Collusion Resistant Watermarking Schemes for Cryptographic Functionalities

Rupeng Yang[1,2], Man Ho Au[2⋆], Junzuo Lai[3⋆], Qiuliang Xu[4⋆], and Zuoxia Yu[2]

[1] School of Computer Science and Technology, Shandong University,
Jinan, 250101, China
orbbyrp@gmail.com

[2] Department of Computing, The Hong Kong Polytechnic University,
Hung Hom, Hong Kong, China
csallen@comp.polyu.edu.hk, zuoxia.yu@gmail.com

[3] College of Information Science and Technology, Jinan University,
Guangzhou, 510632, China
laijunzuo@gmail.com

[4] School of Software, Shandong University,
Jinan, 250101, China
xql@sdu.edu.cn

Abstract. A cryptographic watermarking scheme embeds a message into a program while preserving its functionality. Recently, a number of watermarking schemes have been proposed, which are proven secure in the sense that given *one* marked program, any attempt to remove the embedded message will substantially change its functionality.

In this paper, we formally initiate the study of collusion attacks for watermarking schemes, where the attacker's goal is to remove the embedded messages given *multiple* copies of the same program, each with a different embedded message. This is motivated by practical scenarios, where a program may be marked multiple times with different messages.

The results of this work are twofold. First, we examine existing cryptographic watermarking schemes and observe that all of them are vulnerable to collusion attacks. Second, we construct collusion resistant watermarking schemes for various cryptographic functionalities (e.g., pseudorandom function evaluation, decryption, etc.). To achieve our second result, we present a new primitive called puncturable functional encryption scheme, which may be of independent interest.

Keywords: Watermarking, Watermarkable PRF, Collusion Resistance, Public Extraction

⋆ Corresponding author.

1 Introduction

A watermarking scheme allows one to embed some information into a program[1] without significantly changing its functionality. It has many natural applications, including ownership protection, information leaker tracing, etc.

The formal definition of watermarking schemes for programs is first presented by Barak et al. in [BGI$^+$01]. Subsequently, new properties of watermarking schemes are presented in [HMW07, NW15, CHV15]. They are briefly summarized below.

- **Unremovability:** This is the essential security property for watermarking schemes, which requires that it should be hard to *remove* or *modify* the embedded information in a marked program without destroying it.
- **Public Extraction:** Anyone should be able to extract the embedded information in a marked program. In other words, the extraction key will be made public.
- **Public Marking:** Anyone should be able to embed information into a program. In other words, the marking key will be made public.
- **Unforgeability:** Only the authorized entity who holds the marking key should be able to embed information into a program. Obviously, it requires keeping the marking key secret and is not compatible with the "public marking" property.
- **Message-Embedding:** This property allows one to embed a given string (instead of merely a mark symbol) into the watermarked program.

Despite being a natural concept and perceived to have a wide range of applications, watermarking schemes provably secure against arbitrary removal strategies were not presented until 2015. In [CHN$^+$16] (which is a merged version of [NW15] and [CHV15]), Cohen et al. construct a publicly extractable watermarking scheme for the evaluation algorithm of pseudorandom functions (PRFs) from indistinguishability obfuscators. Based on the watermarkable PRF families, they also construct watermarkable public key encryption (PKE) schemes and watermarkable signature schemes. However, Cohen et al.'s schemes do not achieve standard unforgeability. Subsequently, Yang et al. [YAL$^+$18] improve the watermarkable PRF in [CHN$^+$16] to achieve both standard unforgeability and public extraction simultaneously.

In another line of research, initiated by Boneh et al. in [BLW17], watermarkable PRFs are constructed from variants of constraint-hiding constrained PRFs (e.g., privately programmable PRF and translucent puncturable PRF). Boneh et al.'s scheme is constructed from privately programmable PRF, which is instantiated from indistinguishability obfuscator in [BLW17]. Subsequently, based on a translucent puncturable PRF, Kim and Wu [KW17] present the first construction of watermarkable PRF from standard lattice assumptions. Then,

[1] In this paper, we focus on watermarking schemes for programs and only consider those with provable security against arbitrary removal strategies. We refer readers to Sec. 1.2 for an extended introduction to the area.

Peikert and Shiehian [PS18] construct privately programmable PRF from LWE, which provides another way to instantiate watermarkable PRF from standard assumptions. Recently, in [QWZ18] and [KW19], watermarkable PRFs with public marking are constructed from constraint-hiding constrained PRF and puncturable extractable PRF respectively, both of which can be instantiated from standard lattice assumptions.

Besides, a very simple yet elegant construction of watermarking scheme for any PKE scheme is presented by Baldimtsi et al. [BKS17]. However, their scheme does not support multi-message-embedding inherently. That is, each program can only be marked with at most one message during the life-time of the scheme.

Collusion Resistance of Watermarking schemes. In practical applications, it is usually required that unremovability of watermarking schemes should hold under "collusion attacks", where the attacker can access several copies of the same program embedded with different information. As a concrete example, consider the following scenario. A software development company developed a program and would like to outsource its testing to several organizations. To prevent these organizations from leaking the program, the company will employ a watermarking scheme to embed the name of the target organization into the copy being sent. Here, the watermarking scheme should enable the company to trace program leakers even when a few target organizations collude.

However, for all previous watermarking schemes [CHN$^+$16, BLW17, KW17, BKS17, PS18, YAL$^+$18, QWZ18, KW19], the unremovability is only proved against an adversary who attempts to remove or modify the embedded message given a *single* marked program[2], and it is unknown whether they are secure against collusion attacks. Thus, the following question arises naturally:

Can we build collusion resistant watermarking scheme?

1.1 Our Results

In this paper, we explore the existence of watermarking schemes secure against collusion attacks. First, we observe that unfortunately, all existing watermarking schemes are *vulnerable* to collusion attacks (we elaborate this in Sec. 2). Then, we consider how to develop watermarkable cryptographic primitives secure against the collusion attacks. Specifically, our contributions are as follows.

- We present the notion of *collusion resistant watermarking scheme* to capture collusion attacks. It requires a stronger unremovability (namely, collusion resistant unremovability) that allows the adversary to obtain watermarked circuits embedded with different messages for the same functionality.

[2] In a concurrent work [GKM$^+$19], collusion resistant watermarking schemes for public-key cryptographic primitives are presented. However, their constructions are under a relaxed notion of functionality-preserving. In this work, we achieve collusion resistance while preserving the original "statistical functionality-preserving" proposed in [CHN$^+$16].

- We give a construction of *collusion resistant watermarkable PRF*, which is the first watermarkable cryptographic primitive provably secure against the collusion attacks. To achieve this, we introduce a new message-embedding technique in the watermarking setting and propose a new primitive, namely, *puncturable functional encryption scheme*, which we believe will find additional applications in constructing advanced cryptographic primitives.
- Based on our construction of collusion resistant watermarkable PRF, we also construct watermarkable PKE schemes and watermarkable signature schemes, both of which have collusion resistant unremovability.

We compare the main features achieved by current watermarking schemes and our watermarking schemes in Table 1. We remark that in addition to collusion resistance, our schemes can achieve many desirable features, including public extraction, unforgeability, and message-embedding.

Table 1: The Comparison.

		Unforgeability	Public Extraction	Public Marking	Message Embedding	Collusion Resistance
[CHN+16]	PRF	✗	✓	✗	✓	✗
	PKE	✗	✓	✗	✓	✗
	Signature	✗	✓	✗	✓	✗
[YAL+18]	PRF	✓	✓	✗	✓	✗
[BLW17]	PRF	✓	✗	✗	✓	✗
[KW17]	PRF	✓	✗	✗	✓	✗
[QWZ18]	PRF	✗	✗	✓	✓	✗
[KW19]	PRF	✓	✗	✗	✓	✗
	PRF	✓†	✗	✓	✓	✗
[BKS17]	PKE	✓	✗	✗	✗	-
	PKE	✓	✓	✗	✗	-
This work	PRF	✓	✓	✗	✓	✓
	PKE	✓	✓	✗	✓	✓
	Signature	✓	✓	✗	✓	✓

†: Weaker versions of unforgeability are achieved for this scheme.

The presented collusion resistant watermarking schemes are built on several cryptographic primitives, which can be constructed from indistinguishability obfuscator and standard lattice assumptions.

Theorem 1.1 (Informal). *Assuming the worst-case hardness of appropriately parameterized GapSVP and SIVP problems and the existence of indistinguishability obfuscator, there exist collusion resistant watermarkable PRF/PKE/signature schemes.*

Remark 1.1. It is worth noting that our constructions of collusion resistant watermarking schemes rely on the existence of indistinguishability obfuscator. However, this seems essential, at least for collusion resistant watermarkable PRF. To see this, recall that as proved in [BGI$^+$01], watermarking scheme perfectly preserving the functionality of the watermarked program does not exist. Thus, a marked key of PRF must evaluate differently with the original key on some inputs, i.e., the marked key can be viewed as a constrained key of the original key. Besides, the marked key should hide its constrained inputs, since otherwise, the attacker is likely to remove the embedded messages via resetting outputs on constrained inputs. Therefore, we can approximately view a collusion resistant watermarkable PRF as a collusion resistant constraint-hiding constrained PRF, which, as shown in [CC17], can imply indistinguishability obfuscator. Nonetheless, we are not able to formalize this intuition. It is an interesting open problem to give a formal construction of indistinguishability obfuscator from collusion resistant watermarkable PRF.

1.2 Related Works

Additional Related Works on Watermarking Schemes. In this paper, we concentrate on watermarking schemes provably secure against arbitrary removal strategies. There are also numerous works (see [CMB$^+$07] and references therein) attempting to use ad hoc techniques to watermark a wide class of digital objects, such as images, audios, videos, etc. However, these constructions lack rigorous security analysis and are (potentially) vulnerable to some attacks.

In another line of research [NSS99, YF11, Nis13], watermarking schemes for cryptographic objects (e.g., the key, the signature, etc.) are constructed and rigorously analyzed. However, their security definition considers a restricted adversary that will not change the format of the watermarked objects.

Puncturable Symmetric Key Functional Encryption. One byproduct of this work is a new primitive called puncturable functional encryption. A similar primitive, which is called puncturable symmetric key functional encryption, is also studied in previous works [BV15, KNT18]. In particular, it is used to construct the indistinguishability obfuscator in these works.

We stress that these two types of primitives are incomparable. First, while succinctness is the key property for a puncturable symmetric key functional encryption scheme, it is not required in our puncturable functional encryption scheme. Thus, our scheme cannot be used in constructions of indistinguishability obfuscators. On the other hand, our puncturable functional encryption scheme will puncture a secret key on a ciphertext, but in a puncturable symmetric key functional encryption scheme, secret keys are punctured on a message or on a tag. Thus, their schemes are also inapplicable to our setting.

Traitor Tracing Scheme. The notion of collusion resistant watermarking scheme is somewhat similar to the notion of traitor tracing scheme, which aims at tracing secret key leakers among a set of users holding functionally equivalent secret keys in a broadcast encryption setting. Since first presented in [CFN94], traitor tracing has been formally studied for a long time (see e.g., [BSW06, BN08, BZ14, NWZ16, GKW18, CVW$^+$18] and references therein for an overview of previous works).

Generally, in a traitor tracing scheme, there is a common public key pk and each user holds a different secret key. Data encrypted under the common public key can be decrypted by all users in the system. Moreover, there exists a tracing algorithm, which outputs a set of users on input a "pirate decoder" that can decrypt ciphertexts under pk. It is guaranteed that the tracing algorithm can identify at least one of the users in the coalition that produces the pirate decoder.

