How to Watermark Cryptographic Functions *

Ryo Nishimaki
NTT
Secure Platform Laboratories
nishimaki.ryo@lab.ntt.co.jp

Abstract

We introduce a notion of watermarking for cryptographic functions and propose a concrete scheme for watermarking cryptographic functions. Informally speaking, a digital watermarking scheme for cryptographic functions embeds information, called a *mark*, into functions such as one-way functions and decryption functions of public-key encryption. There are two basic requirements for watermarking schemes. (1) A mark-embedded function must be functionally equivalent to the original function. (2) It must be difficult for adversaries to remove the embedded mark without damaging the original functionality. In spite of its importance and usefulness, there have only been a few theoretical works on watermarking for functions (or programs). Furthermore, we do not have rigorous and meaningful definitions of watermarking for cryptographic functions and concrete constructions.

To solve the above problem, we introduce a notion of watermarking for cryptographic functions and define its security. Furthermore, we present a lossy trapdoor function (LTF) based on the decisional linear (DLIN) problem and a watermarking scheme for the LTF. Our watermarking scheme is secure under the DLIN assumption in the standard model. We use techniques of dual system encryption and dual pairing vector spaces (DPVS) to construct our watermarking scheme. This is a new application of DPVS.

Keywords: digital watermarking, dual pairing vector space, dual system encryption, vector decomposition problem, copyrighting functions,

1 Introduction

1.1 Background

Digital watermarking is a technology that enables us to embed information, called a "mark", into digital objects such as images, movies, and audio files. Such marks should be detected by using some procedure. There are two main properties of digital watermarking. The first is that the appearance (or functionality) of marked objects is almost the same as that of the original objects

*An extended abstract of this paper appeared in Eurocrypt 2013, the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, LNCS 7881, pages 111-125. This is the full version.

The second is that removing embedded marks without destroying the object is difficult. A main application of watermarking is protecting copyright. That is, we can prevent unauthorized copying of digital content by detecting watermarks. Another application is tracing and identifying owners of digital content. For example, if we find a potentially guilty user and illegally copied digital content, we can detect a watermark and identify the owner who distributed the illegal copy.

Most watermarking methods have been designed for perceptual objects, such as images. Only a few studies have focused on watermarking for non-perceptual objects (e.g., software, program). Software is quite common digital content and can be easily copied. Software piracy is a serious problem today. If illegally copied software is distributed, profits of software companies decrease. Watermarking for programs is one of tools to solve the problem and has very useful, attractive, and practical applications. However, they are little understood. We briefly explain related studies on program watermarking below.

Naccache, Shamir, and Stern introduced the notion of copyrighted functions and proposed a method for tracking different copies of functionally equivalent algorithms containing "marks" [NSS99]. A copyrighted function is drawn from a keyed function family. It guarantees that no adversary can output a functionally equivalent function with a new key even if many keyed functions are given. This is related to watermarking schemes for program (functions), but their security definition is a bit weak and not sufficient for program watermarking because copyrighted functions do not guarantee that embedded marks are not removed.

Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang considered the notion of software watermarking (program watermarking) from a cryptographic point of view in their seminal work [BGI+12]. They proposed a formalization of software watermarking and its security definition. However, the definition is too strong since it is simulation-based security. They gave an impossibility result for general-purpose program watermarking by using impossibility results of general-purpose program obfuscation [BGI+12]. "General-purpose" means that program watermarking/obfuscation can be applied to *any* program. Their security requirements cannot be achieved, so they leave positive theoretical results about watermarking (achieving concrete constructions for specific function families by using a game-based security definition) as an open problem.

Yoshida and Fujiwara introduced the notion of *watermarking for cryptographic data* and a concrete scheme for signatures [YF11]. Their idea is very exciting, but they did not propose a formal security definition of watermarking for cryptographic data and their scheme is not provably secure. They claim that the security of their scheme is based on the *vector decomposition (VD)* problem, which was introduced by Yoshida, Mitsunari, and Fujiwara [YMF10], but their proof is heuristic, that is, they did not show a reduction.

Hopper, Molnar, and Wagner proposed a rigorous complexity-theoretic (game-based) definition of security for watermarking schemes. Their definition seems to be very useful, but they focused on watermarking for only *perceptual* objects [HMW07]. They gave no concrete construction that satisfies their security definition.

1.2 Motivations and Applications

As explained in the previous section, there is no watermarking scheme for (cryptographic) functions [1] that is provably secure in a complexity-theoretic definition of security. Copyrighted functions by Naccache et al. are provably secure based on the factoring assumption, but their definition of security is weaker than that of watermarking, and their construction can embed a *bounded* number of marks [NSS99]. Before we introduce our contribution, we present several applications of watermarking to explain motivations.

Traceable cryptographic primitives. One application of watermarking for cryptographic functions (we often call it cryptographic watermarking) is constructing various *traceable cryptographic primitives*. If we have a watermarking scheme for cryptographic functions, for example, trapdoor one-way functions, collision-resistant hash functions (CRHF), and decryption functions, we can construct a variety of traceable primitives or copyrighted cryptographic primitives since private-key encryption, public-key encryption (PKE), digital signatures, and so on are constructed from trapdoor one-way functions and often use CRHFs in their algorithms.

As pointed out by Naccache et al., watermarked functions have the following applications [NSS99]:

- We can produce software or program that generates ciphertexts of the Feistel cipher based on a one-way function [LR88], signatures of Rompel's signature scheme [Rom90], or decrypted values of ciphertexts under PKE schemes based on a trapdoor one-way function. If a malicious user illegally generate copies of such software and distributes them, then a company that sold the software can trace them and identify the guilty users.

- A company can sell MAC-functions based on watermarked one-way functions to users for a log-in system on the Internet. The company records user IDs and marked functions in a database. If the users use them, they can log-in a member web site without revealing their identity since all marked functions are functionally equivalent. However, if a malicious user distributes an illegal copy and it is discovered, then the company can identify the guilty user identity by detecting an embedded mark.

Black-box traitor tracing. Kiayias and Yung proposed a method of constructing black-box traitors tracing schemes from copyrighted PKE functions [KY02]. When we broadcast digital content to a set of legitimate subscribers, we can use broadcast encryption schemes. If some of the subscribers leak partial information about their decryption keys to a pirate, who is a malicious user in broadcast encryption systems, then the pirate may be able to construct a pirate-decoder. That is, the pirate may access to the content though s/he is not a subscriber. Traitor tracing enables us to identify such malicious subscribers called traitor [CFN94]. Our cryptographic watermarking scheme can be seen as a generalized notion of copyrighted functions and our construction is based on identity-based encryption (IBE) schemes whose private keys for identities are marked (these are

[1] We consider functions as a kind of program.

copyrighted decryption functions of PKE), so our construction technique can be used to construct *black-box traitor tracing* schemes and it has a quite powerful application.

Theoretical treatment of watermarking. There are many heuristic methods for software watermarking [CT02], but there have only been a few studies that theoretically and rigorously treat the problem in spite of its importance. Functions can be seen as a kind of software (and program) and a large amount of software uses cryptographic functions, especially in a broadcast system, users must use software with decryption functions to view content. We believe that our scheme is an important step toward constructing practical software watermarking.

1.3 Our Contributions and Construction Ideas

To solve problems explained in Background section, we introduce the notion of watermarking for cryptographic functions, a game-based security definition of them, and a concrete construction. Our watermarking scheme is provably secure under the decisional linear (DLIN) assumption. To the best of our knowledge, this is the first provably secure watermarking scheme for functions (program) in terms of theoretical cryptography and we solved the open problem proposed by Barak *et al.* [BGI+12].

Our security notion is based on the notion of strong watermarking introduced by Hopper et al. [HMW07], but details are different since we focus on the definition for cryptographic functions. Their definition takes into account only perceptual objects and they modeled the notion of similarity by a perceptual metric space on objects that measures the distance between objects. Therefore, to construct watermarking schemes for cryptographic functions, we should modify their definition. We define the similarity by preserving functionality. That is, if a marked function is functionally equivalent to an original function, then we say that the marked function is similar to the original function. Watermarking schemes should guarantee that no adversary can generate a function which is functionally equivalent to a marked function but unmarked. That is, no adversary can remove embedded marks without destroying functions.

We propose a watermarking scheme for lossy trapdoor functions (LTFs) [PW11]. LTFs are quite powerful cryptographic functions. They imply standard trapdoor one-way functions, oblivious transfers, CRHFs, and secure PKE schemes against adaptive chosen ciphertext attacks (CCA) [PW11]. The watermarking scheme consists of four algorithms, key generation, mark, detect, and remove algorithms. Marked function indices are functionally equivalent to the original ones, that is, for any input, outputs of marked functions are the same as those of the original function. The construction can be used to construct an IBE scheme that can generate marked private keys for identities and marked signatures since our LTFs are based on IBE schemes, as explained in the next paragraph. That is, we can construct decryption algorithms in which watermarks can be embedded.

