in \mathcal{M}$ and key pair $(vk, msk) \leftarrow \mathsf{Setup}(1^\lambda)$:

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

In addition, for all constraints $f \in \mathcal{F}$ where $f(m) = 1$,

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

Constrained Unforgeability A constrained signature scheme is secure if for every stateful admissible PPT $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda in \mathbb{N}$:

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m^*, \mathsf{sig}^*) = 1 : \begin{array}{c} (\mathsf{msk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (m^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{msk},\cdot), \mathsf{Constrain}(\mathsf{msk},\cdot)}(1^\lambda, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

where $\mathcal{A}$ is admissible if (1) it does not make a signing query on message m^* ; and (2) it does not make a constrained key query for any function $f \in \mathcal{F}$ such that $f(m^*) = 1$.

D.2 Definition: Watermarkable Signatures

A watermarkable signatures scheme consists of the following algorithms:

- $\mathsf{WMSetup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{xk}, \mathsf{mk})$: on input security parameter outputs verification key vk, signing key sk; extraction key xk, marking key mk.
- $\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \to \mathsf{sk}_\tau$. On input the marking key mk, a signing key sk, and a mark $\tau \in \mathcal{M}_\tau$, the marking algorithm outputs a marked key sk_τ.

- $\mathsf{Sign}(\mathsf{sk}, m) \to \mathsf{sig}$. On input a signing key sk and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature sig. $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to 0/1$. On input a verification key vk, a message $m \in \mathcal{M}$, and a signature sig, the verification algorithm outputs a bit to signify whether the signature is valid or not.
- $\mathsf{Extract}(\mathsf{xk}, \mathsf{vk}, C) \to \tau/\bot$. On input the extraction key xk, a verification key vk, and a circuit C, the extraction algorithm either outputs a mark $\tau \in \mathcal{M}_\tau$ or $\bot$.

The correctness, meaningfulness and other properties are natural to derive from the general watermarking definition Section 4.1. We refer to [GKM+19] for more details and only present functionality-preserving and security defintions.

Functionality Preserving A watermarkable signature scheme satisfies the functionality-preserving property if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}, (\mathsf{vk}, \mathsf{sk}, \mathsf{mk}, \mathsf{xk}) \leftarrow \mathsf{WMSetup}(1^\lambda)$, $m \in \mathcal{M}, \tau \in \mathcal{M}_\tau$, the following holds:

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, \mathsf{wSecEval}(\mathsf{sk}_\tau, m)) = 1 : \begin{array}{c} (\mathsf{vk}, \mathsf{sk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda), \\ \mathsf{sk}_\tau \leftarrow \mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

We provide the unremovability definition, which is different from the one defined in [GKM+19], but their scheme can be modified to satisfy this definition.

γ-Unremovability For every stateful γ-unremovable admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\mathsf{Extract}(\mathsf{xk}, \mathsf{vk}, C) \notin \mathcal{Q} \wedge C \text{ is } \gamma\text{-good} : \begin{array}{c} (\mathsf{sk}, \mathsf{vk}, \mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{WMSetup}(1^\lambda) \\ C \leftarrow \mathcal{A}^{\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \cdot)}(1^\lambda, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

where $\mathcal{Q}$ is the set of marks queried by $\mathcal{A}$ and C is said to be a (PPT admissible, stateful) γ-good adversary if:

$$\Pr\left[C^{\mathsf{Sign}(\mathsf{sk}, \cdot)} \to (m^*, \mathsf{sig}^*) : \ \mathsf{Verify}(\mathsf{vk}, m^*, \mathsf{sig}^*) = 1\right] \geq \gamma.$$

C is admissible if and only if it does not query $\mathsf{Sign}(\mathsf{sk}, \cdot)$ on m^*.

Discussions on Security Definitions and [GKM+19] construction In the [GKM+19]'s watermarkable signature scheme, the unmarked Sign function and the circuit used to compute signatures using marked keys have different output distributions, if we let $\mathcal{A}$ query $\mathsf{Sign}(\mathsf{sk}, \cdot)$ oracle in the first stage but allow C to choose the challenge message to sign on, then there can be an attack. Because the [GKM+19] scheme 's $\mathsf{Sign}(\mathsf{sk}, \cdot)$ and marked signing circuit have different output distributions. If C can choose its own m^* but the Extract algorithm does not know which queries $\mathcal{A}$ has made in stage 1, then C can be hardcoded with some m^*, sig' where watermark cannot be extracted. We thus give the above definition, which suffice for our applications 1.

D.3 Construction

We show the following modified construction from [GKM+19] based on a constrained signature scheme CSig defined in Appendix D.1:

- Let $\mathcal{T}' = \mathcal{T} \cup \{\perp\}$. For a mark $\tau^* \in \mathcal{T}$, let $f_{\tau^*} : \mathcal{T}' \times \mathcal{M} \to \{0,1\}$ be the function $f_{\tau^*}(\tau, m) = 1$ if $\tau = \tau^*$ and 0 otherwise.