Comparing watermarking and traitor tracing. Both (collusion resistant) watermarking and traitor tracing will issue copies of a program (or a key), which are embedded with some information, to users and aim at recovering the embedded information from a functionally-similar program/key generated by them. However, solutions to the traitor tracing problem do not yield watermarking schemes directly, since these two notions also have several inherent differences.

First, in a traitor tracing scheme, secret keys of users are issued by a center, while in a watermarking scheme, user can choose their watermarked programs themselves. Another difference is that in a traitor tracing scheme, secret keys of all users are functionally equivalent, while in a watermarking scheme, programs with different functionalities can be watermarked in the same watermarking scheme. Besides, traitor tracing schemes focus on tracing secret key leakers in an encryption scheme, while watermarking schemes aim at marking general purpose programs (although we only know how to watermark some specific cryptographic functionalities currently).

A closer look at how to construct traitor tracing schemes. In [BSW06], Boneh et al. present a classic paradigm to construct traitor tracing schemes, which is also used or adapted in many subsequent works [BZ14, NWZ16, GKW18]. The construction proceeds in two steps.

First, a private linear broadcast encryption (PLBE) scheme is constructed. Recall that a PLBE scheme has a sequence $(sk_1, \ldots, sk_N)$ of N secret keys for a public key and each ciphertext is labeled with an integer in $[0, N]$. A secret key sk_i is only able to decrypt a ciphertext with label j when $j < i$. Thus, a ciphertext with label 0 can be decrypted by all secret keys, while a ciphertext with label N can not be decrypted by any secret key. Also, it is required that it is computationally infeasible to distinguish a ciphertext with label j and that with label $j - 1$ if sk_j is not given.

A PLBE scheme implies a traitor tracing scheme [BSW06, GKW18]. More concretely, the traitor tracing scheme supports a user set of size N and the ith user in that set is given secret key sk_i. Broadcast messages will be encrypted with label 0. When tracing colluders from a pirate decoder, the tracing algorithm feeds the decoder with ciphertexts labeled with 0 to N sequentially and outputs i if there exists a "large gap" in decryption success probability between ciphertexts labeled with $i-1$ and those labeled with i. Note that the decoder can decrypt with a high success probability on ciphertext labeled with 0 (due to the usefulness of the decoder) and can decrypt with a negligible success probability on ciphertext labeled with N (due to the security of PLBE), thus, there must exists a large gap in decryption success probability between $i-1$ and i for some $i \in [N]$. Also, as no one could distinguish ciphertexts labeled with $i-1$ and that labeled with i without sk_i, the large gap must occur between $i-1$ and i such that the colluders possess sk_i. Therefore, the tracing algorithm can recover at least one of the colluders.

1.3 Roadmap

The rest of the paper is organized as follows. We give an overview of our construction in Sec. 2. Then in Sec. 3, we review notations used in this work. We present the formal definition of collusion resistant watermarkable PRF in Sec. 4. Then in Sec. 5, we define and construct puncturable functional encryption, which is employed to construct collusion resistant watermarkable PRF. We show our main construction of collusion resistant watermarkable PRF in Sec. 6 and present constructions of collusion resistant watermarking schemes for public key primitives in Sec. 7. Finally, in Sec. 8, we conclude our work with a few possible future works.

2 Technical Overview

In this section, we provide an overview of our construction of collusion resistant watermarkable PRF. Our starting point is the watermarking scheme WM_0 presented in [CHN+16] (or more accurately, its variant in [YAL+18]). We first explain why WM_0 (and all previous watermarking schemes) are not collusion resistant and describe the main challenges in achieving collusion resistance. Then we give a high-level idea on how to address these challenges.

A brief overview of WM_0***.*** Roughly speaking, WM_0 works as follows. The extraction key/marking key pair of WM_0 is a public key/secret key pair (pk, sk) of a PKE scheme. To embed a message msg into a PRF key k, the marking algorithm outputs an obfuscation of the following circuit, which evaluates the function $\mathsf{PRF}(k, \cdot)$ correctly at all points, except for some "punctured points".

$$\mathtt{C}(x) = \begin{cases} f(\mu) \oplus msg & \text{if } \mu = \mathtt{Dec}(sk, x) \in \mathcal{V} \\ \mathsf{PRF}(k, x) & \text{otherwise.} \end{cases}$$

Here, $\mathtt{Dec}$ is the decryption algorithm of the underlying PKE scheme, $\mathcal{V}$ is a set defined by the PRF key k and f is a suitable function.

When extracting the embedded message from a watermarked circuit, the extraction algorithm first samples a string $\mu \in \mathcal{V}$ and encrypts it with the public key pk. Next, it evaluates the circuit on the ciphertext and obtains an output z. Finally, it computes $msg = z \oplus f(\mu)$. The above extraction procedure will be repeated multiple times and the extraction algorithm will output the majority result or an "UNMARKED" symbol if no majority is found.

Security of the scheme relies on the fact that punctured points (i.e. those decrypted to a string in $\mathcal{V}$) are hidden[3]. As a result, the adversary, who is only allowed to alter the marked circuit slightly, is not able to change the output values on a large enough fraction of punctured points, and thus the extraction algorithm can still extract the correct message.

Why WM_0 ***is not collusion resistant?*** However, if watermarked circuits embedded with different messages for the same PRF key k are given, one can easily locate all punctured points via comparing the outputs of the circuits. In addition, it is easy to modify or remove the embedded messages via resetting outputs on all punctured points.

In more detail, given two circuits $\mathtt{C}_1$ and $\mathtt{C}_2$ that are generated by embedding different messages, say msg_1 and msg_2, into the same PRF key k, an attacker can output a circuit $\mathtt{C}^*$ embedded with a new message msg^* as follow:

$$\mathtt{C}^*(x) = \begin{cases} \mathtt{C}_1(x) \oplus msg_1 \oplus msg^* & \text{if } \mathtt{C}_1(x) \neq \mathtt{C}_2(x). \\ \mathtt{C}_1(x) & \text{otherwise.} \end{cases}$$

It is not hard to see that $\mathtt{C}^*$ will compute the PRF with key k correctly on almost all inputs except that it will output $f(\mu) \oplus msg^*$ on an input whose decryption μ is in $\mathcal{V}$. Therefore, the attacker can compromise the unremovability of WM_0 via a collusion attack[4].

Since nearly all[5] previous watermarking schemes are constructed following the blueprint proposed in [CHN+16], we can use a similar strategy to show that they are not collusion resistant. We stress that all collusion attacks are based on the fragility of the way messages are embedded and do not take advantage of the concrete instantiations of the schemes.

The challenge in achieving collusion resistance. To better explain why WM_0 is not able to achieve collusion resistance, we describe WM_0 in a modular manner.

In a high level, on input a PRF key k and a message msg, the marking algorithm of WM_0 works as follows:

[3] One could find some punctured points via generating them from public information, but cannot distinguish a random punctured point from a random point in the input space.

[4] We remark that this will not affect the claimed security of WM_0. The attacks only show that WM_0 is not applicable in scenarios where collusion attacks are a legit threat.

[5] The watermarking scheme proposed in [BKS17] is constructed in a different approach, however, it cannot embed different messages into the same program.

1. Generates two sequences $\mathcal{X} = (x_1, \dots, x_l)$ and $\mathcal{Y} = (y_1, \dots, y_l)$, where x_i and y_i are in the input space and the output space of the watermarked PRF respectively. More concretely, in WM_0, for each pair (x_i, y_i), $x_i = \texttt{Enc}(pk, \mu)$ and $y_i = f(\mu)$ for some $\mu \in \mathcal{V}$.
2. Encodes the message msg into a sequence $\mathcal{Z} = (z_1, \dots, z_l) = \texttt{encode}(\mathcal{X}, \mathcal{Y}, msg)$, where z_i is also in the output space of the watermarked PRF. In more detail, messages are encoded into $\mathcal{Z}$ via a simple "exclusive or" operation in WM_0, i.e., $z_i = y_i \oplus msg$ for $i \in [1, l]$.
3. Outputs a circuit that computes the PRF with k correctly on inputs outside $\mathcal{X}$ and outputs z_i on input x_i (here, x_i is called a punctured point).

Correspondingly, we can abstract the extraction algorithm of WM_0, which takes as input a watermarked circuit $\mathtt{C}$, as follows:

1. Samples a set of pairs $\{x_i, y_i\}$ in $\mathcal{X} \times \mathcal{Y}$.
2. Evaluates $z_i = \mathtt{C}(x_i)$ for each x_i.
3. Recovers the message $msg = \texttt{decode}(\{x_i, y_i, z_i\})$. Here, the decoding algorithm outputs the majority of $y_i \oplus z_i$.

The key observation underlying our collusion attack is that the simple "xor" encoding scheme is fragile in the collusion setting. First, for two different messages msg and msg', let $(z_1, \dots, z_l) = \texttt{encode}(\mathcal{X}, \mathcal{Y}, msg)$ and $(z'_1, \dots, z'_l) = \texttt{encode}(\mathcal{X}, \mathcal{Y}, msg')$, then we have $z_i \neq z'_i$ for $i \in [1, l]$. This makes it easy to locate all punctured points in $\mathcal{X}$ by comparing outputs of circuits embedded with different messages. In addition, it is easy to overwrite the encoded message in a codeword $\mathcal{Z} = (z_1, \dots, z_l)$. For example, one can reset $z_i = z_i \oplus \Delta$ for $i \in [1, l]$ to xor the encoded message with Δ.

In [KW17, QWZ18, KW19], different message encoding schemes are applied. However, all of them inherit the aforementioned weakness to some extent, and thus are not robust against collusion attacks.

To solve this problem, we need to develop a robust message encoding scheme, where `decode` can recover the original embedded messages even if a collusion attacker can locate some punctured points[6] and will reset outputs on its located punctured points. Next, we explore how to develop a robust message encoding scheme and integrate it with the other part of WM_0.

Addressing the challenge: a robust message encoding scheme. We design our encoding scheme via using ideas from the realm of traitor tracing. In particular, our scheme is inspired by the well-known framework presented in [BSW06] (we recall this framework in Sec. 1.2).

The message space of our encoding scheme is $[1, N]$[7]. The input of the encoding algorithm is two sequences $\mathcal{X} = (x_1, \dots, x_l), \mathcal{Y} = (y_1, \dots, y_l)$ and a message

[6] This seems unavoidable since circuits embedded with different messages should be run differently on some points to enable message extraction.

[7] Here, we assume that N is polynomial in the security parameter and will show how to remove this restriction later.

$msg \in [1, N]$. Here, we divide the whole sequence $\mathcal{X}$ into N parts, namely, $\mathcal{X}_1$, $\ldots, \mathcal{X}_N$, each of which is labeled with an index in $[1, N]$ (we elaborate how to define $\mathcal{X}_i$ later). To encode a message msg, the encoding algorithm sets $z_i = y_i$ if $x_i \in \mathcal{X}_1 \cup \mathcal{X}_2 \cup \ldots \mathcal{X}_{msg}$ and sets z_i to be the correct PRF output otherwise. The output of the encoding algorithm is the sequence $(z_1, \ldots, z_l)$.