Key Techniques and Ideas Behind Our Construction. Our construction is based on the dual pairing vector space (DPVS) proposed by Okamoto and Takashima [OT09, OT10, OT12]. We

use the IBE scheme of Okamoto and Takashima [OT12] (which is a special case of their inner-product predicate encryption (IPE) scheme) and that of Lewko [Lew12] to construct LTFs. Loosely speaking, LTFs are constructed from homomorphic encryption schemes, and the IBE schemes of Okamoto-Takashima and Lewko are homomorphic. There are many other homomorphic encryption schemes but we selected Okamoto-Takashima and Lewko IBE schemes because they are constructed by DPVS and the dual system encryption methodology introduced by Waters [Wat09]. The methodology is a key technique to achieve a watermarking scheme.

First, we explain how we use the dual system encryption methodology to construct watermarking schemes. We apply the dual system encryption technique to not only security proofs but also *constructions of cryptographic primitives*. In the dual system encryption, there are two types for ciphertexts and private-keys respectively. The first type is called *normal* ciphertexts/keys and the second type is called *semi-functional* ciphertexts/keys. They have the following properties. Semi-functional ciphertexts can be decrypted using normal keys and normal ciphertext can be decrypted using semi-functional keys. However, semi-functional ciphertexts cannot be decrypted using semi-functional keys. Normal ciphertexts/keys are computationally indistinguishable from semi-functional ciphertexts/keys. In most cases, function indices of LTFs consist of *ciphertexts of homomorphic encryption* [FGK$^+$10, HO12, PW11], so, intuitively speaking, if we can construct a function index by using not only (normal) ciphertexts but also semi-functional keys, then the function index is functionally equivalent to a function index generated by (normal ciphertexts and) normal keys as long as normal ciphertexts are used. Moreover, if we use semi-functional ciphertexts, we can determine whether a function index is generated by semi-functional keys or not since semi-functional ciphertexts cannot be decrypted using semi-functional key. Thus, a function index that consists of semi-functional keys can be seen as a marked index and semi-functional ciphertexts can be used in a detection algorithm of a watermarking scheme. This is the main idea. Note that our construction technique can be used to construct an IBE scheme whose private keys can be marked because our LTFs are based on such an IBE scheme.

Next, we explain how we construct watermarking scheme by using DPVS. DPVS is linear space defined over bilinear groups and a vector consists of group elements [OT09, OT10]. One of key features of DPVS is that if we conceal (i.e., do not publish) some basis of a subspace then we can set a hidden linear subspace. A pair of dual orthonormal bases over groups are denoted by $\mathbb{B}$ and $\mathbb{B}^*$. They are generated by a random linear transformation matrix that consists of elements in a finite field. We use a hidden linear subspace spanned by a subset of $\mathbb{B}/\mathbb{B}^*$ for semi-functional ciphertexts/keys as Okamoto-Takashima and Lewko IBE schemes [Lew12, OT10, OT12]. We denote the subset by $\widehat{\mathbb{B}} \subset \mathbb{B}$, $\widehat{\mathbb{B}}^* \subset \mathbb{B}^*$, respectively. A hidden linear subspace for semi-functional ciphertexts and keys can be used as a detect key and a mark key of our watermarking scheme, respectively. Thus, we can embed "marks" into the hidden linear subspace and they are indistinguishable from non-marked objects because the decisional subspace problem is believed to be hard [OT08, OT10]. Informally speaking, the decisional subspace problem is determining whether a given vector is spanned by $\mathbb{B}$ (resp, $\mathbb{B}^*$) or $\mathbb{B} \setminus \widehat{\mathbb{B}}$ (resp, $\mathbb{B}^* \setminus \widehat{\mathbb{B}}^*$).

Okamoto and Takashima introduced complexity problems based on the DLIN problem to prove the security of their scheme [OT10, OT12] and these problems are deeply related to the VD prob-

lem [YMF10] and the decisional subspace problem. The VD problem says that it is difficult to decompose a vector in DPVS into a vector spanned by bases of a subspace. Lewko also introduced the subspace assumption [Lew12], which is implied by the DLIN assumption and highly related to the decisional subspace assumption introduced by Okamoto and Takashima [OT08] and the VD problem. All assumptions introduced by Okamoto-Takashima [OT10, OT12] and Lewko [Lew12] are implied by the standard DLIN assumption.

If we can decompose a vector in DPVS into each linearly independent vector, then we can convert semi-functional ciphertexts/keys into normal ciphertexts/keys by eliminating elements in hidden linear subspaces, that is, we can remove an embedded mark from a marked function index. Galbraith and Verheul and Yoshida, Mitsunari, and Fujiwara argued that the VD problem is related to computational Diffie-Hellman problem [GV08, YMF10]. It is believed that the VD problem is hard. Therefore, no adversary can remove marks of our watermarking scheme (this is a just intuition). However, we do not directly use the VD problem but the DLIN problem to prove the security of our scheme. On the other hand, if we have a linear transformation matrix behind dual orthonormal bases of DPVS, then we can easily solve the VD problem [OT08, OT10], that is, we can remove a mark if we have the matrix. Such an algorithm was proposed by Okamoto and Takashima [OT08].

Our construction is a new application of DPVS. DPVS has been used to construct fully secure functional encryption, IPE, IBE and attribute-based signatures [Lew12, LOS$^+$10, OT09, OT10, OT11, OT12], but to the best of our knowledge, a linear transformation matrix for dual orthonormal bases in DPVS has never been explicitly used for algorithms of cryptographic schemes. This is of independent interest.

1.4 Organization of This Paper

In Section 2, we introduce some notations and known cryptographic definitions, tools, and techniques. In Section 3, we introduce our definition of watermarking for cryptographic functions. In Section 4 and 5, we propose the first and second concrete instantiations of watermarking schemes for lossy trapdoor functions, respectively. In Section 6, we list a few concluding remarks and open issues.

2 Preliminaries

Notations and Conventions. For any $n \in \mathbb{N} \setminus \{0\}$, let $[n]$ be the set $\{1, \ldots, n\}$. When D is a random variable or distribution, $y \xleftarrow{\mathsf{R}} D$ denote that y is randomly selected from D according to its distribution. If S is a set, then $x \xleftarrow{\mathsf{U}} S$ denotes that x is uniformly selected from S. $y := z$ denotes that y is set, defined or substituted by z. When b is a fixed value, $A(x) \to b$ (e.g., $A(x) \to 1$) denotes the event that machine (or algorithm) A outputs a on input x. We say that function $f : \mathbb{N} \to \mathbb{R}$ is negligible in $\lambda \in \mathbb{N}$ if $f(\lambda) = \lambda^{-\omega(1)}$. Hereafter, we use $f < \mathsf{negl}(\lambda)$ to mean that f is negligible in λ. We denote the finite field of order q by $\mathbb{F}_q$, and $\mathbb{F}_q \setminus \{0\}$ by $\mathbb{F}_q^\times$. A

vector symbol denotes a vector representation over $\mathbb{F}_q$, e.g., $\vec{x}$ denotes $(x_1, \ldots, x_n) \in \mathbb{F}_q^n$. For two vectors $\vec{x}$ and $\vec{v}$, $\langle \vec{x}, \vec{v} \rangle$ denotes the inner-product $\sum_{i=1}^{n} x_i v_i$. Matrix $\boldsymbol{X}^\top$ denotes the transpose of matrix $\boldsymbol{X}$. A bold face small letter denotes an element of vector space $\mathbb{V}$, e.g., $\boldsymbol{x} \in \mathbb{V}$. For bases $\mathbb{B} := (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N)$ and $\mathbb{B}^* := (\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_N^*)$ where $\boldsymbol{b}_i, \boldsymbol{b}_j^* \in \mathbb{V}$, $(x_1, \ldots x_N)_{\mathbb{B}} := \sum_{i=1}^{N} x_i \boldsymbol{b}_i$ and $(y_1, \ldots y_N)_{\mathbb{B}^*} := \sum_{i=1}^{N} y_i \boldsymbol{b}_i^*$. Set $GL(n, \mathbb{F}_q)$ denotes the general linear group of degree n over $\mathbb{F}_q$. If we use notation g/G to denote a generator in $\mathbb{G}$, then we use multiplicative/additive notation. We denote probabilistic polynomial-time by PPT.

Let $\mathcal{X} = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ denote two ensembles of random variables indexed by $\lambda \in \mathbb{N}$. The statistical distance between two random variables X and Y over a countable set S is defined as $\Delta(X, Y) := \frac{1}{2} \sum_{\alpha \in S} |\Pr[X = \alpha] - \Pr[Y = \alpha]|$.

Definition 2.1 We say that $\mathcal{X}$ and $\mathcal{Y}$ are statistically indistinguishable (We write $\mathcal{X} \overset{\mathsf{s}}{\approx} \mathcal{Y}$ to denote this) if

$$\Delta(X_\lambda, Y_\lambda) < \mathsf{negl}(\lambda).$$

Definition 2.2 We say that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable (We write $\mathcal{X} \overset{\mathsf{c}}{\approx} \mathcal{Y}$ to denote this) if for all non-uniform PPT algorithm D,

$$|\Pr[D(X_\lambda) = 1] - \Pr[D(Y_\lambda) = 1]| < \mathsf{negl}(\lambda).$$

2.1 Cryptographic Bilinear Maps (or Pairings)

We consider cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ of prime order q. A bilinear map is an efficient mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfying the following properties.

bilinearity: For all $g \in \mathbb{G}_1$, $\hat{g} \in \mathbb{G}_2$ and $a, b \xleftarrow{\mathsf{U}} \mathbb{F}_q$, $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab}$.

non-degeneracy: If $g/\hat{g}$ generates $\mathbb{G}_1/\mathbb{G}_2$, then $e(g, \hat{g}) \neq 1$.