$\mathsf{WMSetup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{mk}, \mathsf{xk})$: outputs a signing/verification key-pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{CSig.Setup}(1^\lambda)$; $\mathsf{mk} = \perp, \mathsf{xk} = \mathsf{sk}$.

$\mathsf{Sign}(\mathsf{sk}, m) \to \mathsf{sig}$: On input a signing key sk, and a message $m \in \mathcal{M}$, the signing algorithm signs $\mathsf{sig}' \leftarrow \mathsf{CSig.Sign}(\mathsf{sk}, (\perp, m))$, and outputs the signature $\mathsf{sig} = (\perp, \mathsf{sig}')$.

$\mathsf{Verify}(\mathsf{vk}, m, \mathsf{sig}) \to b$. On input a verification key vk, a message $m \in \mathcal{M}$, and a signature $\mathsf{sig} = (\tau', \mathsf{sig}')$; the verification algorithm outputs $b \leftarrow \mathsf{CSig.Verify}(\mathsf{vk}, (\tau', m), \mathsf{sig}')$.

$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau) \to C$. On input a marking key $\mathsf{mk} = \perp$, a signing key sk, and a mark $\tau \in \mathcal{T}$, the marking algorithm computes $\mathsf{sk}_\tau \leftarrow \mathsf{CSig.Constrain}(\mathsf{sk}, f_\tau)$ and outputs a circuit $C_\tau : \mathcal{M} \to \mathcal{SIG}$ where $C_\tau(\cdot) := (\tau, \mathsf{CSig.ConstrainSign}(\mathsf{sk}_\tau, (\tau, \cdot)))$.

$\mathsf{Extract}(\mathsf{xk}, \mathsf{vk}, C) \to \tau/\perp$: on $\mathsf{xk} = \mathsf{CSig.sk}, \mathsf{vk} = \mathsf{CSig.vk}$ and circuit $C : \mathcal{M} \to \mathcal{SIG}$, perform the following for $T = \lambda/\gamma$ times where γ is the unremovability parameter:

- For $i \in [T]$: compute $(m_i = (\tau_i', m_i'), \mathsf{sig}_i) \leftarrow C^{\mathsf{Sign}(\mathsf{sk},\cdot)}$. If $\mathsf{CSig.Verify}(\mathsf{vk}, (\tau_i, m_i'), \mathsf{sig}_i) = 1$ and m_i' has not been queried, abort and output τ_i'.

The correctness, functionlity-preserving, meaningfulness proof will all follow exactly from [GKM+19]. Even though our extraction algorithm is different, the unremovability proof will follow similarly: any valid signatures provided by C must contain some $\tau_i \in \mathcal{Q}$ of the marking queries, otherwise it helps break the constrained unforgeable security of the constrained signature. The only change is that the reduction does not answer any $\mathsf{Sign}(\mathsf{sk}, \cdot)$ queries in the first stage, and thus do not need to query $\mathsf{CSig.Sign}(\mathsf{sk}, \cdot)$ oracle when interacting with $\mathcal{A}$, but only when interacting with the circuit C produced by $\mathcal{A}$.

E Watermarkable CCA-secure Hybrid Encryption (Key Encapsulation Scheme)

Theorem E.1. *Assuming LWE, there exists secure watermarkable implementation of a CCA-secure hybrid encryption scheme.*

The assumption of LWE comes from the need of watermarkable CCA-secure PKE.

Construction A watermarkable implementation of CCA-secure hybrid encryption $\mathsf{wHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Mark}, \mathsf{Extract})$ can be built from a CCA-secure secret key encryption scheme $\mathsf{SKE} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ and a watermarkable implementatin of CCA-secure public key encryption $\mathsf{wPKE} = (\mathsf{wPKE.WMSetup}, \mathsf{wPKE.Enc}, \mathsf{wPKE.Dec}, \mathsf{wPKE.Mark}, \mathsf{wPKE.Extract})$ as follows:

- $\mathsf{KeyGen}(\lambda)$: compute $(\mathsf{wPKE.pk}, \mathsf{wPKE.sk}, \mathsf{wPKE.xk}, \mathsf{wPKE.mk}) \leftarrow \mathsf{wPKE.WMSetup}(\lambda)$. Output $\mathsf{sk} = \mathsf{wPKE.sk}$; $\mathsf{pk} = \mathsf{wPKE.pk}$; $\mathsf{xk} = \mathsf{wPKE.xk}$; $\mathsf{mk} = \mathsf{wPKE.mk}$
- $\mathsf{Enc}(\mathsf{pk}, m)$: compute $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.KeyGen}(\lambda)$; then compute $\mathsf{ct}_1 \leftarrow \mathsf{wPKE.Enc}(\mathsf{pk}, \mathsf{SKE.sk})$; compute $\mathsf{ct}_2 \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, m)$. Output $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: parse $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$; compute $\mathsf{sk}' \leftarrow \mathsf{wPKE.Dec}(\mathsf{sk}, \mathsf{ct}_1)$ and then $m' \leftarrow \mathsf{SKE.Dec}(\mathsf{sk}', \mathsf{ct}_2)$. Output m'.
- $\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau)$: parse $\mathsf{mk} = \mathsf{wPKE.mk}$; $\mathsf{sk} = \mathsf{wPKE.sk}$; output $\mathsf{sk}_\tau \leftarrow \mathsf{wPKE.Mark}(\mathsf{sk}, \tau)$;