We also modify the decoding algorithm. It takes as input a set of tuples (x_i, y_i, z_i), where (x_i, y_i) is sampled from $\mathcal{X} \times \mathcal{Y}$ and z_i is the output of the tested circuit on input x_i, and works as follows:

1. Set $p_0 = 1$ and $p_{N+1} = 0$.
2. For $ind \in [1, N]$, estimate the fraction p_{ind} of "correctly reprogrammed" points in set $\mathcal{X}_{ind}$, where a point x_i is "correctly reprogrammed" if $y_i = z_i$. This can be accomplished via testing polynomially-many points in $\mathcal{X}_{ind}$.
3. If there exists $ind \in [0, N]$ such that $|p_{ind} - p_{ind+1}|$ is noticeable (i.e., a "large gap" at ind is found), output the message $msg = ind$. Here, $msg = 0$ denotes the code is not decodable (i.e., the circuit is unmarked).

Next, we argue why our new message encoding scheme is robust under collusion attacks. Observe that, given a few (say 2) circuits $\mathtt{C}_1$ and $\mathtt{C}_2$ embedded with messages msg_1 and msg_2 respectively (w.l.o.g. assuming $msg_1 < msg_2$), the attacker can locate punctured points in $\mathcal{X}_{ind}$ for $ind \in (msg_1, msg_2]$ by comparing outputs of $\mathtt{C}_1$ and $\mathtt{C}_2$. However, we note that

- If the attacker cannot distinguish punctured points in $\mathcal{X}_{ind_1}$ and $\mathcal{X}_{ind_2}$ for $ind_1, ind_2 \in (msg_1, msg_2]$, it cannot make $|p_{ind_1} - p_{ind_2}|$ noticeable via resetting outputs on located punctured points.
- If the attacker cannot distinguish a punctured point $x_i \in \mathcal{X}_{ind}$ from a random point for $ind \notin (msg_1, msg_2]$, it will not be able to reset the output on such x_i. Thus, we have $p_{ind} = 1$ for $ind \in [1, msg_1]$ and $p_{ind} = 0$ for $ind \in (msg_2, N]$.

Consequently, if the aforementioned indistinguishability properties are guaranteed, the large gap(s) must occur at either msg_1 or msg_2 (or at both points), i.e., the decoding algorithm could output the embedded message(s).

One problem of the above solution is that the message space is restricted to be a polynomial-size set. This is because the decoding algorithm needs to scan all indices linearly to find a large gap. Addressing this problem, we employ the refined binary search presented in [BCP14, NWZ16] to search the "large gap". The search algorithm can find all (polynomially-many) large gap points from an *exponentially large* interval in a pre-defined polynomial time, as long as $|p_{ind_1} - p_{ind_2}|$ is negligible for all (adaptively chosen) interval $[ind_1, ind_2) \subseteq [0, N + 1]$ not containing a large gap point. In this way, we can set the message space to be $[1, N]$ for an exponentially large N.

Towards integrating our new encoding scheme with WM_0***.*** Next, we integrate our encoding scheme with the remaining part of WM_0. First, we will specify how to label punctured points with indices. Then, we will show how to achieve indistinguishability properties required by our robust message encoding scheme. More precisely, we will argue that for a collusion attacker, who can locate some punctured points via comparing outputs of watermarked circuits embedded with different messages, both the unlocated punctured points and labels of the located punctured points are hidden.

Labeling punctured points with indices. Recall that in WM_0, the domain of the PRF is the ciphertext space of a PKE scheme and punctured points are encryptions of plaintexts in a set $\mathcal{V}$. To label a punctured point with an index *ind*, we append *ind* to the underlying plaintext, i.e., we define $\mathcal{X}_{ind} = \{\mathtt{Enc}(pk, \mu \| ind)\}_{\mu \in \mathcal{V}}$, where pk is the public key of the underlying PKE scheme and serves as WM_0's extraction key.

Hiding punctured points and labels. Next, we explore how to hide unlocated punctured points and labels of located punctured points. For simplicity, we consider an adversary who gets two watermarked circuits $\mathtt{C}_1$ and $\mathtt{C}_2$ for the same PRF key k, which are embedded with messages msg_1 and msg_2 respectively, where $msg_1 < msg_2$. Recall that our message encoding scheme is able to recover the embedded messages if the following two properties are guaranteed:

- Pseudorandomness of punctured points in $\mathcal{X}_{ind}$ for an adaptively chosen $ind \notin (msg_1, msg_2]$.
- Indistinguishability between punctured points in $\mathcal{X}_{ind_1}$ and $\mathcal{X}_{ind_2}$ for adaptively chosen $ind_1, ind_2 \in (msg_1, msg_2]$.

Unfortunately, the PKE scheme employed in WM_0, which is a puncturable encryption scheme, does not provide the desired properties. To see this, consider an input x from $\mathcal{X}_{ind}$, where $ind \in (msg_1, msg_2]$. Since $\mathtt{C}_1(x) \neq \mathtt{C}_2(x)$, a secret key that can decrypt x must be included in both $\mathtt{C}_1$ and $\mathtt{C}_2$ (otherwise, the circuit cannot recognize it and deal with it properly). However, the puncturable encryption scheme cannot guarantee indistinguishability on ciphertexts that are decryptable.

To bridge the gap, we present a new cryptographic primitive that we call *puncturable functional encryption* and replace puncturable encryption used in WM_0 with it. Roughly speaking, a puncturable functional encryption scheme enhances a functional encryption scheme with the puncturing capability and enjoys both security of functional encryption schemes and that of puncturable encryption schemes. Especially, similar to a functional encryption, it has the "*adaptive indistinguishability*" property, which could ensure indistinguishability of ciphertexts as long as no secret key distinguishing them is provided. Also, similar to a puncturable encryption, it has the "*ciphertext pseudorandomness*" property, which could ensure pseudorandomness of a ciphertext given a secret key punctured on it.

Now, for two punctured points x_1 and x_2 in $\mathcal{X}_{ind_1}$ and $\mathcal{X}_{ind_2}$ respectively, where $ind_1, ind_2 \in (msg_1, msg_2]$, since none of them will be reprogrammed in $\mathtt{C}_1$ while both of them will be reprogrammed in $\mathtt{C}_2$, secret keys hardwired in $\mathtt{C}_1$ and $\mathtt{C}_2$ do not need to distinguish them. Thus, their indistinguishability comes from the adaptive indistinguishability of the puncturable functional encryption scheme directly. Meanwhile, for a punctured points x in $\mathcal{X}_{ind}$ for $ind \notin (msg_1, msg_2]$, since $\mathtt{C}_1(x) = \mathtt{C}_2(x)$, we can regard $\mathtt{C}_1$ and $\mathtt{C}_2$ as the same circuit when considering pseudorandomness of x. Thus, the pseudorandomness of x can be reduced to the ciphertext pseudorandomness of the puncturable functional encryption scheme, just as what has been argued in the original security proof (in the non-collusion setting) for WM_0.

Constructing puncturable functional encryption. To construct a puncturable functional encryption scheme, we employ a functional encryption scheme, a puncturable encryption scheme, and a statistical sound non-interactive zero-knowledge (NIZK) proof. We integrate them via a "two-layer encryption" approach.

More precisely, a plaintext is first encrypted into an inner ciphertext using the functional encryption scheme. Then the NIZK proof is employed to prove that the inner ciphertext is properly encrypted. Finally, both the inner ciphertext and the proof is encrypted into an outer ciphertext under the puncturable encryption scheme. When decrypting a ciphertext, the decryption algorithm first decrypts the outer ciphertext. It aborts if the proof is invalid and outputs the decryption of the inner ciphertext otherwise. Main security properties of the constructed puncturable functional encryption (namely, adaptively indistinguishability and ciphertext pseudorandomness) reduce to corresponding security properties of underlying functional encryption and puncturable encryption respectively.

3 Notations

Let a be a string, we use $\|a\|$ to denote the length of a. Let $\mathcal{S}$ be a finite set, we use $\|\mathcal{S}\|$ to denote the size of $\mathcal{S}$, and use $s \overset{\$}{\leftarrow} \mathcal{S}$ to denote sampling an element s uniformly from set $\mathcal{S}$. For a string a and a set $\mathcal{S}$ of strings, we use $a\|\mathcal{S}$ to denote the set $\{x : \exists s \in \mathcal{S}, x = a\|s\}$. For n elements $e_1, \ldots, e_n$, we use $\{e_1, \ldots, e_n\}$ to denote a set containing these elements and use $(e_1, \ldots, e_n)$ to denote an ordered list of these elements. We write $negl(\cdot)$ to denote a negligible function, and write $poly(\cdot)$ to denote a polynomial. For integers $a \le b$, we write $[a, b]$ to denote all integers from a to b. For two circuits $\mathtt{C}_1$ and $\mathtt{C}_2$, we write $\mathtt{C}_1 \equiv \mathtt{C}_2$ to denote that for any input x, $\mathtt{C}_1(x) = \mathtt{C}_2(x)$. Following the syntax in [BLW17], for a circuit family C indexed by a few, say m, constants, we write $C[c_1, \ldots, c_m]$ to denote a circuit with constants $c_1, \ldots, c_m$.

Chernoff Bound. We make use of the Chernoff bound in our security proof. There are various forms of the Chernoff bound, here we use the one from [Goe15].

Lemma 3.1 (Chernoff Bounds). *Let $X = \sum_{i=1}^{n} X_i$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_i are independent. Let $\mu = \mathbb{E}(X) = \sum_{i=1}^{n} p_i$. Then*

$$\Pr[X \ge (1+\delta)\mu] \le e^{-\frac{\delta^2}{2+\delta}\mu} \text{ for all } \delta > 0;$$

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

Besides, we also employ some cryptographic primitives and their definitions are recalled in the full version of this paper.

4 Definition of Collusion Resistant Watermarkable PRF

In this section, we give the formal definition of the collusion resistant watermarkable PRF, which is adapted from definitions of watermarkable PRF in previous works [CHN$^+$16,BLW17,KW17]. The main difference between our definition and previous ones is that the unremovability holds against an adversary that can obtain polynomially-many (instead of one) watermarked circuits for the same PRF key from the challenge oracle. Besides, the extraction algorithm takes an additional parameter q, which can be roughly viewed as the number of colluders, as input. The correctness and the unforgeability hold for arbitrary positive integer q; for the unremovability, a large enough q is needed. In particular, if q is larger than the number of colluders, the extraction algorithm can extract a non-empty subset of the coalition, while using a smaller q may lead to an error symbol.

Remark 4.1. Our definition of collusion resistant unremovability implicitly requires that the adversary is only allowed to obtain a bounded number (i.e., q) of watermarked circuits from the challenge oracle. Thus, it falls into the category of "bounded collusion resistance". Nonetheless, in our definition, the bound q does not need to be fixed in the setup phase and may be varied in different extraction procedures. In fact, if the extractor has a way to know the number of colluders in advance, the scheme remains secure against an arbitrary number of colluders. Besides, since the extraction algorithm is able to detect if a smaller q is used, in practice, the extractor can re-execute the extraction algorithm with a larger q after receiving an error message from the extraction algorithm.