If $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$, that is, both groups are the same, we call $(q, \mathbb{G}, \mathbb{G}_T, e, g)$ symmetric pairing groups. Let $\mathcal{G}_{\mathsf{bmp}}$ be a standard parameter generation algorithm that takes as input security parameter λ and outputs parameters $(q, \mathbb{G}, \mathbb{G}_T, e, g)$.

2.2 Function Family of Lossy Trapdoor Functions

Definition 2.3 (Lossy Trapdoor Functions [PW08, PW11]) A lossy trapdoor function LTF with domain D consists of four polynomial-time algorithms having the following properties.

Injective Key Generation: $\mathsf{LTF.IGen}$ outputs (ek, ik) where ek/ik is an evaluation/inversion key.

Evaluation: For $X \in \mathsf{D}$, $\mathsf{LTF.Eval}_{ek}(X)$ outputs an image $Y = f_{ek}(X)$.

Inversion: $\mathsf{LTF.Invert}_{ik}(Y)$ outputs a pre-image $X = f_{ik}^{-1}(Y)$.

Lossy Key Generation: $\mathsf{LTF.LGen}$ outputs $(ek', \perp)$ where ek' is an evaluation key.

Correctness: For all $(ek, ik) \xleftarrow{R} \mathsf{LTF.IGen}(1^\lambda)$, and $X \in \mathsf{D}$, we have $f_{ik}^{-1}(f_{ek}(X)) = X$.

Indistinguishability: Let λ be a security parameter. For all PPT $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{ltf},\mathcal{A}}(\lambda) := \left|\Pr[\mathcal{A}(1^\lambda, [\mathsf{LTF.IGen}(1^\lambda)]_1)] - \Pr[\mathcal{A}(1^\lambda, [\mathsf{LTF.LGen}(1^\lambda)]_1)]\right| < \mathsf{negl}(\lambda),$$

where $[A]_1$ is the first output of algorithm A.

Lossiness: We say that LTF is ℓ-lossy if for all $ek' \xleftarrow{R} \mathsf{LTF.LGen}(1^\lambda)$, the image set $f_{ek'}(\mathsf{D})$ is of size at most $|\mathsf{D}|/2^\ell$.

We define a function family of LTF, $\mathsf{LTF}_\lambda := \{\mathsf{LTF.Eval}_{ek}(,\cdot)|(ek, ik) \xleftarrow{R} \mathsf{LTF.Gen}(1^\lambda, b), b \in \{0,1\}\}$ where $\mathsf{LTF.Gen}(1^\lambda, 0) := \mathsf{LTF.IGen}(1^\lambda)$ and $\mathsf{LTF.Gen}(1^\lambda, 1) := \mathsf{LTF.LGen}(1^\lambda)$.

2.3 Dual Pairing Vector Space [OT09, OT10]

Definition 2.4 *"Dual pairing vector spaces (DPVS)"* $(q, \mathbb{V}, \mathbb{G}_T, \mathbb{A}, e)$ by a direct product of symmetric pairing groups $(q, \mathbb{G}, \mathbb{G}_T, e, G)$ are a tuple of prime q, N-dimensional vector space $\mathbb{V} := \mathbb{G}^N$ over $\mathbb{F}_q$, cyclic group $\mathbb{G}_T$ of order q, canonical basis $\mathbb{A} := (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N)$ of $\mathbb{V}$, where $\boldsymbol{a}_i := (0, \ldots, 0, G, 0, \ldots, 0)$ (only the i-th coordinate is G), and pairing $\boldsymbol{e} : \mathbb{V} \times \mathbb{V} \to \mathbb{G}_T$. The pairing is defined as $\boldsymbol{e}(\boldsymbol{x}, \boldsymbol{y}) := \prod_{i=1}^N e(G_i, H_i) \in \mathbb{G}_T$ where $\boldsymbol{x} := (G_1, \ldots, G_N) \in \mathbb{V}$ and $\boldsymbol{y} := (H_1, \ldots, H_N) \in \mathbb{V}$. This is non-degenerate bilinear, i.e., $\boldsymbol{e}(s\boldsymbol{x}, t\boldsymbol{y}) = e(\boldsymbol{x}, \boldsymbol{y})^{st}$ and if $\boldsymbol{e}(\boldsymbol{x}, \boldsymbol{y}) = 1$ for all $\boldsymbol{y} \in \mathbb{V}$, then $\boldsymbol{x} = \boldsymbol{0}$. For all i and j, $\boldsymbol{e}(\boldsymbol{a}_i, \boldsymbol{a}_j) = e(G, G)^{\delta_{i,j}}$ where $\delta_{i,j} = 1$ if $i = j$, and 0 otherwise, and $e(G, G) \neq 1$.

DPVS also has linear transformations $\phi_{i,j}$ on $\mathbb{V}$ s.t. $\phi_{i,j}(\boldsymbol{a}_j) = \boldsymbol{a}_i$ and $\phi_{i,j}(\boldsymbol{a}_k) = \boldsymbol{0}$ if $k \neq j$, which can be easily achieved by $\phi_{i,j}(\boldsymbol{x}) := (\overbrace{0, \ldots, 0}^{i-1}, G_j, \overbrace{0, \ldots, 0}^{N-i})$ where $\boldsymbol{x} := (G_1, \ldots, G_N)$. We call $\phi_{i,j}$ canonical maps. DPVS generation algorithm $\mathcal{G}_{\mathsf{dpvs}}$ takes input 1^λ and $N \in \mathbb{N}$, and outputs a description of $\mathsf{param}'_{\mathbb{V}} := (q, \mathbb{V}, \mathbb{G}_T, \mathbb{A}, e)$ with security parameter λ and N-dimensional $\mathbb{V}$. It can be constructed using $\mathcal{G}_{\mathsf{bmp}}$.

We describe random dual orthonormal bases generator $\mathcal{G}_{\mathsf{ob}}(1^\lambda, N)$, which is used as a subroutine in the proposed scheme.

$$
\begin{aligned}
&\underline{\mathcal{G}_{\mathsf{ob}}(1^\lambda, N)} \\
&\mathsf{param}'_{\mathbb{V}} := (q, \mathbb{V}, \mathbb{G}_T, \mathbb{A}, e) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{dpvs}}(1^\lambda, N), \\
&\boldsymbol{X} := (\chi_{i,j}) \xleftarrow{\mathsf{U}} GL(N, \mathbb{F}_q),\ \psi \xleftarrow{\mathsf{U}} \mathbb{F}_q^\times, \\
&(\vartheta_{i,j}) := \psi(\boldsymbol{X}^\top)^{-1},\ g_T := e(G,G)^\psi,\ \mathsf{param}_{\mathbb{V}} := (\mathsf{param}'_{\mathbb{V}}, g_T), \\
&\boldsymbol{b}_i := \sum_{j=1}^{N} \chi_{i,j} \boldsymbol{a}_j, \mathbb{B} := (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N), \\
&\boldsymbol{b}^*_i := \sum_{j=1}^{N} \vartheta_{i,j} \boldsymbol{a}_j, \mathbb{B}^* := (\boldsymbol{b}^*_1, \ldots, \boldsymbol{b}^*_N), \\
&\text{return } (\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*).
\end{aligned}
$$

We briefly explain some important properties of DPVS [OT10].

Vector space $\mathbb{V}$: A vector space consists of N groups, i.e., $\mathbb{V} := \overbrace{\mathbb{G} \times \cdots \times \mathbb{G}}^{N}$, whose element is expressed by N-dimensional vector $\boldsymbol{x} := (x_1 G, \ldots, x_N G)$ ($x_i \in \mathbb{F}_q$ for $i = 1, \ldots, N$).

Canonical base $\mathbb{A}$: There is canonical basis $\mathbb{A} := (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N)$ of $\mathbb{V}$, where $\boldsymbol{a}_1 := (G, 0, \ldots, 0)$, $\boldsymbol{a}_2 := (0, G, 0, \ldots, 0), \ldots, \boldsymbol{a}_N := (0, \ldots, 0, G)$.

Pairing operation: $\boldsymbol{e}(\boldsymbol{x}, \boldsymbol{y}) := \prod_{i=1}^{N} e(x_i G, y_i G) = e(G,G)^{\sum_{i=1}^{N} x_i y_i} = e(G,G)^{\vec{x} \cdot \vec{y}} \in \mathbb{G}_T$, where $\boldsymbol{x} := (x_1 G, \ldots, x_N G)$ and $\boldsymbol{y} := (y_1 G, \ldots, y_N G)$. Here, $\boldsymbol{x}$ and $\boldsymbol{y}$ are expressed by coefficient vectors over basis $\mathbb{A}$ such that $(x_1, \ldots, x_N)_{\mathbb{A}} = (\vec{x})_{\mathbb{A}} := \boldsymbol{x}$ and $(y_1, \ldots, y_N)_{\mathbb{A}} = (\vec{y})_{\mathbb{A}} := \boldsymbol{y}$.