- $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux} = \bot, C)$:
  - parse $\mathsf{xk} := \mathsf{wPKE.xk}$; $\mathsf{pk} := \mathsf{wPKE.pk}$. Create the following circuit C_{PKE} with black-box access to C:
    - C_{PKE} is hardcoded with pk and simulates the CCA-PKE game for C as follows:
    - give pk to C; for C's decryption queries $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$: query an external oracle $\mathsf{sk}' \leftarrow \mathsf{wPKE.Dec}(\mathsf{wPKE.sk}, \mathsf{ct}_1)$; output $m' \leftarrow \mathsf{SKE.Dec}(\mathsf{sk}', \mathsf{ct}_2)$.
    - C submits challenge messages (m_0, m_1); C_{PKE} submits $(\mathsf{sk}_0 \leftarrow \mathsf{SKE.KeyGen}(1^\lambda), \mathsf{sk}_1 \leftarrow \mathsf{SKE.KeyGen}(1^\lambda))$ to the external challenger; C_{PKE} receives challenge ciphertext ct^* from the challenger and C_{PKE} sends $\mathsf{ct}^{**} = (\mathsf{ct}^*, \mathsf{ct}_2^* \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_0, m_b))$ to C, where $m_b \leftarrow \{m_0, m_1\}$.
    - C continues to simulate the decryption oracle as above to decrypt only valid ciphertexts.
    - If C outputs $b' = b$, then C_{PKE} outputs guess 0; else it outputs guess 1.
  - output $\tau/\bot \leftarrow \mathsf{wPKE.Extract}(\mathsf{wPKE.xk}, \mathsf{wPKE.pk}, \mathsf{aux} = \bot, C_{\mathsf{PKE}})$.

Now we prove Theorem E.1

Proof. Suppose there exists adversary $\mathcal{A}$ that breaks the γ-unremovability of watermarkable CCA-secure PKE for some non-negligible γ, i.e. $\mathcal{A}^{\mathsf{Dec}(\mathsf{sk},\cdot),\mathsf{Mark}(\mathsf{mk},\mathsf{sk},\cdot)(\mathsf{pk}}$ produces some program C such that $\Pr[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C) \notin \mathcal{Q}] \geq \epsilon$ for some non-negligible ϵ, whereas C satisfies $\Pr[G_{CCA}(\mathsf{sk}, \mathsf{pk}, C) = 1] \geq \frac{1}{2} + \gamma$ (for G_{CCA} see Section 5.5.2). The probability $\Pr[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C) \notin \mathcal{Q}]$ is taken over the randomness in $\mathsf{WMSetup}$ and the probability $\Pr[G_{CCA}(\mathsf{sk}, \mathsf{pk}, C) = 1]$ is take over the randomness used in G_{CCA}.

We will show that we can either break γ'-unremovability of the watermarkable wPKE or CCA-security of SKE.

For any $(\mathsf{sk}, \mathsf{pk}, \mathsf{xk}, \mathsf{mk})$ generated by $\mathsf{WMSetup}(1^\lambda)$ and any $\mathcal{A}^{\mathsf{Dec}(\mathsf{sk},\cdot),\mathsf{Mark}(\mathsf{mk},\mathsf{sk},\cdot)}(\mathsf{pk})$ producing a γ-good C such that $\Pr[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C) \notin \mathcal{Q}] \geq \epsilon$, one of the following cases must hold:

- **Case 1: the program C_{PKE} created during the execution of $\mathsf{Extract}(\mathsf{xk}, C)$ is a γ'-good adversary for the stage-2 game $G_{\mathsf{CCA-PKE}}(\mathsf{sk}, \mathsf{pk}, \cdot)$ for some non-negligible γ'.** The reduction $\mathcal{B}_{\mathsf{wPKE}}$ works as follows:
  - $\mathcal{B}_{\mathsf{wPKE}}$ receives wPKE.pk from the challenger and sends to $\mathcal{A}$. $\mathcal{B}_{\mathsf{wPKE}}$;
  - for $\mathcal{A}$'s marking queries : query the challenger's marking oracle $\mathsf{Mark}(\mathsf{wPKE.sk}, \cdot)$ and forward the output.

  After $\mathcal{A}$ outputs program C, $\mathcal{B}_{\mathsf{wPKE}}$ creates circuit C'_{wPKE} with black-box access to C:
  - C'_{wPKE} is hardcoded with pk and makes external queries to simulate the decryption oracle for C:
    - for C's decryption queries $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$: query the challenger's decryption oracle $\mathsf{sk}' \leftarrow \mathsf{wPKE.Dec}(\mathsf{wPKE.sk}, \mathsf{ct}_1)$; output $m' \leftarrow \mathsf{SKE.Dec}(\mathsf{sk}', \mathsf{ct}_2)$.
  - C submits challenge messages (m_0, m_1); C_{PKE} submits $(\mathsf{sk}_0 \leftarrow \mathsf{SKE.KeyGen}(1^\lambda), \mathsf{sk}_1 \leftarrow \mathsf{SKE.KeyGen}(1^\lambda))$ to the external challenger; C_{PKE} receives challenge ciphertext ct^* from the challenger and C_{PKE} sends $\mathsf{ct}^{**} = (\mathsf{ct}^*, \mathsf{ct}_2^* \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_0, m_b))$ to C, where $m_b \leftarrow \{m_0, m_1\}$.
  - C continues to simulate the decryption oracle as above to decrypt only valid ciphertexts.