Definition 4.1 (Watermarkable PRFs [CHN$^+$16, BLW17, KW17, adapted]). *Let* $\mathsf{PRF} = (\mathsf{PRF}.\mathtt{KeyGen}, \mathsf{PRF}.\mathtt{Eval})$ *be a PRF family with key space* $\mathcal{K}$*, input space* $\{0,1\}^n$ *and output space* $\{0,1\}^m$*. The watermarking scheme with message space* $\mathcal{M}$ *for* PRF *(more accurately, the evaluation algorithm of* PRF*) consists of three algorithms:*

- ***Setup.*** *On input the security parameter* λ*, the setup algorithm outputs the mark key* MK *and the extraction key* EK*.*
- ***Mark.*** *On input the mark key* MK*, a secret key* $k \in \mathcal{K}$ *of* PRF*, and a message* $msg \in \mathcal{M}$*, the marking algorithm outputs a marked circuit* $\mathtt{C}$*.*
- ***Extract.*** *On input the extraction key* EK*, a circuit* $\mathtt{C}$*, and a parameter* q*, the extraction algorithm outputs either a set* $\mathcal{L} \subseteq \mathcal{M}$ *or a symbol* $\mathsf{UNMARKED}$ *or an error symbol* $\perp$*.*

Definition 4.2 (Watermarking Correctness). *Correctness of the watermarking scheme requires that for any* $k \in \mathcal{K}$*,* $msg \in \mathcal{M}$*, and any polynomial* $q \geq 1$*, let* $(MK, EK) \leftarrow \mathtt{Setup}(1^\lambda)$*,* $\mathtt{C} \leftarrow \mathtt{Mark}(MK, k, msg)$*, we have:*

- ***Functionality Preserving.*** *$\mathtt{C}(\cdot)$ and $\mathsf{PRF}.\mathtt{Eval}(k,\cdot)$ compute identically on all but a negligible fraction of inputs.*
- ***Extraction Correctness.*** $\Pr[\mathtt{Extract}(EK,\mathtt{C},q) \neq \{msg\}] \leq negl(\lambda)$.

Before presenting the security definition of collusion resistant watermarkable PRF, we first introduce oracles the adversaries can query during the security experiments. Here, the marking oracle is identical to the one defined in previous works, while we redefine the challenge oracle to capture the scenario that the adversary can obtain multiple circuits embedded with different messages for the same secret key.

- **Marking Oracle $\mathcal{O}^{M}_{MK}(\cdot,\cdot)$.** On input a message $msg \in \mathcal{M}$ and a PRF key $k \in \mathcal{K}$, the oracle returns a circuit $\mathtt{C} \leftarrow \mathtt{Mark}(MK,k,msg)$.
- **Challenge Oracle $\mathcal{O}^{C}_{MK}(\cdot)$.** On input a polynomial-size set M of messages from $\mathcal{M}$, the oracle first samples a key $k^* \leftarrow \mathsf{PRF}.\mathtt{KeyGen}(1^\lambda)$. Then, for each $msg^*_i \in \mathsf{M}$, it computes $\mathtt{C}^*_i \leftarrow \mathtt{Mark}(MK,k^*,msg^*_i)$. Finally, it returns the set $\{\mathtt{C}^*_1,\ldots,\mathtt{C}^*_Q\}$, where $Q = \|\mathsf{M}\|$.

Definition 4.3 (Collusion Resistant Unremovability). *The watermarking scheme for a PRF is collusion resistant unremovable if for any polynomial q, for all polynomial-time (PPT) and unremoving-admissible adversaries $\mathcal{A}$, we have* $\Pr[\mathtt{ExptUR}_{\mathcal{A},q}(\lambda) = 1] \leq negl(\lambda)$*, where we define the experiment* $\mathtt{ExptUR}$ *and unremoving-admissibility as follows:*

1. *The challenger samples $(MK,EK) \leftarrow \mathtt{Setup}(1^\lambda)$ and returns EK to $\mathcal{A}$.*
2. *Then, $\mathcal{A}$ is allowed to make multiple queries to the marking oracle.*
3. *Next, $\mathcal{A}$ submits a set M^* of Q messages in $\mathcal{M}$ to the challenge oracle and gets a set C^* of circuits back.*
4. *Then, $\mathcal{A}$ is further allowed to make multiple queries to the marking oracle.*
5. *Finally $\mathcal{A}$ submits a circuit $\tilde{\mathtt{C}}$. The experiment outputs 0 if*
   (a) $q < Q$ and either $\mathtt{Extract}(EK,\tilde{\mathtt{C}},q)$ is a non-empty subset of M^* or it equals to the error symbol $\perp$.
   (b) $q \geq Q$ and $\mathtt{Extract}(EK,\tilde{\mathtt{C}},q)$ is a non-empty subset of M^*.
   Otherwise, the experiment outputs 1.

Here, an adversary $\mathcal{A}$ is unremoving-admissible if there exists circuit $\mathtt{C}^*_i \in \mathsf{C}^*$ that $\mathtt{C}^*_i$ and $\tilde{\mathtt{C}}$ compute identically on all but a negligible fraction of inputs.

Definition 4.4 (δ-Unforgeability). *The watermarking scheme for a PRF is δ-unforgeable if for any polynomial $q \geq 1$ and for all PPT and δ-unforging-admissible adversaries $\mathcal{A}$, we have* $\Pr[\mathtt{ExptUF}_{\mathcal{A},q}(\lambda) = 1] \leq negl(\lambda)$*, where we define the experiment* $\mathtt{ExptUF}$ *and unforging-admissiability as follows:*

1. *The challenger samples $(MK,EK) \leftarrow \mathtt{Setup}(1^\lambda)$ and returns EK to $\mathcal{A}$.*
2. *Then, $\mathcal{A}$ is allowed to make multiple queries to the marking oracle.*
3. *Finally, $\mathcal{A}$ submits a circuit $\tilde{\mathtt{C}}$. The experiment outputs 0 if $\mathtt{Extract}(EK,\tilde{\mathtt{C}}, q) = \mathsf{UNMARKED}$; otherwise, the experiment output 1.*

Here, let Q' be the number of queries $\mathcal{A}$ made to the marking oracle, then an adversary $\mathcal{A}$ is δ-unforging-admissible if for all $i \in [1,Q']$, its submitted circuit $\tilde{\mathtt{C}}$ and the circuit $\mathtt{C}_i$ compute differently on at least a δ fraction of inputs, where $\mathtt{C}_i$ is the output of the marking oracle on the ith query.

5 Puncturable Functional Encryption

In this section, we define puncturable functional encryption and give a concrete construction. A puncturable functional encryption scheme can achieve functionalities and security of both puncturable encryption schemes and functional encryption schemes. Besides, as we will use the puncturable functional encryption scheme together with an indistinguishability obfuscator, we also require it to have an "iO-compatible correctness", which demands a decryption consistency for different secret keys. More precisely, when using two secret keys sk_1, sk_2 for functions f_1, f_2 respectively, for any string *ct* in the ciphertext space, either both secret keys will fail in decrypting it or there exists a plaintext μ in the plaintext space that decrypting *ct* under sk_1 and sk_2 will lead to $f_1(\mu)$ and $f_2(\mu)$ respectively.

5.1 Definition of Puncturable Functional Encryption

Definition 5.1 (Puncturable Functional Encryption). *A puncturable functional encryption scheme for a family of function $\mathcal{F}$[8] with plaintext space $\{0,1\}^m$ and ciphertext space $\{0,1\}^n$ consists of five algorithms:*

- ***Setup.*** *On input the security parameter λ, the setup algorithm outputs the master public key/master secret key pair (mpk, msk).*
- ***KeyGen.*** *On input the master secret key msk and a function $f \in \mathcal{F}$, the key generation algorithm outputs a secret key sk for f.*
- ***Enc.*** *On input the master public key mpk and a message $msg \in \{0,1\}^m$, the encryption algorithm outputs the ciphertext ct.*
- ***Dec.*** *On input a secret key sk and a ciphertext $ct \in \{0,1\}^n$, the decryption algorithm outputs a string msg or a decryption failure symbol $\perp$.*
- ***Puncture.*** *On input a secret key sk and two ciphertexts ct_1, ct_2, the puncture algorithm outputs a punctured secret key sk'.*

Next, we describe properties of puncturable functional encryption schemes.

Definition 5.2 (Properties of Puncturable Functional Encryption). *A puncturable functional encryption scheme* $\mathsf{PFE} = (\texttt{Setup}, \texttt{KeyGen}, \texttt{Enc}, \texttt{Dec}, \texttt{Puncture})$ *with plaintext space $\{0,1\}^m$, ciphertext space $\{0,1\}^n$ and supported function family $\mathcal{F}$ is required to have the following properties.*

- ***Correctness.*** *For any message $msg \in \{0,1\}^m$ and any $f \in \mathcal{F}$, let $(mpk, msk) \leftarrow \texttt{Setup}(1^\lambda)$, $sk \leftarrow \texttt{KeyGen}(msk, f)$, and $ct \leftarrow \texttt{Enc}(mpk, msg)$, then we have $\Pr[\texttt{Dec}(sk, ct) = f(msg)] = 1$.*
- ***Sparseness.*** *For any $f \in \mathcal{F}$, let $(mpk, msk) \leftarrow \texttt{Setup}(1^\lambda)$, $sk \leftarrow \texttt{KeyGen}(msk, f)$, and $ct \overset{\$}{\leftarrow} \{0,1\}^n$, then we have $\Pr[\texttt{Dec}(sk, ct) \neq \perp] \leq negl(\lambda)$.*

[8] In this work, we concentrate on schemes supporting function family $\mathcal{F}$ of polynomial-size circuit with output space $\{0,1\}^m$.

- ***Punctured Correctness.*** *For any $f \in \mathcal{F}$, any strings $ct_0, ct_1 \in \{0,1\}^n$ and any* unbounded *adversary $\mathcal{A}$, we have*

$$\Pr\left[\begin{array}{l} (mpk, msk) \leftarrow \texttt{Setup}(1^\lambda); \\ sk \leftarrow \texttt{KeyGen}(msk, f); \\ sk' \leftarrow \texttt{Puncture}(sk, \{ct_0, ct_1\}); \\ ct \leftarrow \mathcal{A}(mpk, msk, sk, sk'); \end{array} : \begin{array}{r} ct \notin \{ct_0, ct_1\} \wedge \\ \texttt{Dec}(sk, ct) \neq \texttt{Dec}(sk', ct) \end{array}\right] \leq negl(\lambda)$$

- ***iO-Compatible Correctness.*** *For each master public key/master secret key pair (mpk, msk), the ciphertext space can be divided into two disjoint parts, namely, the valid ciphertext set $\mathcal{V}_{(mpk,msk)}$ and the invalid ciphertext set $\mathcal{I}_{(mpk,msk)}$, which satisfy $\mathcal{V}_{(mpk,msk)} \cup \mathcal{I}_{(mpk,msk)} = \{0,1\}^n$ and $\mathcal{V}_{(mpk,msk)} \cap \mathcal{I}_{(mpk,msk)} = \emptyset$. The iO-compatible correctness requires that:*

  1. *For any $f \in \mathcal{F}$ and any* unbounded *adversary $\mathcal{A}$, we have:*

$$\Pr\left[\begin{array}{l} (mpk, msk) \leftarrow \texttt{Setup}(1^\lambda); \\ sk \leftarrow \texttt{KeyGen}(msk, f); \\ ct \leftarrow \mathcal{A}(mpk, msk, sk); \end{array} : \begin{array}{c} ct \in \mathcal{I}_{(mpk,msk)} \wedge \\ \texttt{Dec}(sk, ct) \neq \perp \end{array}\right] \leq negl(\lambda)$$

  2. *For any $f_1, f_2 \in \mathcal{F}$ and any* unbounded *adversary $\mathcal{A}$, we have:*

$$\Pr\left[\begin{array}{ll} (mpk, msk) \leftarrow \texttt{Setup}(1^\lambda); & ct \in \mathcal{V}_{(mpk,msk)} \wedge \\ sk_1 \leftarrow \texttt{KeyGen}(msk, f_1); & (\forall msg \in \{0,1\}^m, \\ sk_2 \leftarrow \texttt{KeyGen}(msk, f_2); & \texttt{Dec}(sk_1, ct) \neq f_1(msg) \vee \\ ct \leftarrow \mathcal{A}(mpk, msk, sk_1, sk_2); & \texttt{Dec}(sk_2, ct) \neq f_2(msg)) \end{array}\right] \leq negl(\lambda)$$

- ***Adaptive Indistinguishability.*** *For any PPT adversary $\mathcal{A}_1$, $\mathcal{A}_2$, we have:*

$$\Pr\left[\begin{array}{l} (mpk, msk) \leftarrow \texttt{Setup}(1^\lambda); \\ (st, msg_0, msg_1) \leftarrow \mathcal{A}_1^{\mathcal{O}_{msk}(\cdot)}(mpk); \\ b \leftarrow \{0,1\}; \\ ct \leftarrow \texttt{Enc}(mpk, msg_b); \\ b' \leftarrow \mathcal{A}_2(st, ct); \end{array} : b = b'\right] \leq 1/2 + negl(\lambda)$$

where $\mathcal{O}_{msk}$ takes as input a function $f \in \mathcal{F}$ and outputs a secret key $sk \leftarrow$ $\texttt{KeyGen}(msk, f)$; for all f submitted to the oracle $\mathcal{O}_{msk}$, $f(msg_0) = f(msg_1)$; and the $\mathcal{O}_{msk}$ can only be queried two times. Note that in our security proof for collusion resistant watermarkable PRF, we only require a two-key security, thus we just define this type of adaptive indistinguishability here.