Base change: Canonical basis $\mathbb{A}$ is changed to *dual orthonormal bases* $\mathbb{B} := (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N)$ and $\mathbb{B}^* := (\boldsymbol{b}^*_1, \ldots, \boldsymbol{b}^*_N)$ of $\mathbb{V}$ using a uniformly chosen (regular) linear transformation, $\boldsymbol{X} := (\chi_{i,j}) \xleftarrow{\mathsf{U}} GL(N, \mathbb{F}_q)$, such that $\boldsymbol{b}_i = \sum_{j=1}^{N} \chi_{i,j} \boldsymbol{a}_j = (\chi_{i,1} G, \ldots, \chi_{i,N} G)$, $\boldsymbol{b}^*_i = \sum_{j=1}^{N} \vartheta_{i,j} \boldsymbol{a}_j = (\vartheta_{i,1} G, \ldots \vartheta_{i,N} G)$ ($i = 1, \ldots, N$), and $(\vartheta_{i,j}) := \psi(\boldsymbol{X}^\top)^{-1}$ where $\psi \xleftarrow{\mathsf{U}} \mathbb{F}_q^\times$. It holds that $\boldsymbol{e}(\boldsymbol{b}_i, \boldsymbol{b}^*_j) = e(G,G)^{\delta_{i,j}}$ ($\delta_{i,j} = 1$ if $i = j$, and $\delta_{i,j} = 0$ if $i \neq j$). Here, $\boldsymbol{x} := x_1 \boldsymbol{b}_1 + \cdots + x_N \boldsymbol{b}_N \in \mathbb{V}$ and $\boldsymbol{y} := y_1 \boldsymbol{b}^*_1 + \cdots y_N \boldsymbol{b}^*_N \in \mathbb{V}$ can be expressed by coefficient vectors over basis $\mathbb{B}$ and $\mathbb{B}^*$ such that $(x_1, \ldots, x_N)_{\mathbb{B}} = (\vec{x})_{\mathbb{B}} := \boldsymbol{x}$ and $(y_1, \ldots, y_N)_{\mathbb{B}^*} = (\vec{y})_{\mathbb{B}^*} := \boldsymbol{y}$, and $\boldsymbol{e}(\boldsymbol{x}, \boldsymbol{y}) = e(G,G)^{\sum_{i=1}^{N} x_i y_i} = e(G,G)^{\vec{x} \cdot \vec{y}} \in \mathbb{G}_T$.

Intractable problem: A decisional problem in this approach is the decisional subspace problem [OT08]. It is to tell $\boldsymbol{v} := v_{N_2+1} \boldsymbol{b}_{N_2+1} + \cdots v_{N_1} \boldsymbol{b}_{N_1} = (0, \ldots, 0, v_{N_2+1}, \ldots, v_{N_1})_{\mathbb{B}}$ from $\boldsymbol{u} := v_1 \boldsymbol{b}_1 + \cdots v_{N_1} \boldsymbol{b}_{N_1} = (v_1, \ldots, v_{N_1})_{\mathbb{B}}$, where $(v_1, \ldots v_{N_1}) \xleftarrow{\mathsf{U}} \mathbb{F}_q^{N_1}$ and $N_2 + 1 < N_1$.

Trapdoor: If we have trapdoor $\boldsymbol{t}^* \in \mathsf{span}\left\langle \boldsymbol{b}^*_1, \ldots, \boldsymbol{b}^*_{N_2} \right\rangle$, then we can efficiently solve the decisional subspace problem. Given $\boldsymbol{v} := v_{N_2+1}\boldsymbol{b}_{N_2+1} + \cdots v_{N_1}\boldsymbol{b}_{N_1}$ or $\boldsymbol{u} := v_1\boldsymbol{b}_1 + \cdots v_{N_1}\boldsymbol{b}_{N_1}$, we can tell $\boldsymbol{v}$ from $\boldsymbol{u}$ using $\boldsymbol{t}^*$ since $e(\boldsymbol{v}, \boldsymbol{t}^*) = 1$ and $e(\boldsymbol{u}, \boldsymbol{t}^*) \neq 1$ with high probability.

Advantage of this approach: It is easy to decompose $x_i\boldsymbol{a}_i = (0, \ldots, 0, x_iG, 0, \ldots, 0)$ from $\boldsymbol{x} := x_1\boldsymbol{a}_1 + \cdots + x_N\boldsymbol{a}_N = (x_1G, \ldots, x_NG)$. In contrast, DPVS approach employs basis $\mathbb{B}$, which is linearly transformed from $\mathbb{A}$ using a secret random matrix $\boldsymbol{X} \in \mathbb{F}_q^{n\times n}$. It seems hard to decompose $x_i\boldsymbol{b}_i$ from $\boldsymbol{x}' := x_1\boldsymbol{b}_1 + \cdots x_N\boldsymbol{b}_N$ (and the decisional subspace problem seems intractable). In addition, the secret matrix $\boldsymbol{X}$ (and the dual orthonormal basis $\mathbb{B}^*$ of $\mathbb{V}$) can be used as trapdoors for the decomposability (and distinguishability for the decisional subspace problem through the pairing operation over $\mathbb{B}$ and $\mathbb{B}^*$).

Vector decomposition problem. The VD problem was originally introduced by Yoshida, Mitsunari, and Fujiwara [YMF10]. We present the definition of a higher dimensional version by Okamoto and Takashima [OT08] to fit the VD problem into DPVS.

Definition 2.5 (CVDP: (ℓ_1, ℓ_2)-Computational Vector Decomposition Problem [OT08]) Let λ be a security parameter and $\mathcal{G}_{\mathsf{ob}}$ be an algorithm that outputs a description of a ℓ_1-dimensional DPVS $(q, \mathbb{V}, \mathbb{G}_T, \mathbb{A}, e)$ and $\ell_1 > \ell_2$. Let $\mathcal{A}$ be a PPT machine. For all $\lambda \in \mathbb{N}$, we define the $\mathsf{CVDP}_{(\ell_1,\ell_2)}$ advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{cvdp}}_{\mathcal{A},(\ell_1,\ell_2)}(\lambda) := \Pr\left[\omega = \sum_{i=1}^{\ell_2} x_i\boldsymbol{b}_i \;\middle|\; \begin{array}{l} (\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{R} \mathcal{G}_{\mathsf{ob}}(1^\lambda, \ell_1), \\ (x_1, \ldots, x_{\ell_1}) \xleftarrow{U} (\mathbb{F}_q)^{\ell_1}, \\ \boldsymbol{v} := \sum_{i=1}^{\ell_1} x_i\boldsymbol{b}_i, \omega \xleftarrow{R} \mathcal{A}(1^\lambda, \mathsf{param}_{\mathbb{V}}, \mathbb{B}, \boldsymbol{v}) \end{array}\right].$$

The $\mathsf{CVDP}_{(\ell_1,\ell_2)}$ assumption: For all PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{cvdp}}_{\mathcal{A},(\ell_1,\ell_2)}(\lambda) < \mathsf{negl}(\lambda)$.

A specific class of the CVDP instances that are specified over canonical basis $\mathbb{A}$ are tractable.

Lemma 2.6 (Easy Basis [OT08]) *Let $\mathbb{A}$ be a canonical basis of $\mathbb{V}$, and $\mathsf{CVDP}^{\mathbb{A}}_{(\ell_1,\ell_2)}$ be a specific class of $\mathsf{CVDP}_{(\ell_1,\ell_2)}$ in which $\mathbb{B}$ is replaced by $\mathbb{A}$. The canonical maps $\phi_{i,j}$ on $\mathbb{V}$ can solve $\mathsf{CVDP}^{\mathbb{A}}_{(\ell_1,\ell_2)}$ in polynomial time.*

Trapdoor. If we have a trapdoor, linear transformation matrix $\boldsymbol{X}$ behind $\mathbb{B}$, then we can efficiently decompose vectors in DPVS, i.e., solve $\mathsf{CVDP}_{(\ell_1,\ell_2)}$ by using the efficient algorithm Decomp given by Okamoto and Takashima [OT08]. The input is $(\boldsymbol{v}, (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\ell_2}), \boldsymbol{X}, \mathbb{B})$ such that $\boldsymbol{v} := \sum_{i=1}^{\ell_1} y_i\boldsymbol{b}_i$ is a target vector for decomposition, $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\ell_2})$ is a subspace to be decomposed into, $\boldsymbol{X}$ is a trapdoor (matrix), and $\mathbb{B} := (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\ell_1})$ is a basis generated by using $\boldsymbol{X}$. Algorithm $\mathsf{Decomp}(\boldsymbol{v}, (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\ell_2}), \boldsymbol{X}, \mathbb{B})$: computes $\boldsymbol{u} := \sum_{i=1}^{\ell_1}\sum_{j=1}^{\ell_2}\sum_{\kappa=1}^{\ell_1} \tau_{i,j}\chi_{j,\kappa}\phi_{\kappa,i}(\boldsymbol{v})$ where ϕ is the canonical map in Definition 2.4, $(\chi_{i,j}) = \boldsymbol{X}$ and $(\tau_{i,j}) := (\boldsymbol{X})^{-1}$.