– If C outputs $b' = b$, then C_{PKE} outputs guess 0; else it outputs guess 1.

By the design of out $\mathsf{Extract}$ and by our assumption, we must have that $\Pr[\mathsf{wPKE.Extract}(\mathsf{wPKE.xk}, \mathsf{wPKE.pk}, \mathsf{aux}, C_{\mathsf{PKE}}) \notin \mathcal{Q}] \geq \epsilon$ where C_{PKE} is the program created dring $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C)$.
For any $(\mathsf{xk}, \mathsf{mk}, \mathsf{sk}, \mathsf{pk})$, it is easy to see that program C_{PKE} created by $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C)$'s input-output behavior is the same as C'_{wPKE} above. Therefore, if $\Pr[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C_{\mathsf{PKE}}) \in \mathcal{Q}]$ is γ'-good and $\Pr[\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C_{\mathsf{PKE}})] \geq \epsilon$, so will C'_{wPKE} satisfy these two conditions. Thus $\mathcal{B}_{\mathsf{wPKE}}$ breaks the γ'-unremovability.

- **Case 2: the program C_{PKE} created during the execution of $\mathsf{Extract}(\mathsf{xk}, C)$ is not a γ'-good adversary for the stage-2 game $G_{\mathsf{CCA-PKE}}(\mathsf{sk}, \mathsf{pk}, \cdot)$ for any non-negligible γ'.** In this case, we must have that $|\Pr[C_{\mathsf{PKE}} \to 0 | \mathsf{ct}^* = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{sk}_0)] - \Pr[C_{\mathsf{PKE}} \to 0 | \mathsf{ct}^* = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{sk}_1)] = |\Pr[C \text{ outputs } b' = b | \mathsf{ct}^* = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{sk}_0)] - \Pr[C \text{ outputs } b' = b | \mathsf{ct}^* = \mathsf{PKE.Enc}(\mathsf{pk}, \mathsf{sk}_1)]| \leq \mathsf{negl}(\lambda)$;
  Thus we can switch to doing the following in running $\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, C)$: when C submits challenge messages (m_0, m_1); C_{PKE} sends $\mathsf{ct}^{**} = (\mathsf{ct}_1^* = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{sk}_1), \mathsf{ct}_2^* \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_0, m_b))$ to C, where $m_b \leftarrow \{m_0, m_1\}$ where ct^* is always $\mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{sk}_1)$.
  Since C is still $(\gamma - \mathsf{negl}(\lambda))$ good, we will build a reduction to break IND-CCA security of SKE.
  The reduction $\mathcal{B}_{\mathsf{SKE}}$ samples wPKE's keys on its own and can query the oracles $\mathsf{SKE.Enc}(\mathsf{sk}_0, \cdot)$, $\mathsf{SKE.Dec}(\mathsf{sk}_0, \cdot)$ provided by the SKE challenger.
  After $\mathcal{A}$ outputs C, $\mathcal{B}_{\mathsf{SKE}}$ enters stage-2 of reduction and simulates the encryption and decryption oracles by submitting the decryption queries to the encryption, decryption oracles $\mathsf{SKE.Enc}(\mathsf{sk}_0, \cdot), \mathsf{SKE.Dec}(\mathsf{sk}_0, \cdot)$ in the CCA-security game of SKE. In the challenge phase, C submits challenge messages (m_0, m_1); $\mathcal{B}^2_{\mathsf{SKE}}$ submits (m_0, m_1) to the external challenger; and samples $\mathsf{sk}_1 \leftarrow \mathsf{SKE.KeyGen}(1^\lambda)$,; C_{PKE} receives challenge ciphertext $\mathsf{ct}_2^* \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_0, m_b)$ from the challenger, where sk_0 is only known to the challenger and C_{PKE} sends $\mathsf{ct}^{**} = (\mathsf{ct}_1^* = \mathsf{wPKE.Enc}(\mathsf{pk}, \mathsf{sk}_1), \mathsf{ct}_2^* \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_0, m_b))$ to C. C continues to simulate the encryption and decryption oracle as above to decrypt only valid ciphertexts. In the end, if C outputs b', then $\mathcal{B}^2_{\mathsf{SKE}}$ outputs the same b'.

□

F Watermarkable Functional Encryption from Watermarkable Attribute-Based Encryption

We give a high-level description on how to turn the [GKP+13] functional encryption construction from ABE, FHE, garbled circuits into a watermarkable FE based on watermarkable ABE. We will not elaborate details of the construction as we cannot possibly cover all explicit constructions for watermarking-compositions of existing schemes, and our main goal here is to give reference to another example of watermarking-composition when applied to an advanced encryption scheme.