- ***Ciphertext Pseudorandomness.*** *For any PPT adversary $\mathcal{A}_1$, $\mathcal{A}_2$, we have:*

$$\Pr\left[\begin{array}{l} (st, msg, f) \leftarrow \mathcal{A}_1(1^\lambda); \\ (mpk, msk) \leftarrow \texttt{Setup}(1^\lambda); \\ sk \leftarrow \texttt{KeyGen}(msk, f); \\ ct_0 \leftarrow \texttt{Enc}(mpk, msg); \\ ct_1 \overset{\$}{\leftarrow} \{0,1\}^n; \\ sk' \leftarrow \texttt{Puncture}(sk, \{ct_0, ct_1\}); \\ b \leftarrow \{0,1\}; \\ b' \leftarrow \mathcal{A}_2(st, mpk, sk', ct_b, ct_{1-b}); \end{array} : b = b' \right] \leq 1/2 + negl(\lambda)$$

5.2 Construction of Puncturable Functional Encryption

In this section, we present our construction of puncturable functional encryption.

Let λ be the security parameter. Let n, m, l, n' be positive integers that are polynomial in λ. Our construction is based on the following three building blocks:

- A functional encryption scheme $\mathsf{FE} = (\mathsf{FE}.\texttt{Setup}, \mathsf{FE}.\texttt{KeyGen}, \mathsf{FE}.\texttt{Enc}, \mathsf{FE}.\texttt{Dec})$ with plaintext space $\{0,1\}^m$, ciphertext space $\{0,1\}^n$ and encryption randomness space $\mathcal{R}$. Also, we require that it supports a family $\mathcal{F}$ of polynomial-size circuit with output space $\{0,1\}^m$.
- A statistically sound NIZK proof system $\mathsf{NIZK} = (\mathsf{NIZK}.\texttt{KeyGen}, \mathsf{NIZK}.\texttt{Prove}, \mathsf{NIZK}.\texttt{Verify})$ for $\mathcal{L}$, where

$$\mathcal{L} = \{(mpk, ct) : \exists (msg, r), ct = \mathsf{FE}.\texttt{Enc}(mpk, msg; r)\}.$$

  and require that the proof size is n' when proving a statement in $\mathcal{L}$.
- A puncturable encryption scheme $\mathsf{PE} = (\mathsf{PE}.\texttt{KeyGen}, \mathsf{PE}.\texttt{Puncture}, \mathsf{PE}.\texttt{Enc}, \mathsf{PE}.\texttt{Dec})$ with plaintext space $\{0,1\}^{n+n'}$ and ciphertext space $\{0,1\}^l$.

We construct $\mathsf{PFE} = (\mathsf{PFE}.\texttt{Setup}, \mathsf{PFE}.\texttt{KeyGen}, \mathsf{PFE}.\texttt{Puncture}, \mathsf{PFE}.\texttt{Enc}, \mathsf{PFE}.\texttt{Dec})$ for $\mathcal{F}$, which has a plaintext space $\{0,1\}^m$ and a ciphertext space $\{0,1\}^l$, as follows:

- **Setup.** On input a security parameter λ, the setup algorithm generates $(mpk, msk) \leftarrow \mathsf{FE}.\texttt{Setup}(1^\lambda)$, $crs \leftarrow \mathsf{NIZK}.\texttt{KeyGen}(1^\lambda)$, and $(pk, sk) \leftarrow \mathsf{PE}.\texttt{KeyGen}(1^\lambda)$. Then it outputs the master public key $MPK = (mpk, crs, pk)$ and the master secret key $MSK = (msk, sk, mpk, crs)$ of PFE.
- **KeyGen.** On input a master secret key $MSK = (msk, sk, mpk, crs)$ of PFE and a function $f \in \mathcal{F}$, the key generation algorithm generates $fsk \leftarrow \mathsf{FE}.\texttt{KeyGen}(msk, f)$ and outputs a secret key $SK = (fsk, sk, mpk, crs)$ of PFE.

- **Enc.** On input a master public key $MPK = (mpk, crs, pk)$ of PFE and a message $msg \in \{0,1\}^m$, the encryption algorithm first samples $r \in \mathcal{R}$. Then, it computes $ct = \mathsf{FE}.\mathtt{Enc}(mpk, msg; r)$, and $\pi \leftarrow \mathsf{NIZK}.\mathtt{Prove}(crs, (mpk, ct), (msg, r))$. Finally, it outputs $CT \leftarrow \mathsf{PE}.\mathtt{Enc}(pk, ct\|\pi)$.
- **Dec.** On input a secret key $SK = (fsk, sk, mpk, crs)$ of PFE and a ciphertext $CT \in \{0,1\}^l$, the decryption algorithm first decrypts CT with the secret key of PE and gets $ct\|\pi \leftarrow \mathsf{PE}.\mathtt{Dec}(sk, CT)$. It aborts and outputs $\perp$ if $ct\|\pi = \perp$ or $\mathsf{NIZK}.\mathtt{Verify}(crs, (mpk, ct), \pi) = 0$. Otherwise, it outputs $\mathsf{FE}.\mathtt{Dec}(fsk, ct)$.
- **Puncture.** On input a secret key $SK = (fsk, sk, mpk, crs)$ of PFE and two ciphertexts $CT_1, CT_2 \in \{0,1\}^l$, the puncture algorithm generates $sk' \leftarrow \mathsf{PE}.\mathtt{Puncture}(sk, \{CT_1, CT_2\})$ and outputs $SK' = (fsk, sk', mpk, crs)$.

Theorem 5.1. *If* FE *is a secure functional encryption for* $\mathcal{F}$ *with perfect correctness and (two-key) adaptive security,* NIZK *is a NIZK proof system with adaptively statistical soundness and adaptive zero-knowledge for language* $\mathcal{L}$*, and* PE *is a secure puncturable encryption scheme, then* PFE *is a secure puncturable functional encryption as defined in Sec. 5.1.*

We give proof of Theorem 5.1 in the full version of this paper.

6 Construction of Collusion Resistant Watermarkable PRF

In this section, we show how to obtain collusion resistant watermarkable PRF families. In particular, we construct a collusion resistant watermarking scheme for any puncturable PRF with weak key-injectivity and constrained one-wayness.

Let λ be the security parameter. Let δ be a positive real value and $d = \lambda/\delta = poly(\lambda)$. Let n, m, l, κ be positive integers that are polynomial in λ and $n = l + poly(\lambda)$. Let

$$\mathsf{PRF} = (\mathsf{PRF}.\mathtt{KeyGen}, \mathsf{PRF}.\mathtt{Eval}, \mathsf{PRF}.\mathtt{Constrain}, \mathsf{PRF}.\mathtt{ConstrainEval})$$

be a family of puncturable PRF with key space $\mathcal{K}$, input space $\{0,1\}^n$, and output space $\{0,1\}^m$. Our watermarking scheme for PRF is built on the following building blocks.

- A puncturable functional encryption scheme $\mathsf{PFE} = (\mathsf{PFE}.\mathtt{Setup}, \mathsf{PFE}.\mathtt{KeyGen}, \mathsf{PFE}.\mathtt{Puncture}, \mathsf{PFE}.\mathtt{Enc}, \mathsf{PFE}.\mathtt{Dec})$ with plaintext space $\{0,1\}^{(d+1)\cdot l+\kappa}$, ciphertext space $\{0,1\}^n$ and encryption randomness space $\mathcal{R}$. Also, we require that it supports a family $\mathcal{F}$ of polynomial-size circuits with output space $\{0,1\}^{(d+1)\cdot l+\kappa}$.
- A family of prefix puncturable PRF $\mathsf{F} = (\mathsf{F}.\mathtt{KeyGen}, \mathsf{F}.\mathtt{Eval}, \mathsf{F}.\mathtt{Constrain}, \mathsf{F}.\mathtt{ConstrainEval})$ with input space $\{0,1\}^{(d+1)\cdot l}$ and output space $\mathcal{K}$.
- An indistinguishability obfuscator iO for all polynomial-size circuits.
- Two pseudorandom generators $\mathsf{G} : \{0,1\}^l \to \{0,1\}^n$ and $\mathsf{G}' : \{0,1\}^{\frac{l}{2}} \to \{0,1\}^l$.

- A family of collision-resistant hash function $\mathcal{H}$ with input space $\{0,1\}^{d\cdot m}$ and output space $\{0,1\}^{l}$.

We construct $\mathsf{WM} = (\mathsf{WM}\,.\,\texttt{Setup}, \mathsf{WM}\,.\,\texttt{Mark}, \mathsf{WM}\,.\,\texttt{Extract})$, which has a message space $\{0,1\}^{\kappa}\backslash\{0^{\kappa}\} = [1, 2^{\kappa}-1]$, as follows:

- **Setup.** On input a security parameter λ, the setup algorithm first samples $H \xleftarrow{\$} \mathcal{H}$ and generates $K \leftarrow \mathsf{F}\,.\,\texttt{KeyGen}(1^{\lambda})$. Then it generates $(mpk, msk) \leftarrow \mathsf{PFE}\,.\,\texttt{Setup}(1^{\lambda})$ and $sk \leftarrow \mathsf{PFE}\,.\,\texttt{KeyGen}(msk, \texttt{ID})$, where $\texttt{ID} : \{0,1\}^{(d+1)\cdot l+\kappa} \rightarrow \{0,1\}^{(d+1)\cdot l+\kappa}$ is the identity function, i.e., for any $x \in \{0,1\}^{(d+1)\cdot l+\kappa}$, $ID(x) = x$. Next, it computes $\texttt{E} \leftarrow \mathsf{iO}(\texttt{Ext}[mpk, K])$, where $\texttt{Ext}$ is defined in Figure 1[9]. Finally, the output of the setup algorithm is (MK, EK) where $MK = (sk, K, H)$ and $EK = (H, \texttt{E})$.
- **Mark.** On input a mark key $MK = (sk, K, H)$, a secret key $k \in \mathcal{K}$ for PRF and a message msg, the marking algorithm outputs a circuit $\texttt{C} \leftarrow \mathsf{iO}(\texttt{M}[sk, K, H, k, msg])$, where $\texttt{M}$ is defined in Figure 1[10].
- **Extract.** On input an extraction key $EK = (H, \texttt{E})$, a circuit $\texttt{C}$, and a parameter q, the extraction algorithm first computes $\epsilon = 1/((\kappa+1)\cdot q + 1)$, $T = \lambda/\epsilon^2$, and $S = q \cdot (\kappa+1)$ and sets a variable *counter* $= 0$. Then it computes $\mathcal{L} = \texttt{Trace}(0, 2^{\kappa}, 1, 0, \epsilon, T, \texttt{E}, H, \texttt{C})$, where $\texttt{Trace}(\cdot)$ is defined in Figure 1.
  In this procedure, the algorithm also maintains the variable *counter* and increase it by 1 each time the function $\texttt{Test}(\cdot)$ defined in Figure 1 is invoked. The algorithm aborts and outputs $\perp$ once *counter* $> S$. In case the algorithm does not abort, it checks the set $\mathcal{L}$ returned by $\texttt{Trace}$. It outputs $\perp$ if $\mathcal{L} = \emptyset$ and outputs $\mathsf{UNMARKED}$ if $\mathcal{L} = \{0\}$. Otherwise, it outputs $\mathcal{L}$.