Lemma 2.7 ([OT08]) *Algorithm* Decomp *solves* $\mathsf{CVDP}_{(\ell_1,\ell_2)}$ *by using* $\boldsymbol{X} := (\chi_{i,j})$ *such that* $\boldsymbol{b}_i := \sum_{j=1}^{\ell_1} \chi_{i,j}\boldsymbol{a}_j$.

Multiplicative Notation of Dual Pairing Vector Space by Lewko [Lew12]. We introduce a multiplicative notation by Lewko [Lew12]. For $\vec{v}, \vec{w} \in \mathbb{Z}_p^n$ (p is a prime), $a \in \mathbb{Z}_p$, and $g \in \mathbb{G}$, we define $g^{\vec{v}} := (g^{v_1}, \ldots, g^{v_n})$, $g^{a\vec{v}} := (g^{av_1}, \ldots, g^{av_n})$, $g^{\vec{v}+\vec{w}} := (g^{v_1+w_n}, \ldots, g^{v_n+w_n})$, and $\boldsymbol{e}(g^{\vec{v}}, g^{\vec{w}}) := \prod_{i=1}^n e(g^{v_i}, g^{w_i})$. Lewko defined algorithm $\mathsf{Dual}(\mathbb{Z}_p^n)$ as follows: It chooses $\vec{b}_i, \vec{b}_j^* \in \mathbb{Z}_p^n$ and $\psi \xleftarrow{\mathsf{U}} \mathbb{Z}_p$ such that $\langle \vec{b}_i, \vec{b}_j^* \rangle = 0 \bmod p$ for $i \neq j$, $\langle \vec{b}_i, \vec{b}_i^* \rangle = \psi \bmod p$ for all $i \in [n]$ and outputs $(\mathfrak{B}, \mathfrak{B}^*)$ where $\mathfrak{B} := (\vec{b}_1, \ldots, \vec{b}_n)$ and $\mathfrak{B}^* := (\vec{b}_1^*, \ldots, \vec{b}_n^*)$. We use the notation $(\mathfrak{B}, \mathfrak{B}^*)$ to express dual orthonormal bases in $\mathbb{Z}_p$ to distinguish from dual orthonormal bases $(\mathbb{B}, \mathbb{B}^*)$ in $\mathbb{V}$. We can consider $\boldsymbol{b}_i = g^{\vec{b}_i}$, $\boldsymbol{b}_j^* = g^{\vec{b}_j^*}$, $\vec{b}_i = (\chi_{i,1}, \ldots, \chi_{i,n})$, $\vec{b}_i^* = (\vartheta_{i,1}, \ldots, \vartheta_{i,n})$ where $\boldsymbol{X} = (\chi_{i,j})$ and $\psi(\boldsymbol{X}^{-1})^\top = (\vartheta_{i,j})$.

Let m be a fixed positive integer such that $m \leq n$, $A \in \mathbb{Z}_p^{m \times m}$ an invertible matrix, and $S_m \subseteq [n]$ a subset of size m. Lewko proposed how to obtain new dual orthonormal bases $(\mathfrak{B}_A, \mathfrak{B}_A^*)$ from $(\mathfrak{B}, \mathfrak{B}^*)$ by using A. If B_m is an $n \times m$ matrix over $\mathbb{Z}_p$ whose columns are vectors $\vec{b}_i \in \mathfrak{B}$ such that $i \in S_m$, then $B_m A$ is also an $n \times m$ matrix. Let $\mathfrak{B}_A := (\vec{a}_1, \ldots, \vec{a}_n)$ where $\vec{a}_i := \vec{b}_i$ for all $i \notin S_m$ and $\vec{a}_i := (B_m A)_\ell$ for $i \in S_m$, i is the ℓ-th element of S_m and $(B_m A)_\ell$ denotes the ℓ-th column of $B_m A$. If B_m^* is $n \times m$ matrix over $\mathbb{Z}_p$ whose columns are vectors $\vec{b}_i^* \in \mathfrak{B}^*$ such that $i \in S_m$, then $B_m(A^{-1})^\top$ is also $n \times m$ matrix. Let $\mathfrak{B}_A^* := (\vec{a}_1^*, \ldots, \vec{a}_n^*)$ where $\vec{a}_i^* := \vec{b}_i^*$ for all $i \notin S_m$ and $\vec{a}_i^* := (B_m^*(A^{-1})^\top)_\ell$ for $i \in S_m$, i is the ℓ-th element of S_m and $(B_m(A^{-1})^\top)_\ell$ denotes the ℓ-th column of $B_m(A^{-1})^\top$. Lewko showed that these newly generated bases are also dual orthonomal bases.

Lemma 2.8 ([Lew12]) *For any fixed positive integers* $m \leq n$*, any fixed invertible* $A \in \mathbb{Z}_p^{m \times m}$ *and set* $S_M \subseteq [n]$ *of size* m*, if* $(\mathfrak{B}, \mathfrak{B}^*) \xleftarrow{\mathsf{R}} \mathsf{Dual}(\mathbb{Z}_p^n)$*, then* $(\mathfrak{B}_A, \mathfrak{B}_A^*)$ *is also distributed as a random sample from* $\mathsf{Dual}(\mathbb{Z}_p^n)$ *and its distribution is independent of* A.

2.4 Complexity Assumptions

We write following assumptions by the multiplicative notation, but we can also use assumptions by the additive notation.

Definition 2.9 (DLIN Assumption) The DLIN problem is to guess $\beta \in \{0,1\}$, given $(\Gamma, g, f, h, f^\delta, h^\sigma, Q_\beta) \xleftarrow{\mathsf{R}} \mathcal{G}_\beta^{\mathsf{dlin}}(1^\lambda)$, where $\mathcal{G}_\beta^{\mathsf{dlin}}(1^\lambda)$: $\Gamma := (q, \mathbb{G}, \mathbb{G}_T, e, g) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{bmp}}(1^\lambda)$, $\xi, \kappa, \delta, \sigma \xleftarrow{\mathsf{U}} \mathbb{Z}_p$, $f := g^\xi, h := g^\kappa$, $Q_0 := g^{\delta+\sigma}$, $Q_1 \xleftarrow{\mathsf{U}} \mathbb{G}$, return $\mathcal{I} := (\Gamma, f, h, f^\delta, h^\sigma, Q_\beta)$. This advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{dlin}}(\lambda)$ is defined as follows.

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{dlin}}(\lambda) := \left| \Pr\left[\mathcal{A}(\mathcal{I}) \to 1 \,\middle|\, \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}_0^{\mathsf{dlin}}(1^\lambda)\right] - \Pr\left[\mathcal{A}(\mathcal{I}) \to 1 \,\middle|\, \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}_1^{\mathsf{dlin}}(1^\lambda)\right] \right|.$$

We say that the DLIN assumption holds if for all PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{dlin}}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

Definition 2.10 (Subspace Assumption) The subspace problem is to guess $\beta \in \{0,1\}$, given (Γ, D, Q_β), where $\mathcal{G}^{\mathsf{dss}}_\beta(1^\lambda)$: $\Gamma \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{bmp}}(1^\lambda)$, $(\mathfrak{B}, \mathfrak{B}^*) \xleftarrow{\mathsf{U}} \mathsf{Dual}(\mathbb{Z}_p^n)$, $\eta, \beta, \tau_1, \tau_2, \tau_3, \mu_1, \mu_2, \mu_3 \xleftarrow{\mathsf{U}} \mathbb{Z}_p$, for $i \in [k]$, $U_i := g^{\mu_1 \vec{b}_i + \mu_2 \vec{b}_{k+i} + \mu_3 \vec{b}_{2k+i}}$ and

$$V_i := g^{\tau_1 \eta \vec{b}^*_i + \tau_2 \beta \vec{b}^*_{k+i}} \qquad W_i := g^{\tau_1 \eta \vec{b}^*_i + \tau_2 \beta \vec{b}^*_{k+i} + \tau_3 \vec{b}^*_{2k+i}}$$
$$Q_0 := (V_1, \ldots, V_k) \qquad Q_1 := (W_1, \ldots, W_k)$$

$D := (g^{\vec{b}_1}, \ldots, g^{\vec{b}_{2k}}, g^{\vec{b}_{3k+1}}, \ldots, g^{\vec{b}_n}, g^{\eta \vec{b}^*_1}, \ldots, g^{\eta \vec{b}^*_k}, g^{\beta \vec{b}^*_{k+1}}, \ldots, g^{\beta \vec{b}^*_{2k}}, g^{\vec{b}^*_{2k+1}}, \ldots, g^{\vec{b}^*_n}, U_1, \ldots, U_k, \mu_3)$, return $\mathcal{I} := (\Gamma, D, Q_\beta)$. This advantage $\mathsf{Adv}^{\mathsf{dss}}_{\mathcal{A}}(\lambda)$ is defined as follows.

$$\mathsf{Adv}^{\mathsf{dss}}_{\mathcal{A}}(\lambda) := \left| \Pr\left[\mathcal{A}(\mathcal{I}) \to 1 \,\middle|\, \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}^{\mathsf{dss}}_0(1^\lambda)\right] - \Pr\left[\mathcal{A}(\mathcal{I}) \to 1 \,\middle|\, \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}^{\mathsf{dss}}_1(1^\lambda)\right] \right|.$$

We say that the subspace assumption holds if for all PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{dss}}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