All the main algorithms are the same as the [GKP+13] construction. We will mainly remark on how we do extraction and refer interested readers to [GKP+13] for the detailed construction.

Watermarkable Functional Encryption: Definitions A watermarkable FE scheme can be given different watermarking definitions. One is to watermark the master secret key and given out an unmarked functional key to the adversarial program $\mathcal{A}$; the other is to let $\mathcal{A}$ submit a function f in the first stage and gets to query on marked versions of the functional key sk_f. Both notions are interesting.

For the first notion, we can consider the following security game: The γ-Unremovability for a watermarkable single-key FE scheme says, for all $\lambda \in \mathbb{N}$, and for all PPT admissible stateful adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\Pr\left[\begin{array}{c}\mathsf{Extract}(\mathsf{xk},\mathsf{mpk},\mathsf{aux}=x,C)\notin\mathcal{Q}\\ \wedge C \text{ is } \gamma\text{-good}\end{array} : \begin{array}{c}(\mathsf{msk},\mathsf{mpk},\mathsf{xk},\mathsf{mk})\leftarrow\mathsf{WMSetup}(1^\lambda)\\ (x,C)\leftarrow\mathcal{A}^{\mathsf{Mark}(\mathsf{mk},\mathsf{msk},\cdot)}(1^\lambda,\mathsf{mpk})\end{array}\right]\leq\mathsf{negl}(\lambda).$$

where $\mathcal{Q}$ is the set of marks queried by $\mathcal{A}$ and $C=(C_1,C_2,D)$ is a PPT admissible, stateful γ-good adversary in the security game $G_{\mathsf{ABE}}(\mathsf{msk},\mathsf{mpk},\mathsf{aux}=(x),\cdot)$ if:

$$\Pr\left[D(\mathsf{st},\mathsf{ct}_b)=b : \begin{array}{c}(f,\mathsf{st}_1)\leftarrow C_1(\mathsf{mpk})\\ \mathsf{sk}_f\leftarrow\mathsf{FE.KeyGen}(\mathsf{msk},f)\\ \mathsf{ct}_0\leftarrow\mathsf{FE.Enc}(\mathsf{mpk},x),\mathsf{ct}_1\leftarrow\mathsf{Sim}(\mathsf{mpk},\mathsf{sk}_f,f,f(x),1^{|x|}),b\leftarrow\{0,1\}\\ \mathsf{st}\leftarrow C_2(\mathsf{st}_1,\mathsf{ct}_b)\end{array}\right]\geq\frac{1}{2}+\gamma.$$

where Sim is some PPT simulator.

A selective variant is requiring $\mathcal{A}$ output x before seeing mpk. Note that we need $\mathcal{A}$ to output x in the first stage because by relying out construction on ABE, the choice of message x will influence the choice of the ABE reduction's choice of challenge attribute, which must be committed to before entering stage 2. We can also let $\mathcal{A}$ outputs f first so that $\mathcal{A}$ sees sk_f before outputing x (which matches the semantics of a fully secure FE). But then the $\mathsf{Extract}$ algorithm will have to take in f as an auxiliary inputs.

The case where sk_f gets watermarked is slightly trickier to define. We need to work with an indistinguishability-based FE security where the function family $\mathcal{F}$ is a "high-entropy" function family. Also, the $\mathsf{Extract}$ needs to take in the challenge f as an auxiliary input. We refer to [GKM+19] Remark 4.5 for more discussions. [GKM+19]'s watermarkable predicate encryption scheme can be extended to a watermarkable FE scheme for watermarking a functional key when working with the above restricted function class.

F.1 Two-Outcome Attribute-Based Encryption

The first building block for [GKP+13] FE is a two-outcome ABE, called ABE_2. It has the similar syntax to a normal ABE except that: the encryption scheme takes in mpk,x,m_0,m_1; when decrypting with a functional key sk_f on ciphertext ct_{x,m_0,m_1}, the decryption will output m_b when $f(x)=b$.

The single-key ABE_2 security game is exactly like a single-key ABE security game except that $\mathcal{A}$ provides 3 challenge messages (m,m_0,m_1) and attribute x, then $\mathcal{A}$ is given either $\mathsf{ABE}_2.\mathsf{Enc}(\mathsf{mpk},x,(m_0,m))$ if $f(x)=1$ for the key generation query f, or $\mathsf{ABE}_2.\mathsf{Enc}(\mathsf{mpk},x,(m,m_1))$ if $f(x)=0$, and asked to output b.

A single-key ABE_2 scheme can be built with a simple construction from a regular single-key ABE . We refer the readers to [GKP+13] Appendix B for details and give a high level idea: the scheme generates two pairs regular ABE keys; the encryption generates two encryptions of the same message and attribute, each under one of the master public keys respectively; the decryption algorithm on sk_f and $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$ will try to decrypt both ciphertexts, and output that one that has a valid decryption result.