Theorem 6.1. *If* PRF *is a secure puncturable PRF with weak key-injectivity and constrained one-wayness,* PFE *is a secure puncturable functional encryption scheme as defined in Sec. 5.1,* F *is a secure prefix puncturable PRF,* G *and* G' *are pseudorandom generators,* $\mathcal{H}$ *is a family of collision-resistant hash function, and* iO *is a secure indistinguishability obfuscator, then* WM *is a secure watermarking scheme with collusion resistant unremovability and* δ*-unforgeability, as defined in Sec. 4, for* PRF.

We present the proof of Theorem 6.1 in the full version of this paper.

Here, we provide a brief overview on how to prove the collusion resistant unremovability of WM. For simplicity, we consider an adversary who only gets two circuits $\texttt{C}_1$ and $\texttt{C}_2$ for the same secret key k embedded with messages msg_1 and msg_2 respectively, where $msg_1 < msg_2$, and omit its advantage in viewing the public key and querying the marking oracle.

[9] The circuit $\texttt{Ext}$, as well as all circuits $\texttt{Ext}^{(\cdot)}$ appeared in the security proofs for WM will be padded to the same size.

[10] The circuit $\texttt{M}$, as well as all circuits $\texttt{M}^{(\cdot)}$ appeared in the security proof for WM will be padded to the same size.

```
Ext
Constant: $mpk, K$
Input: $a_1, \dots, a_d, b, ind, r$

  1. $t_1 = \mathsf{G}'(a_1), \dots, t_d = \mathsf{G}'(a_d)$.
  2. $x = \mathsf{PFE}.\mathtt{Enc}(mpk, t_1 \| \dots \| t_d \| b \| ind; r)$.
  3. $k' = \mathsf{F}.\mathtt{Eval}(K, t_1 \| \dots \| t_d \| b)$.
  4. $y = \mathsf{PRF}.\mathtt{Eval}(k', x)$.
  5. Output $(x, y)$.
```

```
Trace
Input: $ind_1, ind_2, p_1, p_2, \epsilon, T, \mathtt{E}, H, \mathtt{C}$
  1. $\Delta = |p_1 - p_2|$.
  2. If $\Delta \le \epsilon$: return $\emptyset$.
  3. If $ind_2 - ind_1 = 1$: return $\{ind_1\}$.
  4. $ind_3 = \lfloor \frac{ind_1 + ind_2}{2} \rfloor$.
  5. $p_3 = \mathtt{Test}(ind_3, T, \mathtt{E}, H, \mathtt{C})$.
  6. Return $\mathtt{Trace}(ind_1, ind_3, p_1, p_3, \epsilon, T, \mathtt{E}, H, \mathtt{C}) \cup \mathtt{Trace}(ind_3, ind_2, p_3, p_2, \epsilon, T, \mathtt{E}, H, \mathtt{C})$.
```

```
M
Constant: $sk, K, H, k, msg$
Input: $x$

  1. $(t_1 \| \dots \| t_d \| b \| ind) = \mathsf{PFE}.\mathtt{Dec}(sk, x)$.
  2. If $(t_1 \| \dots \| t_d \| b \| ind \ne \perp) \wedge (ind \le msg) \wedge (H(\mathsf{PRF}.\mathtt{Eval}(k, \mathsf{G}(t_1)), \dots, \mathsf{PRF}.\mathtt{Eval}(k, \mathsf{G}(t_d))) = b)$
     (a) $k' = \mathsf{F}.\mathtt{Eval}(K, t_1 \| \dots \| t_d \| b)$.
     (b) $y = \mathsf{PRF}.\mathtt{Eval}(k', x)$.
     (c) Output $y$.
  3. Otherwise, output $\mathsf{PRF}.\mathtt{Eval}(k, x)$.
```

```
Test
Input: $ind, T, \mathtt{E}, H, \mathtt{C}$
  1. $Acc = 0$
  2. For $i \in [1, T]$:
     (a) Sample $a_1, \dots, a_d \xleftarrow{\$} \{0,1\}^{\frac{l}{2}}$ and $r \xleftarrow{\$} \mathcal{R}$.
     (b) $t_1 = \mathsf{G}'(a_1), \dots, t_d = \mathsf{G}'(a_d)$.
     (c) $b = H(\mathtt{C}(\mathsf{G}(t_1)), \dots, \mathtt{C}(\mathsf{G}(t_d)))$.
     (d) $(x, y) = \mathtt{E}(a_1, \dots, a_d, b, ind, r)$.
     (e) If $\mathtt{C}(x) = y$: $Acc = Acc + 1$.
  3. Return $\frac{Acc}{T}$.
```

Fig. 1 The circuit `Ext`, the circuit `M`, the function `Trace`, and the function `Test`

Following the syntax used in Sec. 2, we denote an input encrypted from $t_1 \| \dots \| t_d \| b \| ind$ satisfying $b = H(\mathsf{PRF}.\mathtt{Eval}(k, \mathsf{G}(t_1)), \dots, \mathsf{PRF}.\mathtt{Eval}(k, \mathsf{G}(t_d)))$ as a punctured point labeled with an index *ind*. Also, we use $\mathcal{X}_{ind}$ to denote the set of all punctured points labeled with the index *ind*.

First, as shown in [BCP14, NWZ16], the Trace algorithm can output a non-empty subset of $\{msg_1, msg_2\}$ if the adversary cannot distinguish 1) two punctured points labeled with different indices adaptively chosen from $(msg_1, msg_2]$ and 2) a punctured point labeled with an index adaptively chosen outside $(msg_1, msg_2]$ and a random point.

For two punctured points in $\mathcal{X}_{ind_1}$ and $\mathcal{X}_{ind_2}$ respectively, where $ind_1, ind_2 \in (msg_1, msg_2]$, both of them are properly punctured and reprogrammed in $\mathtt{C}_2$ while none of them are punctured in $\mathtt{C}_1$, thus the decryption (in both $\mathtt{C}_1$ and $\mathtt{C}_2$) do not need to distinguish them. So, their indistinguishability comes from the adaptive indistinguishability of PFE.

The adaptive indistinguishability of PFE also implies indistinguishability of two punctured points in $\mathcal{X}_{ind_1}$ and $\mathcal{X}_{ind_2}$ when both ind_1 and ind_2 are in $[1, msg_1]$ or both of them are in $(msg_2, 2^\kappa - 1]$. This could reduce the problem of claiming the pseudorandomness of a punctured point labeled with an index adaptively chosen from $[1, msg_1]$ (or $(msg_2, 2^\kappa - 1]$) to the problem of claiming the pseudorandomness of a punctured points from $\mathcal{X}_1$ (resp. $\mathcal{X}_{2^\kappa - 1}$), where the latter claim can be implied by the ciphertext pseudorandomness of PFE. In this way, pseudorandomness of punctured points in $\mathcal{X}_{ind}$ for $ind \notin (msg_1, msg_2]$ is proved.

It is worth noting that when arguing indistinguishability between a punctured point from $\mathcal{X}_1$ and a random input, we also need to show that the marked circuits are able to hide punctured points that are punctured and identically reprogrammed in all circuits. This indicates that our construction of watermarkable PRF involves a collusion resistant constraint-hiding constrained PRF implicitly.

7 Collusion Resistant Watermarking Schemes for Other Cryptographic Functionalities

In this section, we show how to construct watermarking schemes for advanced cryptographic functionalities, including the decryption algorithm of a PKE scheme and the signing algorithm of a signature scheme. The constructions are based on the observation that the PKE scheme (and the signature scheme) constructed in [SW14] has a decryption algorithm (resp. signing algorithm) that is nothing more than a puncturable PRF evaluation. The observation was initially presented in [NW15, CHN+16] and was used to construct the watermarkable PKE scheme and the watermarkable signature scheme therein.

Here, as an example, we give a detailed description for how to construct collusion resistant watermarkable PKE schemes and omit the construction for collusion resistant watermarkable signature schemes. We start by presenting the formal definition of watermarkable PKE scheme. Then we give our construction based on a puncturable PRF, an indistinguishability obfuscator, a puncturable functional encryption scheme, and some standard cryptographic primitives.

7.1 The Definition

The collusion resistant watermarkable PKE scheme can be defined similarly as collusion resistant watermarkable PRF, with the main difference being that in the challenge oracle, the adversary is further given the public key corresponding to the watermarked secret key.

Definition 7.1 (Watermarkable PKEs [CHN+16, adapted]). *Let* $\mathsf{PKE} = (\mathsf{PKE}.\mathtt{KeyGen}, \mathsf{PKE}.\mathtt{Enc}, \mathsf{PKE}.\mathtt{Dec})$ *be a PKE scheme with secret key space* $\mathcal{SK}$*. The watermarking scheme with message space* $\mathcal{M}$ *for* PKE *(more accurately, the decryption algorithm of* PKE*) consists of three algorithms:*

- ***Setup.*** *On input the security parameter* λ*, the setup algorithm outputs the mark key* MK *and the extraction key* EK*.*
- ***Mark.*** *On input the mark key* MK*, a secret key* $sk \in \mathcal{SK}$ *of* PKE*, and a message* $msg \in \mathcal{M}$*, the marking algorithm outputs a marked circuit* $\mathtt{C}$*.*
- ***Extract.*** *On input the extraction key* EK*, a circuit* $\mathtt{C}$*, and a parameter* q*, the extraction algorithm outputs either a set* $\mathcal{L} \subseteq \mathcal{M}$ *or a symbol* $\mathsf{UNMARKED}$ *or an error symbol* $\perp$*.*

Definition 7.2 (Watermarking Correctness). *Correctness of the watermarking scheme requires that for any* $sk \in \mathcal{SK}$*,* $msg \in \mathcal{M}$*, and any polynomial* $q \geq 1$*, let* $(MK, EK) \leftarrow \mathtt{Setup}(1^\lambda)$*,* $\mathtt{C} \leftarrow \mathtt{Mark}(MK, sk, msg)$*, we have:*

- ***Functionality Preserving.*** *$\mathtt{C}(\cdot)$ and $\mathsf{PKE}.\mathtt{Dec}(sk, \cdot)$ compute identically on all but a negligible fraction of inputs.*
- ***Extraction Correctness.*** *$\Pr[\mathtt{Extract}(EK, \mathtt{C}, q) \neq \{msg\}] \leq negl(\lambda)$.*

Before presenting the security definition of the collusion resistant watermarkable PKE, we first introduce oracles the adversaries can query during the security experiments. Note that in the challenge oracle, the adversary is further given the challenge public key.