Theorem 2.11 ([Lew12]) *The DLIN assumption implies the subspace assumption.*

Definition 2.12 (DBDH assumption) The DBDH problem is to guess $\beta \in \{0,1\}$, given $(\Gamma, g, g^a, g^b, g^c, Q_\beta) \xleftarrow{\mathsf{R}} \mathcal{G}^{\mathsf{dbdh}}_\beta(1^\lambda)$, where $\mathcal{G}^{\mathsf{dbdh}}_\beta(1^\lambda)$: $\Gamma := (q, \mathbb{G}, \mathbb{G}_T, e, g) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{bmp}}(1^\lambda)$, $a, b, c \xleftarrow{\mathsf{U}} \mathbb{Z}_p$, $Q_0 := e(g,g)^{abc}$, $Q_1 \xleftarrow{\mathsf{U}} \mathbb{G}_T$, return $(\Gamma, g, g^a, g^b, g^c, Q_\beta)$. This advantage $\mathsf{Adv}^{\mathsf{dbdh}}_{\mathcal{A}}(\lambda)$ is defined as follows.

$$\mathsf{Adv}^{\mathsf{dbdh}}_{\mathcal{A}}(\lambda) := \left| \Pr\left[\mathcal{A}(\mathcal{I}) \to 1 \,\middle|\, \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}^{\mathsf{dbdh}}_0(1^\lambda)\right] - \Pr\left[\mathcal{A}(\mathcal{I}) \to 1 \,\middle|\, \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}^{\mathsf{dbdh}}_1(1^\lambda)\right] \right|$$

We say that the DBDH assumption holds if for all PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{dbdh}}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

Boyen and Waters pointed out that the following theorem trivially holds [BW06], but for confirmation we write a proof [2] .

Theorem 2.13 *For any adversary $\mathcal{A}$, there exists PPT algorithm $\mathcal{B}$ such that* $\mathsf{Adv}^{\mathsf{dbdh}}_{\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{dlin}}_{\mathcal{B}}$.

Proof. Given DLIN instance $(\Gamma, g, g^\xi, g^\kappa, g^{\delta\xi}, g^{\sigma\kappa}, Q)$, adversary $\mathcal{B}$ for the DLIN problem gives adversary $\mathcal{A}$ for the DBDH problem tuple $(\Gamma, g, g^\kappa, g^\xi, Q, T := e(g^{\delta\xi}, g^\kappa) \cdot e(g^{\sigma\kappa}, g^\xi))$ as a DBDH instance. $T = e(g,g)^{\kappa\xi(\delta+\sigma)}$, so if $Q = g^{\delta+\sigma}$, then the tuple is the same as $\mathcal{G}^{\mathsf{dbdh}}_0$. It implicitly holds that $a = \kappa$, $b = \xi$, $c = \delta + \sigma$, $abc = \kappa\xi(\delta+\sigma)$. If $Q = g^\zeta$ is a uniformly random element in $\mathbb{G}$, then $T = e(g,g)^{\kappa\xi(\delta+\sigma)}$ is a uniformly random element in $\mathbb{G}_T$ and the tuple is the same as $\mathcal{G}^{\mathsf{dbdh}}_1$ since δ and σ are uniformly random and independent of ζ, ξ, and κ. □

[2] This proof is based on personal communication with Keita Xagawa

3 Definitions of Cryptographic Watermarking

We define watermarking schemes for cryptographic functions (one-way functions, hash functions, etc.). Our definition of watermarking schemes can be extended to treat cryptographic data introduced by Yoshida and Fujiwara [YF11]. In this paper, we focus on a family of functions $\mathcal{F} := \{\mathcal{F}_\lambda\}_\lambda$. For example, LTFs are cryptographic functions. Function F is sampled from family $\mathsf{LTF}_\lambda := \left\{ f_{ek}(\cdot) | (ek, ik) \xleftarrow{R} \mathsf{LTF.IGen}(1^\lambda) \right\}$.

A watermarking key generation algorithm for family $\mathcal{F}$ takes as inputs security parameter λ and outputs public parameter pk, secret key sk, mark key mk, detect key dk, and remove key rk. That is, our watermarking schemes are *asymmetric key* watermarking schemes. Public parameter pk includes sampling algorithm $\mathsf{Samp}_\mathcal{F}$, which outputs a function $F \xleftarrow{R} \mathcal{F}_\lambda$ (The sampling algorithm can take sk as an input). *Note that the description of* $\mathsf{Samp}_\mathcal{F}$ *does not include* sk. Our cryptographic watermarking schemes for cryptographic functions $\mathcal{F}$ use public parameter pk and secret key sk to choose a function $F \xleftarrow{R} \mathcal{F}_\lambda$ from the function family. It seems to be a restriction, but it is very reasonable and unavoidable in our setting due to following two reasons.

1. Only authorized entities can generate marked functions from an original non-marked function which is privately generated.

2. If anyone can generate non-marked functions then its security is trivially broken as we will see.

A mark key allows us to embed a mark in function F. A marked function F' must be similar to original function F. A detect/remove key allows us to detect/remove a mark in marked function F'. We sometimes use notation $\mathsf{WM}(F)$ to denote a marked function of F.

Definition 3.1 (Watermarking Scheme for Functions) A watermarking scheme for family $\mathcal{F}$ is a tuple of algorithms $\mathsf{CWM}_\mathcal{F} := \{\mathsf{WMGen}, \mathsf{Mark}, \mathsf{Detect}, \mathsf{Remove}\}$ as follows.

WMGen**:** The key generation algorithm takes as input security parameter λ and outputs public parameter pk (including sampling algorithm $\mathsf{Samp}_\mathcal{F}$), secret key sk, mark key mk, detect key dk, and remove key rk. That is, $(pk, sk, mk, dk, rk) \xleftarrow{R} \mathsf{WMGen}(1^\lambda)$.

Mark**:** The mark algorithm takes as inputs mk and unmarked function F and outputs marked function $\widetilde{F}$. That is, $\widetilde{F} \xleftarrow{R} \mathsf{Mark}(mk, F)$.

Detect**:** The detect algorithm takes as inputs dk and function F' and outputs marked (detect a mark) or $\mathsf{unmarked}$ (no mark), that is, $\mathsf{Detect}(dk, F') \to \mathsf{marked}/\mathsf{unmarked}$.

Remove**:** The remove algorithm takes as inputs rk and marked function $\widetilde{F}$ and outputs unmarked function $F := \mathsf{Remove}(rk, \widetilde{F})$.

As Hopper et al. noted [HMW07], we do not allow any online communication between the Detect and Mark procedures.

We define the security of cryptographic watermarking based on the definition of strong watermarking with respect to the metric space proposed by Hopper et al. [HMW07] and software watermarking proposed by Barak et al. [BGI+12]. We borrow some terms from these studies [BGI+12, HMW07]. Hopper et al. defined a metric space equipped with distance function d and say that object O_1 and O_2 are similar if $d(O_1, O_2) \leq \delta$ for some δ. However, we do not use it since we focus on function families (not perceptual objects).

Basically, the following properties should be satisfied. Most objects $F \in \mathcal{F}_\lambda$ must be unmarked. We define similarity by the functional preserving property. That is, for all input x, output distributions $F(x)$ and $F'(x)$ are identical. Given marked function F', an adversary should not be able to construct a new function $\widetilde{F}$, which is functionally equivalent to F' but unmarked without remove key rk.

Our definitions of the non-removability and unforgeability below are game-based definitions and based on the notion of strong watermarking by Hopper et al. [HMW07]. Our definitions are specialized to focus on cryptographic functions (do not use metric spaces). The non-removability states that even if the adversary is given marked functions, it cannot find a function that is similar to a marked function but does not contain any mark. This is based on the security against removal introduced by Hopper et al. [HMW07]. The unforgeability states that the adversary cannot find a new marked function. This is based on the security against insertion introduced by Hopper et al. [HMW07].

Definition 3.2 (Secure Watermarking for Functions) A watermarking scheme for function family $\mathcal{F}$ is secure if it satisfies the following properties.

Meaningfulness: It holds that for any $F \in \mathcal{F}_\lambda$, $\mathsf{Detect}(dk, F) \rightarrow \mathsf{unmarked}$.

Correctness: For any $F \in \mathcal{F}_\lambda$, $(pk, sk, mk, dk, rk) \xleftarrow{R} \mathsf{WMGen}(1^\lambda)$ and $\mathsf{WM}(F) \xleftarrow{R} \mathsf{Mark}(mk, F)$, it holds that $\mathsf{Detect}(dk, \mathsf{WM}(F)) \rightarrow \mathsf{marked}$ and $\mathsf{Detect}(dk, \mathsf{Remove}(rk, \mathsf{WM}(F))) \rightarrow \mathsf{unmarked}$.

Preserving Functionality: For any input $x \in \{0,1\}^n$ and $F \in \mathcal{F}_\lambda$, it holds that $\mathsf{WM}(F)(x) = F(x)$. If function F' preserves the functionality of function F, then we write $F \equiv F'$.

Polynomial Blowup: There exists a polynomial p such that for any $F \in \mathcal{F}_\lambda$, $|\mathsf{WM}(F)| \leq p(|F| + |mk|)$.