In the watermarking construction, we watermark both the master secret keys of the two ABE (regarding single-key watermarkable ABE security: see Definition C.3 except that $\mathcal{A}$ is not give $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle and C can only query $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ once)). The Extract algorithm in the watermarkable ABE_2 will turn the input program C into a circuit C_i to break the single-key ABE security for each of the keys in the construction. The reduction in [GKP$^+$13] Appendix B would go through because each stage of ABE_2 in the unremovability game would map to the stage of the regular ABE unremovability game. That is, making some msk marking queries after receiving mpk, and outputs challenge messages, attribute and program C; then C gets to query a single sk_f and challenged with the challenge ciphertext (If we choose to work with a watermarkable FE scheme where we let $\mathcal{A}$ choose f, we can also let the adversary in watermarkble ABE scheme query one sk_f and then the program C is not allowed to make any queries).

Watermarkable Functional Encryption Construction, Extraction and Security We refer the readers to [GKP$^+$13] Section 3.1 for details on the construction and its preliminaries for definitions of FHE and garbled circuit, since we only plan to give an idea here.

The construction only needs to rely on a watermarkable ABE_2 scheme (which we have shown above can be based on watermarkable regular ABE). The FHE and garbled circuits do not have to be watermarked because they are only generated freshly on each evaluation.

The WMSetup algorithm generates all λ number of $(\mathsf{mpk}_i, \mathsf{msk}_i, \mathsf{mk}_i, \mathsf{xk}_i)$ from the setup of a watermarkable ABE_2 scheme.

The $\mathsf{KeyGen}(\mathsf{msk}, f)$ algorithm generates a $sk_i \leftarrow \mathsf{ABE}_2.\mathsf{KeyGen}(\mathsf{msk}_i, \mathsf{FHE.Eval}^i_f)$ where $\mathsf{FHE.Eval}^i_f$ is outputing the i-th bit after the FHE evaluation on some FHE Eval key FHE.pk, the input function f and some FHE ciphertexts $c_1, \cdots, c_n$.

The $\mathsf{Enc}(\mathsf{mpk}, x)$ algorithm takes input attribute $x = x_1 \cdots x_n$, generates a fresh FHE key pair, encrypt each x_i to get FHE ciphertext c_i. Then produce garbled circuit for the FHE decryption algorithm $\mathsf{FHE.Dec}(\mathsf{FHE.sk}, \cdot)$ with 2λ labels $\{L_i^0, L_i^1\}_{i \in [\lambda]}$. Then we produce ABE_2 ciphertext $\mathsf{ct}_i \leftarrow \mathsf{ABE}_2.\mathsf{Enc}(\mathsf{mpk}_i, (\mathsf{FHE.pk}, \vec{c}), L_i^0, L_i^1)$ for all $i \in [\lambda]$ where $\vec{c} = (c_1, \cdots, c_n)$. Output the ciphertext $(\mathsf{ct}_1, \cdots \mathsf{ct}_\lambda)$ and the garbled circuit.

The decryption algorithm $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct})$ runs the ABE_2 decryption on eachct_i using sk_i to obtain a label $L_i^{d_i}$, for each $i \in [\lambda]$. Then one can recover $f(x) \leftarrow \mathsf{FHE.Dec}(\mathsf{FHE.sk}, d_1 \cdots d_\lambda)$ by using the labels $L_i^{d_i}$ and the garbled circuit.

Now we can talk about how we watermark and extract: the marking scheme simply marks each $\mathsf{ABE}_2.\mathsf{msk}_i$ for $i \in [\lambda]$. To see how we extract, we roughly open up the security proof of the above scheme: On an input program C, the extraction interacts with C as in the stage-2 FE game in our unremovability definition. Recall that in the end of the original security game, we need to prepare either real ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x)$ or a simulated ciphertext, and ask the final stage adversary D to distinguish. But in our extraction algorithm that tries to create circuit C_i to break the unremovability of the i-the ABE_2 scheme, we instead sample either of the following: (1) generate the ciphertext as in the original scheme; (2) use the label $L_i^{d_i}$ twice in the ciphertext $\mathsf{ct}'_i \leftarrow \mathsf{ABE}_2.\mathsf{Enc}(\mathsf{mpk}_i, (\mathsf{FHE.pk}, \vec{c}), L_i^{d_i}, L_i^{d_i})$, where $d_i = \mathsf{FHE.Eval}^i_f(\mathsf{FHE.pk}, \vec{c})$. More specifically, circuit C_i obtains one of these ciphetexts from some external ABE_2 challenger and uses C's output to distinguish them.

Why would a γ-good C in the original security game still be γ-good when we simulate the game as in the above extraction? This relies on the hybrid arguments invoking the IND-CPA security of FHE and then the security of garbled circuit. We refer the details to the proofs for Hybrid 0 to Hybrid 2 in Section 3.2 of [GKP+13]. By invoking these security properties, we move from giving a completely simulated ciphertext, to give the fake ciphertext described in the extraction algorithm (when flipping the coin to 1), while C should still be γ-good with some noticeable probability.