- **Marking Oracle $\mathcal{O}^M_{MK}(\cdot, \cdot)$.** On input a message $msg \in \mathcal{M}$ and a secret key key $sk \in \mathcal{SK}$, the oracle returns a circuit $\mathtt{C} \leftarrow \mathtt{Mark}(MK, sk, msg)$.
- **Challenge Oracle $\mathcal{O}^C_{MK}(\cdot)$.** On input a polynomial-size set M of messages from $\mathcal{M}$, the oracle first generates a key pair $(sk^*, pk^*) \leftarrow \mathsf{PKE}.\mathtt{KeyGen}(1^\lambda)$. Then, for each $msg^*_i \in \mathsf{M}$, it computes $\mathtt{C}^*_i \leftarrow \mathtt{Mark}(MK, sk^*, msg^*_i)$. Finally, it returns the set $\{\mathtt{C}^*_1, \ldots, \mathtt{C}^*_Q\}$, where $Q = \|\mathsf{M}\|$, and the public key pk^*.

Definition 7.3 (Collusion Resistant Unremovability). *The watermarking scheme for a PKE is collusion resistant unremovable if for any polynomial q, for all PPT and unremoving-admissible adversaries $\mathcal{A}$, we have $\Pr[\mathtt{ExptUR}_{\mathcal{A},q}(\lambda) = 1] \leq negl(\lambda)$, where we define the experiment* ExptUR *and unremoving-admissibility as follows:*

1. *The challenger samples $(MK, EK) \leftarrow \mathtt{Setup}(1^\lambda)$ and returns EK to $\mathcal{A}$.*
2. *Then, $\mathcal{A}$ is allowed to make multiple queries to the marking oracle.*
3. *Next, $\mathcal{A}$ submits a set M^* of Q messages in $\mathcal{M}$ to the challenge oracle and gets a set C^* of circuits as well as a public key pk^* back.*
4. *Then, $\mathcal{A}$ is further allowed to make multiple queries to the marking oracle.*
5. *Finally $\mathcal{A}$ submits a circuit $\tilde{\mathtt{C}}$. The experiment outputs 0 if*
   (a) $q < Q$ and either $\mathtt{Extract}(EK, \tilde{\mathtt{C}}, q)$ is a non-empty subset of M^* or it equals to the error symbol $\perp$.
   (b) $q \geq Q$ and $\mathtt{Extract}(EK, \tilde{\mathtt{C}}, q)$ is a non-empty subset of M^*.
   Otherwise, the experiment outputs 1.

Here, an adversary $\mathcal{A}$ is unremoving-admissible if there exists circuit $\mathtt{C}^*_i \in \mathsf{C}^*$ that $\mathtt{C}^*_i$ and $\tilde{\mathtt{C}}$ compute identically on all but a negligible fraction of inputs.

Definition 7.4 (δ-Unforgeability). *The watermarking scheme for a PKE is δ-unforgeable if for any polynomial $q \geq 1$ and for all PPT and δ-unforging-admissible adversaries $\mathcal{A}$, we have $\Pr[\mathtt{ExptUF}_{\mathcal{A},q}(\lambda) = 1] \leq negl(\lambda)$, where we define the experiment* ExptUF *and unforging-admissiability as follows:*

1. *The challenger samples $(MK, EK) \leftarrow \mathtt{Setup}(1^\lambda)$ and returns EK to $\mathcal{A}$.*
2. *Then, $\mathcal{A}$ is allowed to make multiple queries to the marking oracle.*
3. *Finally, $\mathcal{A}$ submits a circuit $\tilde{\mathtt{C}}$. The experiment outputs 0 if $\mathtt{Extract}(EK, \tilde{\mathtt{C}}, q) = \mathsf{UNMARKED}$; otherwise, the experiment output 1.*

Here, let Q' be the number of queries $\mathcal{A}$ made to the marking oracle, then an adversary $\mathcal{A}$ is δ-unforging-admissible if for all $i \in [1, Q']$, its submitted circuit $\tilde{\mathtt{C}}$ and the circuit $\mathtt{C}_i$ compute differently on at least a δ fraction of inputs, where $\mathtt{C}_i$ is the output of the marking oracle on the ith query.

7.2 The Construction

Let λ be the security parameter. Let δ be a positive real value and $d = \lambda/\delta = poly(\lambda)$. Let n, m, l, κ be positive integers that are polynomial in λ and $n = l + poly(\lambda)$. Our watermarkable PKE scheme is built from the following building blocks:

- A family of puncturable PRF $\mathsf{PRF} = (\mathsf{PRF}.\mathtt{KeyGen}, \mathsf{PRF}.\mathtt{Eval}, \mathsf{PRF}.\mathtt{Constrain}, \mathsf{PRF}.\mathtt{ConstrainEval})$ with key space $\mathcal{K}$, input space $\{0, 1\}^n$, and output space $\{0,1\}^m$.
- A puncturable functional encryption scheme $\mathsf{PFE} = (\mathsf{PFE}.\mathtt{Setup}, \mathsf{PFE}.\mathtt{KeyGen}, \mathsf{PFE}.\mathtt{Puncture}, \mathsf{PFE}.\mathtt{Enc}, \mathsf{PFE}.\mathtt{Dec})$ with plaintext space $\{0, 1\}^{(d+1)\cdot l+\kappa}$, ciphertext space $\{0,1\}^n$ and encryption randomness space $\mathcal{R}$. Also, we require that it supports a family $\mathcal{F}$ of polynomial-size circuit with output space $\{0,1\}^{(d+1)\cdot l+\kappa}$.
- A family of prefix puncturable PRF $\mathsf{F} = (\mathsf{F}.\mathtt{KeyGen}, \mathsf{F}.\mathtt{Eval}, \mathsf{F}.\mathtt{Constrain}, \mathsf{F}.\mathtt{ConstrainEval})$ with input space $\{0,1\}^{(d+1)\cdot l}$ and output space $\mathcal{K}$.
- An indistinguishability obfuscator iO for all polynomial-size circuits.
- Three pseudorandom generators $\mathsf{G} : \{0,1\}^l \to \{0,1\}^n$, $\mathsf{G}' : \{0,1\}^{\frac{l}{2}} \to \{0,1\}^l$, and $\tilde{\mathsf{G}} : \{0,1\}^\lambda \to \{0,1\}^n$.
- A family of collision-resistant hash function $\mathcal{H}$ with input space $\{0,1\}^{d\cdot m}$ and output space $\{0,1\}^l$.

For completeness, we first recall how PKE scheme PKE is constructed in [SW14].

- **KeyGen.** On input a security parameter λ, the key generation algorithm first samples $k \xleftarrow{\$} \mathcal{K}$. Then, it computes $\mathtt{P} \leftarrow \mathsf{iO}(\mathtt{Encrypt}[k])$, where $\mathtt{Encrypt}$ is defined in Figure 2 and is properly padded. Finally, the output of the key generation algorithm is (pk, sk) where $pk = \mathtt{P}$ and $sk = k$.
- **Enc.** On input a public key $pk = \mathtt{P}$ and a message $msg \in \{0,1\}^m$, the encryption algorithm samples $r \xleftarrow{\$} \{0,1\}^\lambda$ and outputs $\mathtt{P}(msg, r)$.
- **Dec.** On input a secret key $sk = k$ and a ciphertext $ct = (x, z)$, the decryption algorithm outputs $msg = \mathsf{PRF}.\mathtt{Eval}(k, x) \oplus z$.

```
                         Encrypt
Constant: $k$
Input: $msg, r$

 1. $x = \tilde{\mathsf{G}}(r)$.
 2. $z = \mathsf{PRF}.\mathtt{Eval}(k, x) \oplus msg$.
 3. Output $ct = (x, z)$.
```

Fig. 2 The circuit `Encrypt`.

Next, we construct the watermarking scheme $\mathsf{WM} = (\mathsf{WM}.\mathtt{Setup}, \mathsf{WM}.\mathtt{Mark}, \mathsf{WM}.\mathtt{Extract})$ for the above constructed PKE scheme, which has a message space $\{0,1\}^\kappa \backslash \{0^\kappa\} = [1, 2^\kappa - 1]$, as follows:

- **Setup.** On input a security parameter λ, the setup algorithm first samples $H \xleftarrow{\$} \mathcal{H}$ and generates $K \leftarrow \mathsf{F}.\mathtt{KeyGen}(1^\lambda)$. Then it generates $(mpk, msk) \leftarrow \mathsf{PFE}.\mathtt{Setup}(1^\lambda)$ and $sk \leftarrow \mathsf{PFE}.\mathtt{KeyGen}(msk, \mathtt{ID})$, where $\mathtt{ID} : \{0, 1\}^{(d+1)\cdot l+\kappa} \rightarrow \{0,1\}^{(d+1)\cdot l+\kappa}$ is the identity function, i.e., for any $x \in \{0, 1\}^{(d+1)\cdot l+\kappa}$, $ID(x) = x$. Next, it computes $\mathtt{E} \leftarrow \mathsf{iO}(\mathtt{Ext}[mpk, K])$, where $\mathtt{Ext}$ is defined in Figure 3 and is properly padded. Finally, the output of the setup algorithm is (MK, EK) where $MK = (sk, K, H)$ and $EK = (H, \mathtt{E})$.
- **Mark.** On input a mark key $MK = (sk, K, H)$, a secret key $k \in \mathcal{K}$ for PKE and a message msg, the marking algorithm outputs a circuit $\mathtt{C} \leftarrow \mathsf{iO}(\mathtt{M}[sk, K, H, k, msg])$, where $\mathtt{M}$ is defined in Figure 3 and is properly padded.
- **Extract.** On input an extraction key $EK = (H, \mathtt{E})$, a circuit $\mathtt{C}$, and a parameter q, the extraction algorithm first computes $\epsilon = 1/((\kappa + 1) \cdot q + 1)$, $T = \lambda/\epsilon^2$, and $S = q \cdot (\kappa + 1)$ and sets a variable *counter* $= 0$. Then it computes $\mathcal{L} = \mathtt{Trace}(0, 2^\kappa, 1, 0, \epsilon, T, \mathtt{E}, H, \mathtt{C})$, where $\mathtt{Trace}(\cdot)$ is defined in Figure 3.
  In this procedure, the algorithm also maintains the variable *counter* and increase it by 1 each time the function $\mathtt{Test}(\cdot)$ defined in Figure 3 is invoked. The algorithm aborts and outputs $\perp$ once *counter* exceeds S. In case the algorithm does not abort, it checks the set $\mathcal{L}$ returned by $\mathtt{Trace}$. It outputs $\perp$ if $\mathcal{L} = \emptyset$ and outputs $\mathsf{UNMARKED}$ if $\mathcal{L} = \{0\}$. Otherwise, it outputs $\mathcal{L}$.

Theorem 7.1. *If* PRF *is a secure puncturable PRF with weak key-injectivity and constrained one-wayness,* PFE *is a secure puncturable functional encryption scheme as defined in Sec. 5.1,* F *is a secure prefix puncturable PRF,* G, G' *and* $\tilde{\mathsf{G}}$ *are pseudorandom generators,* $\mathcal{H}$ *is a family of collision-resistant hash function, and* iO *is a secure indistinguishability obfuscator, then* WM *is a secure watermarking scheme with collusion resistant unremovability and* δ*-unforgeability for* PKE.