Non-Removability: We say that a watermarking scheme satisfies non-removability (or is non-removable) if it holds that $\mathsf{Adv}^{\mathsf{nrmv}}_{\mathcal{F},\mathcal{A}}(1^\lambda) := \Pr[\mathsf{WMark}_{\mathcal{F},\mathcal{A}}(\lambda) \rightarrow (0, \mathsf{win})] < \mathsf{negl}(\lambda)$. Experiment $\mathsf{WMark}_{\mathcal{F},\mathcal{A}}(\lambda)$ is shown in Figure 1.

Unforgeability: We say that a watermarking scheme satisfies unforgeability (or is unforgeable) if it holds that $\mathsf{Adv}^{\mathsf{uf}}_{\mathcal{F},\mathcal{A}}(1^\lambda) := \Pr[\mathsf{WMark}_{\mathcal{F},\mathcal{A}}(\lambda) \rightarrow (1, \mathsf{win})] < \mathsf{negl}(\lambda)$.

Experiment $\mathsf{WMark}_{\mathcal{F},\mathcal{A}}(\lambda)$:
$(pk, sk, mk, dk, rk) \xleftarrow{R} \mathsf{WMGen}(1^\lambda)$; $\mathsf{MList} := \emptyset$; $\mathsf{CList} := \emptyset$;
$(\beta, F) \xleftarrow{R} \mathcal{A}^{\mathcal{MO},\mathcal{CO},\mathcal{DO}}(1^\lambda, pk)$;
$\mathsf{Detect}(dk, F) \to b$; $\mathsf{IdealDtc}(F) \to B'$;
Case $\beta = 0$: If $b = \mathsf{unmarked}$ and $B' = \{\mathsf{marked}\}$; then return $(0, \mathsf{win})$ else return lose
Case $\beta = 1$: If $b = \mathsf{marked}$, and $B' = \{\mathsf{unmarked}\}$; then return $(1, \mathsf{win})$ else return lose

Oracle $\mathcal{MO}(F)$

$F' \xleftarrow{R} \mathsf{Mark}(mk, F)$;
$\mathsf{MList} := \mathsf{MList} \cup \{F'\}$;
return F';

Oracle $\mathcal{DO}(F)$

$\mathsf{Detect}(dk, F) \to b$;
return b

Oracle $\mathcal{CO}_{\mathcal{F}_\lambda}()$

$F \xleftarrow{R} \mathcal{F}_\lambda$;
$F' \xleftarrow{R} \mathsf{Mark}(mk, F)$;
$\mathsf{CList} := \mathsf{CList} \cup \{F'\}$;
$\mathsf{MList} := \mathsf{MList} \cup \{F'\}$;
return F'

Procedure $\mathsf{IdealDtc}(F)$

if $(\exists F' \in \mathsf{CList} : \ F \equiv F')$; then return $\{\mathsf{marked}\}$
else if $(\exists F' \in \mathsf{MList} : \ F \equiv F')$ then return $\{\mathsf{marked}, \mathsf{unmarked}\}$
else return $\{\mathsf{unmarked}\}$

Figure 1: Experiment for non-removability and unforgeability

The adversary tries to find a function such that the outputs of the actual detection algorithm and the *ideal detection procedure* are different. The ideal detection procedure searches a database and outputs a decision by using online communication to the marking algorithm. The adversary has access to oracles, i.e., the mark, detect, and challenge oracles. The mark oracle returns a marked function for a queried non-marked function. The detect oracle determines whether a queried function is marked or not. The challenge oracle generates a new (non-marked) function, embeds a mark in the new function, and returns the marked function (the original non-marked function is hidden).

Eventually, the adversary outputs function F and bit β. Value β just indicates which experiment is selected. When $\beta = 0$, it means that the adversary claim that it succeeded in removing a mark from some marked function F' without the remove key. This case is for security against removal. When $\beta = 1$, it means that the adversary claim that it succeeded in embedding a mark in some original function F' without the mark key. This case is for security against forgery.

As Hopper et al. explained [HMW07], we must introduce the challenge oracle because if it does not exist, then a trivial attack exists. If the adversary samples an unmarked function $F \in \mathcal{F}_\lambda$, queries it to the mark oracle, and finally outputs them as solutions for $\beta = 0$. The actual detect algorithm returns unmarked but the ideal detect procedure returns $\{\mathsf{marked}, \mathsf{unmarked}\}$ since an equivalent function is recorded in MList.

4 Proposed Watermarking Scheme based on Lewko's scheme

We present LTFs and watermarking schemes for the LTFs that are secure under the DLIN assumption in this section.

Generally speaking, LTFs can be constructed from homomorphic encryption schemes as observed in many papers [FGK+10, HO12, PW11]. Lewko/Okamoto-Takashima proposed an IBE/IPE scheme based on DPVS which is homomorphic and secure under the DLIN assumption (See Appendix A for descriptions of their schemes). We can easily construct a LTF from the IBE scheme by applying the matrix encryption technique introduced by Peikert and Waters [PW08, PW11]. Note that IBE schemes are obtained from IPE schemes where the predicate is the equality test.

In this section, we present a scheme based on the Lewko IBE scheme. Basically, previous works used homomorphic *PKE* schemes to construct LTFs. However, we use homomorphic *IBE* schemes to achieve watermarking scheme since we would like to use identities as tags for function indices and *dual system encryption*. To construct LTFs based on IBE schemes, we use not only ciphertexts under some identity but also a private key for the identity. If there is no private key (we call it conversion key in the scheme), then we cannot obtain valid outputs that can be inverted by an inversion key of the LTF. Note that the conversion key is *not a trapdoor inversion key for the LTF*.

4.1 LTF based on Lewko's IBE scheme

Our LTF $\mathsf{LTF}_{\mathsf{mult}}$ based on the Lewko IBE is as follows.

$\mathsf{LTF.IGen}(1^\lambda)$ **:** It generates $(\mathfrak{D}, \mathfrak{D}^*) \xleftarrow{\cup} \mathsf{Dual}(\mathbb{Z}_p^8)$, chooses $\alpha, \theta, \sigma \xleftarrow{\cup} \mathbb{Z}_p$, $\boldsymbol{\psi} := (\psi_1, \ldots, \psi_\ell) \xleftarrow{\cup} \mathbb{Z}_p^\ell$, and sets $g_T := e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*}$ and $g_{T_j} := g_T^{\psi_j}$ for all $j \in [\ell]$. It chooses an arbitrary $\mathsf{tag} \in \mathbb{Z}_p$ and $s_{1,i}, s_{2,i} \xleftarrow{\cup} \mathbb{Z}_p$ for all $i \in [\ell]$ and generates $u_{i,j} := g_{T_j}^{s_{1,i}} \cdot g_T^{m_{i,j}}$ and $\boldsymbol{v}_i := g^{s_{1,i}\vec{d}_1 + s_{1,i}\mathsf{tag}\vec{d}_2 + s_{2,i}\vec{d}_3 + s_{2,i}\mathsf{tag}\vec{d}_4}$ for all $i,j \in [\ell]$ where $m_{i,i} = 1$ and $m_{i,j} = 0$ (if $i \neq j$). It chooses $r_1, r_2 \xleftarrow{\cup} \mathbb{Z}_p$ and generates $\boldsymbol{k}_{\mathsf{tag}} := g^{(\alpha + r_1\mathsf{tag})\theta\vec{d}_1^* - r_1\theta\vec{d}_2^* + r_2\mathsf{tag}\sigma\vec{d}_3^* - r_2\sigma\vec{d}_4^*}$. It returns $ek := (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{\mathsf{tag}}) := (\{u_{i,j}\}_{i,j=1}^\ell, \{\boldsymbol{v}_i\}_{i=1}^\ell, \boldsymbol{k}_{\mathsf{tag}})$, $ik := \boldsymbol{\psi}$. We call $\boldsymbol{k}_{\mathsf{tag}}$ conversion key. Hereafter, $\{u_{i,j}\}_{i,j}$ and $\{\boldsymbol{v}_i\}_i$ denote $\{u_{i,j}\}_{i,j=1}^\ell$ and $\{\boldsymbol{v}_i\}_{i=1}^\ell$, respectively if it is clear from the context. Note ek includes tag, but we omit it for simplicity.

$\mathsf{LTF.LGen}(1^\lambda)$ **:** This is the same as $\mathsf{LTF.IGen}$ except that for all $i,j \in [\ell]$, $m_{i,j} = 0$ and $ik := \perp$.