Then we can argue that at least one of C_i is γ_i-good for some non negligible γ_i, we can break unremovability of ABE_2 with the i-th key pair. In the marking stage, the i-th ABE_2 reduction can simulate the marking queries by making marking query to the ABE_2 marking oracle (and mark the other keys on its own). After $\mathcal{A}$ commits to chellenge message x, the reduction can also commit the challenge attribute $(\mathsf{FHE.pk}, \vec{c})$ where $\vec{c}$ is computed from x as in the construction. After $\mathcal{A}$ outputs C, the reduction creates circuit C_i as in the extraction algorithm, and C_i can answer C's single key query by making a single query to the ABE_2 $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ oracle.

G Watermarkable Non-malleable PKE from Watermarkable CPA-secure PKE, NIZK and Signatures

G.1 Watermarkable Implementation of Non-malleable CCA2 PKE

We present that an alternative construction to Appendix A, the CCA2 non-malleable PKE in [DDN91] also has a watermarking implementation. We will omit the security proof due to the similarity to Appendix A.

G.2 Construction and Security

Our construction is based on the non-malleable encryption scheme in [DDN91].

Given a watermarkable implementation of a CPA-secure PKE $\mathsf{wPKE} = (\mathsf{WMSetup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Mark}, \mathsf{Extract})$, a NIZK scheme $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ and a signature scheme $\mathsf{DS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. where the verification key size of DS is n.

Construction

$\mathsf{KeyGen}(1^\lambda, 1^n)$: For $i \in [n], b \in \{0,1\}$: compute $(\mathsf{wPKE.pk}_{i,b}, \mathsf{wPKE.sk}_{i,b}, \mathsf{xk} = \mathsf{wPKE.xk}_{i,b}; \mathsf{mk} = \mathsf{wPKE.mk}_{i,b}) \leftarrow \mathsf{wPKE.WMSetup}(\lambda)$;

Compute $\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$; Output $\mathsf{sk} = (\mathsf{wPKE.sk}_{i,b})_{i\in[n],b\in\{0,1\}}$; $\mathsf{pk} = (\{\mathsf{wPKE.pk}_{i,b}\}_{i\in[n],b\in\{0,1\}}, \mathsf{CRS})$; $\{\mathsf{wPKE.pk}_{i,b}\}_{i\in[n],b\in\{0,1\}}$; $\mathsf{mk} = \{\mathsf{wPKE.pk}_{i,b}\}_{i\in[n],b\in\{0,1\}}$

$\mathsf{Enc}(\mathsf{pk}, m)$:

- parse $\mathsf{pk} := \{\mathsf{pk}_{i,b}\}_{i\in[n],b\in\{0,1\}}, \mathsf{CRS}$; compute $(\mathsf{DS.sk}, \mathsf{DS.vk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$; Let us denote i-th bit of $\mathsf{DS.vk}$ as vk_i.
- For each $i \in [n]$: compute $\mathsf{ct}_i \leftarrow \mathsf{wPKE.Enc}(\mathsf{pk}_{i,\mathsf{vk}_i}, m)$;
- compute $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, (\mathsf{ct}_1, \mathsf{ct}_2, \cdots, \mathsf{ct}_n, \mathsf{DS.vk}), (r_1, r_2, \cdots, r_n, m))$ for the following statement:
  $\exists$ witness $(r_1, r_2, \cdots, r_n, m)$ such that $\mathsf{ct}_i = \mathsf{wPKE.Enc}(\mathsf{pk}_{i,\mathsf{vk}_i}, m; r_i)$ for all $i \in [n]$, where r_i is the randomness used in encryption.
- compute a signature $\mathsf{sig} \leftarrow \mathsf{DS.Sign}(\mathsf{DS.sk}, (\{\mathsf{ct}_i\}_{i\in n}, \pi))$

- Output $\mathsf{ct} = (\{\mathsf{ct}_i\}_{i\in n}, \mathsf{DS.vk}, \pi, \mathsf{sig})$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$:

- parse $\mathsf{ct} = (\{\mathsf{ct}_i\}_{i\in n}, \mathsf{DS.vk}, \pi)$; $\mathsf{sk} = \{\mathsf{sk}_{i,b}\}_{i\in[n], b\in\{0,1\}}$;
- if $\mathsf{DS.Verify}(\mathsf{DS.vk}, (\{\mathsf{ct}_i\}_{i\in n}, \pi), \mathsf{sig}) = 1$ continue; else abort and output $\perp$.
- if $\mathsf{NIZK.Verify}(\mathsf{CRS}, \pi, ((\{\mathsf{ct}_i\}_{i\in n}, \mathsf{DS.vk})) = 1$, continue; else abort and output $\perp$.
- compute $m \leftarrow \mathsf{wPKE.Dec}(\mathsf{sk}_{1,\mathsf{vk}_1}, \mathsf{ct}_1)$; output m'.