Proof. Proof of Theorem 7.1 can be proceeded similiarly as the proof of Theorem 6.1, so we omit its details here.

One subtle issue in the proof is that the adversary can additionally obtain a public key from the challenge oracle, which is an obfuscated circuit containing the challenge key k^*. So, we need to further argue that the public key will not leak additional information of k^*. Recall that through the whole security proof, either k^* or its equivalent version or its contrained version punctured on a random point will appear in the view of the adversary. In the first case, the public key will not provide additional information about k^*. In the second case, k^* can be replaced with its equivalent version in the public key and due to the indistinguishability of iO, this cannot be detected by the adversary. In the third case, k^* can be replaced with its contrained version in the public key. Since the probability that the random punctrued point falls in the range of $\tilde{\mathsf{G}}$ is negligible, by the indistinguishability of iO, this will also not affect the adversary's advantage. □

Remark 7.1. We remark that the above strategy is not fully applicable in the watermarkable signature setting. This is because in the verification key of the

```
Ext
Constant: $mpk, K$
Input: $a_1, \dots, a_d, b, ind, r$

  1. $t_1 = \mathsf{G}'(a_1), \dots, t_d = \mathsf{G}'(a_d)$.
  2. $x = \mathsf{PFE}.\mathtt{Enc}(mpk, t_1 \| \dots \| t_d \| b \| ind; r)$.
  3. $k' = \mathsf{F}.\mathtt{Eval}(K, t_1 \| \dots \| t_d \| b)$.
  4. $y = \mathsf{PRF}.\mathtt{Eval}(k', x)$.
  5. Output $(x, y)$.
```

```
Trace
Input: $ind_1, ind_2, p_1, p_2, \epsilon, T, \mathtt{E}, H, \mathtt{C}$
  1. $\Delta = |p_1 - p_2|$.
  2. If $\Delta \le \epsilon$: return $\emptyset$.
  3. If $ind_2 - ind_1 = 1$: return $\{ind_1\}$.
  4. $ind_3 = \lfloor \frac{ind_1 + ind_2}{2} \rfloor$.
  5. $p_3 = \mathtt{Test}(ind_3, T, \mathtt{E}, H, \mathtt{C})$.
  6. Return $\mathtt{Trace}(ind_1, ind_3, p_1, p_3, \epsilon, T, \mathtt{E}, H, \mathtt{C}) \cup \mathtt{Trace}(ind_3, ind_2, p_3, p_2, \epsilon, T, \mathtt{E}, H, \mathtt{C})$.
```

```
M
Constant: $sk, K, H, k, msg$
Input: $ct = (x, z)$

  1. $(t_1 \| \dots \| t_d \| b \| ind) = \mathsf{PFE}.\mathtt{Dec}(sk, x)$.
  2. If $(t_1 \| \dots \| t_d \| b \| ind \ne \perp) \wedge (ind \le msg) \wedge (H(\mathsf{PRF}.\mathtt{Eval}(k, \mathsf{G}(t_1)), \dots, \mathsf{PRF}.\mathtt{Eval}(k, \mathsf{G}(t_d))) = b)$
     (a) $k' = \mathsf{F}.\mathtt{Eval}(K, t_1 \| \dots \| t_d \| b)$.
     (b) $y = \mathsf{PRF}.\mathtt{Eval}(k', x)$.
     (c) Output $y \oplus z$.
  3. Otherwise, output $\mathsf{PRF}.\mathtt{Eval}(k, x) \oplus z$.
```

```
Test
Input: $ind, T, \mathtt{E}, H, \mathtt{C}$
  1. $Acc = 0$
  2. For $i \in [1, T]$:
     (a) Sample $a_1, \dots, a_d \xleftarrow{\$} \{0,1\}^{\frac{l}{2}}$ and $r \xleftarrow{\$} \mathcal{R}$.
     (b) Sample $z_1, \dots, z_d, z^* \xleftarrow{\$} \{0,1\}^m$.
     (c) $t_1 = \mathsf{G}'(a_1), \dots, t_d = \mathsf{G}'(a_d)$.
     (d) $b = H(\mathtt{C}(\mathsf{G}(t_1), z_1) \oplus z_1, \dots, \mathtt{C}(\mathsf{G}(t_d), z_d) \oplus z_d)$.
     (e) $(x, y) = \mathtt{E}(a_1, \dots, a_d, b, ind, r)$.
     (f) If $\mathtt{C}(x, z^*) \oplus z^* = y$: $Acc = Acc + 1$.
  3. Return $\frac{Acc}{T}$.
```

Fig. 3 The circuit Ext, the circuit M, the function Trace, and the function Test for the watermarkable PKE scheme.

signature scheme constructed in [SW14], the pseudorandom random function will compute on all points in its domain (rather than points in the range of a pseudorandom generator), thus, we cannot argue indistinguishability between a verification key generated from a normal key and that generated from a constrained key. To circumvent this problem, we modify the construction of signature scheme slightly and use a watermarked PRF key in the obfuscated circuit of the verification key. But this will lead to a weaker watermarkable signature scheme, which needs the marking key of the watermarking scheme when generating a signing key/verification key pair of the signature scheme.

8 Conclusion and Future Works

In this work, we initiate the study of collusion resistant watermarking by defining and constructing collusion resistant watermarking schemes for common cryptographic functionalities, including PRF, PKE, and signature.

One may note that watermarking schemes constructed in this work only achieve a *negl*$(\cdot)$-unremovability, which guarantees that no attacker can remove or modify the embedded message in a watermarked program via altering the program on a *negligible* fraction of inputs. A stronger form of unremovability, which is called ϵ-unremovability, considers attackers that can alter the watermarked program on a ϵ fraction of inputs for some non-negligible ϵ. In this setting, since the attacker is able to reset the outputs on a non-negligible fraction of inputs, internal variables generated during the extraction procedure may significantly depart from what is expected. In previous works with ϵ-unremovability (e.g., [CHN+16, QWZ18, KW19]), this issue is tackled by repeating some subprocedure multiple times and deciding based on majority. Unfortunately, in our construction, as the extraction algorithm needs to analyze the fraction of reprogrammed points in a set, it seems implausible to use the "repeating-and-choosing-majority" trick. How to construct collusion resistant watermarking schemes with ϵ-unremovability for non-negligible ϵ is an interesting open problem.

Another interesting direction is to explore the possibility of instantiating a collusion resistant watermarkable PRF from standard assumptions. As discussed in Sec. 1.1, a collusion resistant watermarkable PRF can be approximately viewed as a collusion resistant constraint-hiding constrained PRF, which can imply indistinguishability obfuscator. However, we have not provided a formal reduction. It is interesting to formally construct an indistinguishability obfuscator from a collusion resistant watermarkable PRF or construct a collusion resistant watermarkable PRF from standard assumptions.

Besides, it is also interesting to construct collusion resistant watermarking schemes with other desirable features, e.g., constructing collusion resistant watermarking schemes with public marking.

Acknowledgement. We appreciate the anonymous reviewers for their valuable suggestions. Part of this work was supported by the National Natural Science Foundation of China (Grant No. 61572294, 61602396, 61632020, U1636205), Early Career Scheme research grant (ECS Grant No. 25206317) from the Research Grant Council of Hong Kong, the Innovation and Technology Support Programme of Innovation and Technology Fund of Hong Kong (Grant No. ITS/356/17), and the MonashU-PolyU-Collinstar Capital Joint Lab on Blockchain. Junzuo Lai was supported by National Natural Science Foundation of China (Grant No. 61922036, 61572235), and Guangdong Natural Science Funds for Distinguished Young Scholar (No. 2015A030306045).

References

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, pages 52–73. Springer, 2014.

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *CRYPTO*, pages 1–18. Springer, 2001.

[BKS17] Foteini Baldimtsi, Aggelos Kiayias, and Katerina Samari. Watermarking public-key cryptographic functionalities and implementations. In *ISC*, pages 173–191. Springer, 2017.

[BLW17] Dan Boneh, Kevin Lewi, and David J Wu. Constraining pseudorandom functions privately. In *PKC*, pages 494–524. Springer, 2017.

[BN08] Dan Boneh and Moni Naor. Traitor tracing with constant size ciphertext. In *CCS*, pages 501–510. ACM, 2008.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, volume 4004, pages 573–592. Springer, 2006.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, pages 171–190. IEEE, 2015.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, pages 480–499. Springer, 2014.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In *EUROCRYPT*, pages 446–476. Springer, 2017.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, pages 257–270. Springer, 1994.

[CHN$^+$16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *STOC*, pages 1115–1127, 2016.

[CHV15] Aloni Cohen, Justin Holmgren, and Vinod Vaikuntanathan. Publicly verifiable software watermarking. Cryptology ePrint Archive, Report 2015/373, 2015. `https://eprint.iacr.org/2015/373`.

[CMB$^+$07] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Digital watermarking and steganography.* Morgan Kaufmann, 2007.

[CVW$^+$18] Yilei Chen, Vinod Vaikuntanathan, Brent Waters, Hoeteck Wee, and Daniel Wichs. Traitor-tracing from LWE made simple and attribute-based. In *TCC*, 2018.

[GKM$^+$19] Rishab Goyal, Sam Kim, Nathan Manohar, Brent Waters, and David J Wu. Watermarking public-key cryptographic primitives. In *CRYPTO*, pages 367–398. Springer, 2019.

[GKW18] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In *STOC*, 2018.

[Goe15] Michel Goemans. Lecture notes on Chernoff bounds. `http://math.mit.edu/~goemans/18310S15/chernoff-notes.pdf`, February 2015.

[HMW07] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. *TCC*, pages 362–382, 2007.

[KNT18] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Obfustopia built on secret-key functional encryption. In *EUROCRYPT*, pages 603–648. Springer, 2018.

[KW17] Sam Kim and David J Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In *CRYPTO*. Springer, 2017.

[KW19] Sam Kim and David J. Wu. Watermarking PRFs from lattices: Stronger security via extractable PRFs. In *CRYPTO*, pages 335–366. Springer, 2019.

[Nis13] Ryo Nishimaki. How to watermark cryptographic functions. In *EUROCRYPT*, pages 111–125. Springer, 2013.

[NSS99] David Naccache, Adi Shamir, and Julien P Stern. How to copyright a function? In *PKC*, pages 188–196. Springer, 1999.

[NW15] Ryo Nishimaki and Daniel Wichs. Watermarking cryptographic programs against arbitrary removal strategies. Cryptology ePrint Archive, Report 2015/344, 2015. `https://eprint.iacr.org/2015/344`.

[NWZ16] Ryo Nishimaki, Daniel Wichs, and Mark Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In *EUROCRYPT*, pages 388–419. Springer, 2016.

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming PRFs, the LWE way. In *PKC*. Springer, 2018.

[QWZ18] Willy Quach, Daniel Wichs, and Giorgos Zirdelis. Watermarking PRFs under standard assumptions: Public marking and security with extraction queries. In *TCC*, 2018.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484. ACM, 2014.

[YAL$^+$18] Rupeng Yang, Man Ho Au, Junzuo Lai, Qiuliang Xu, and Zuoxia Yu. Unforgeable watermarking schemes with public extraction. In *SCN*, pages 63–80. Springer, 2018.

[YF11] Maki Yoshida and Toru Fujiwara. Toward digital watermarking for cryptographic data. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 94(1):270–272, 2011.