$\mathsf{Mark}(\mathsf{mk}, \mathsf{sk}, \tau)$: parse $\mathsf{mk} = \{\mathsf{mk}_{i,b}\}_{i\in[n], b\in\{0,1\}}$; $\mathsf{sk} = \{\mathsf{sk}_{i,b}\}_{i\in[n], b\in\{0,1\}}$; output $\{\mathsf{sk}_{i,b,\tau} \leftarrow \mathsf{wPKE}_{i,b}.\mathsf{Mark}(\mathsf{sk}_{i,b}, \tau)\}\}_{i\in}$
$\mathsf{Extract}(\mathsf{xk}, \mathsf{pk}, \mathsf{aux} = (m_0, m_1), C)$:

- parse $\mathsf{xk} := (\{\mathsf{xk}_{i,b}\}_{i\in[n], b\in\{0,1\}}, \mathsf{NIZK.td})$; $\mathsf{pk} := (\{\mathsf{pk}_{i,b}\}_{i\in[n], b\in\{0,1\}}, \mathsf{CRS})$. Initialize an empty tuple $\vec{\tau}$;
  For $i^* = 1, \cdots, n; b^* = 0, 1$: Create the following circuit C_{i^*,b^*} with black-box access to C:
  - C_{i^*,b^*} is hardcoded with $(\{\mathsf{pk}_{i,b}\}_{i\in[n], b\in\{0,1\}}, \{\mathsf{xk}_{i,b}\}i \neq i^* \vee b \neq b^*, \mathsf{td}, \mathsf{CRS})$
    C_{i^*,b^*} acts as a stage-2 reduction from CCA2-PKE to CPA-PKE with the keys $(\mathsf{sk}_{i^*,b^*}, \mathsf{pk}_{i^*,b^*})$ simulates the CCA2-PKE stage-2 game for C as follows:
  - For C's decryption queries $\mathsf{ct} = (\{\mathsf{ct}_{i,\mathsf{vk}_i}\}, \mathsf{vk}, \pi, \mathsf{sig})$:
    * First check if $\mathsf{NIZK.Verify}(\mathsf{CRS}, \pi) = 1$, if 0 output $\perp$; else continue;
    * By the extraction key simulation property, since $\mathsf{xk}_{i\neq i^* \vee b\neq b^*}$ can be used to simulate the oracles used in $G_{CPA}(\mathsf{pk}_i, \mathsf{sk}_j, \cdot)$ for $i \neq i^*$ or $b \neq b^*$, C_{i^*,b^*} can simulate the oracle $\mathsf{wPKE.Dec}(\mathsf{sk}_{i^*,1-b^*}, \cdot)$ and thus decrypt $\mathsf{ct}_{i^*,1-b^*}$ in the ciphertext ct to output m.
  - C_{i^*,b^*} submits (m_0, m_1) to the external challenger; C_{i^*,b^*} receives challenge ciphertext $\mathsf{ct}^*_{i^*,b^*} = \mathsf{wPKE.Enc}(\mathsf{pk}_{i^*,b^*}, m_\delta), \delta \leftarrow \{0,1\}$ from the challenger.
  - C_{i^*,b^*} prepares the following ciphertext:
    1. Sample $(\mathsf{DS.vk}^*, \mathsf{DS.sk}^*) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$ so that the i^*-th bit of vk^* is b^*.
    2. Compute $\mathsf{ct}^*_{i,b_i} \leftarrow \mathsf{wPKE.Enc}(\mathsf{pk}_{i,b_i}, m_{\delta'}), \delta' \leftarrow \{0,1\}$ for all $i \neq i^*$, where $b_i = \mathsf{vk}^*_i$.
    3. compute $\widehat{\pi} \leftarrow \mathsf{Sim}(\mathsf{td}, \mathsf{CRS},, (\{\mathsf{ct}^*_{i,b_i}\}_{i\in[n], b_i = \mathsf{vk}^*_i}))$
       where Sim is the simulator algorithm for NIZK.

    Then it sends $\mathsf{ct}^* = (\{\mathsf{ct}^*_{i,b_i}\}_{i\in[n], b_i=\mathsf{vk}^*_i}), \mathsf{vk}^*, \widehat{\pi})$ to C.
  - C continues to simulate the decryption oracle as above to decrypt only valid ciphertexts $\mathsf{ct} \neq \mathsf{ct}^*$, except adding the following check:
    * Check if $\mathsf{vk} = \mathsf{vk}^*$, if yes, output $\perp$; else continue to decrypt.
  - C outputs ℓ ciphertexts $\mathsf{ct}_1, \cdots, \mathsf{ct}_\ell$ and C_i computes $d_j \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_j)$ if $ct_j \neq \mathsf{ct}^*$, else $d_j = \perp$.
  - In the end, feed $(d_1, \cdots, d_\ell)$ to C and C outputs a bit b'. C_i outputs the same as C outputs.
  - Add $\tau/\perp \leftarrow \mathsf{wPKE.Extract}(\mathsf{wPKE.xk}_{i^*,b^*}, \mathsf{wPKE.pk}_{i^*,b^*}, \mathsf{aux} = (m_0, m_1), C_{i^*,b^*})$ to $\vec{\tau}$.
- Output $\vec{\tau}$.

The security proof is highly similar to Appendix A and we omit it here.