$\mathsf{LTF.Eval}(ek, \vec{x})$**:** For input $\vec{x} \in \{0,1\}^\ell$, it parses $ek = (\{u_{i,j}\}_{i,j}, \{\boldsymbol{v}_i\}_i, \boldsymbol{k}_{\mathsf{tag}})$ and computes

$$
\begin{aligned}
y_j &:= \prod_i u_{i,j}^{x_i} = \prod_i g_{T_j}^{x_i s_{1,i}} \cdot g_T^{x_i m_{i,j}} = g_{T_j}^{\langle \vec{x}, \vec{s}_1 \rangle} g_T^{x_j} \\
y_{\ell+1} &:= \prod_i \boldsymbol{v}_i^{x_i} = \prod_i g^{x_i s_{1,i}\vec{d}_1 + x_i s_{1,i}\mathsf{tag}\vec{d}_2 + x_i s_{2,i}\vec{d}_3 + x_i s_{2,i}\mathsf{tag}\vec{d}_4} \\
&= g^{\langle \vec{x}, \vec{s}_1 \rangle \vec{d}_1 + \langle \vec{x}, \vec{s}_1 \rangle \mathsf{tag}\vec{d}_2 + \langle \vec{x}, \vec{s}_2 \rangle \vec{d}_3 + \langle \vec{x}, \vec{s}_2 \rangle \mathsf{tag}\vec{d}_4}
\end{aligned}
$$

where $\vec{s}_1 := (s_{1,1}, \ldots, s_{1,\ell})$, $\vec{s}_2 := (s_{2,1}, \ldots, s_{2,\ell})$, and $y'_{\ell+1} := \boldsymbol{e}(y_{\ell+1}, \boldsymbol{k}_{\mathsf{tag}}) = e(g,g)^{\alpha\theta\vec{d}_1 \cdot \vec{d}_1^* \langle \vec{x}, \vec{s}_1 \rangle}$ and returns output $\boldsymbol{y} := (y_1, \ldots, y_\ell, y'_{\ell+1})$.

$\mathsf{LTF.Invert}(ik, \boldsymbol{y})$**:** For input $\boldsymbol{y}$, it computes $x'_j := y_j / (y'_{\ell+1})^{\psi_j} = g_{T_j}^{\langle \vec{x}, \vec{s}_1 \rangle} g_T^{x_j} / g_T^{\langle \vec{x}, \vec{s}_1 \rangle \cdot \psi_j}$ and let $x_j \in \{0,1\}$ be such that $x'_j = g_T^{x_j}$. It returns $\vec{x} = (x_1, \ldots, x_\ell)$.

Theorem 4.1 $\mathsf{LTF}_{\mathsf{mult}}$ *is a lossy trapdoor function if the DBDH assumption holds.*

Lemma 4.2 (Lossiness of $\mathsf{LTF}_{\mathsf{mult}}$**)** $\mathsf{LTF}_{\mathsf{mult}}$ *is* $(\ell - \log q)$*-lossy.*

Proof. We compute lossiness ℓ'. For a lossy function index generated by $\mathsf{LTF.LGen}$, an output is $\boldsymbol{y} = (g_T^{s'_1 \boldsymbol{\psi}}, e(g,g)^{s'_1}) = (g_{T_1}^{s'_1}, \ldots, g_{T_\ell}^{s'_1}, e(g,g)^{s'_1})$ where $s'_1 = \langle \vec{x}, \vec{s}_1 \rangle \in \mathbb{Z}_p$. Here, secret trapdoor $\boldsymbol{\psi}$ is fixed by the function index. This means that for any given image $\boldsymbol{y}$, there are at least q possible values for $\langle \vec{x}, \vec{s}_1 \rangle$ and pre-images. Therefore, equation $|\mathsf{D}| / 2^{\ell'} = q$ holds by the definition of the lossiness. By this equation, we can derive equation $\ell' = \ell - \log q$ since $|\mathsf{D}| = 2^\ell$. □

We introduce some notations before we show the indistinguishability. We borrow the notation introduced by Peikert and Waters [PW11]. For matrix $\boldsymbol{Y} = (y_{i,j}) \in \mathbb{Z}_p^{h \times w}$, we define $g_T^{\boldsymbol{Y}} = (g_T^{y_{i,j}}) \in \mathbb{G}_T^{h \times w}$. Algorithm $\mathsf{GenConceal}(h, w)$ which was introduced by Peikert and Waters is as follows.

1. Choose $\boldsymbol{\zeta} := (\zeta_1, \ldots, \zeta_h) \xleftarrow{\cup} \mathbb{Z}_p^h$ and $\boldsymbol{\psi} := (\psi_1, \ldots, \psi_w, 1) \xleftarrow{\cup} \mathbb{Z}_p^w \times \{1\}$.
2. Let $\boldsymbol{V} := \boldsymbol{\zeta} \otimes \boldsymbol{\psi} = \boldsymbol{\zeta}^\top \boldsymbol{\psi} \in \mathbb{Z}_p^{h \times (w+1)}$ be the outer product of $\boldsymbol{\zeta}$ and $\boldsymbol{\psi}$.
3. Output $\boldsymbol{C} := g_T^{\boldsymbol{V}} \in \mathbb{G}_T^{h \times (w+1)}$ as the concealer matrix and $\boldsymbol{\psi}$ as the trapdoor.

The original concealer matrix by Peikert and Waters is over $\mathbb{G}^{h \times w}$, but we use a matrix over $\mathbb{G}_T$ since we use bilinear maps.

Lemma 4.3 (Indistinguishability of $\mathsf{LTF}_{\mathsf{mult}}$**)** *If the DBDH assumption holds, then* $\mathsf{LTF}_{\mathsf{mult}}$ *satisfies indistinguishability.*

Proof. For $\boldsymbol{\psi} = (\psi_1, \ldots, \psi_\ell, 1)$, $g_T^{\zeta \boldsymbol{\psi}}$ denotes $(g_T^{\zeta\psi_1}, \ldots, g_T^{\zeta\psi_\ell}, g_T^{\zeta})$. We need three steps to show the lemma.

First, we will show that if the DBDH assumption holds, then $(g_T^{\boldsymbol{\psi}}, \boldsymbol{y} = g_T^{\zeta\boldsymbol{\psi}})$ is computationally indistinguishable from $(g_T^{\boldsymbol{\psi}}, \boldsymbol{y} = g_T^{\boldsymbol{t}})$ where $\zeta \xleftarrow{\cup} \mathbb{Z}_p$, $\boldsymbol{\psi} \xleftarrow{\cup} \mathbb{Z}_p^\ell \times \{1\}$, and $\boldsymbol{t} \xleftarrow{\cup} \mathbb{Z}_p^{\ell+1}$. Note that the $\ell+1$-th element of $\boldsymbol{\psi}$ is fixed to 1. To show the indistinguishability, we define hybrid distribution HYB_j: We chooses $\alpha_0, \zeta \xleftarrow{\cup} \mathbb{Z}_p$ and $\boldsymbol{\psi} \xleftarrow{\cup} \mathbb{Z}_p^\ell \times \{1\}$ and sets $g_T := e(g,g)^{\alpha_0}$ and $\boldsymbol{y} := (g_T^{\zeta\psi_1}, \ldots, g_T^{\zeta\psi_j}, y_{j+1}, \ldots, y_\ell, g_T^{\zeta})$ where $y_k \xleftarrow{\cup} \mathbb{G}_T$ for $k > j$. That is, y_k is uniformly random element for $k > j$. The output is $(g^{\boldsymbol{\psi}}, \boldsymbol{y})$. We note that $\mathsf{HYB}_0 = (g_T^{\boldsymbol{\psi}}, g_T^{\boldsymbol{t}})$ and $\mathsf{HYB}_\ell = (g_T^{\boldsymbol{\psi}}, g_T^{\zeta\boldsymbol{\psi}})$. We show that for each $j \in [\ell]$, HYB_j and HYB_{j-1} are computationally indistinguishable under

the DBDH assumption. We construct PPT algorithm $\mathcal{B}$ that uses distinguisher $\mathcal{D}$ for HYB_j and HYB_{j-1}. $\mathcal{B}$ is given input $(\mathsf{param}_{\mathbb{G}}, g, g^a, g^b, g^c, Q)$ and computes $(\boldsymbol{\tau}, \boldsymbol{y}) \in \mathbb{G}_T^{\ell+1} \times \mathbb{G}_T^{\ell+1}$ as follows. $\mathcal{B}$ sets $\tau_{\ell+1} := g_T := e(g,g)^{\alpha_0}$ for $\alpha_0 \xleftarrow{\mathsf{U}} \mathbb{Z}_p$, $y_{\ell+1} := e(g, g^c)^{\alpha_0} = g_T^c$. It implicitly holds $\zeta := c$.

- For $k \in [j-1]$, $\mathcal{B}$ chooses ψ_k and sets $\tau_k := g_T^{\psi_k}$, $y_k := e(g, g^c)^{\alpha_0 \psi_k} = (g_T^c)^{\psi_k}$.
- For $k = j+1, \ldots, \ell$, $\mathcal{B}$ chooses ψ_k and sets $\tau_k := g_T^{\psi_k}$, $y_k \xleftarrow{\mathsf{U}} \mathbb{G}_T$.
- For $k = j$, $\mathcal{B}$ embeds the instance. That is, $\mathcal{B}$ sets $\tau_j := e(g^a, g^b)^{\alpha_0} = g_T^{ab}$, $y_j := Q^{\alpha_0}$ (implicitly $\psi_j := ab$).

If $Q = e(g,g)^{abc}$, then $y_j = g_T^{\psi_j \zeta}$ and $(\boldsymbol{\tau}, \boldsymbol{y}) = \mathsf{HYB}_j$. If $Q \xleftarrow{\mathsf{U}} \mathbb{G}_T$, then $y_j \xleftarrow{\mathsf{U}} g_T^{t_j}$ where $t_j \xleftarrow{\mathsf{U}} \mathbb{Z}_p$ and $(\boldsymbol{\tau}, \boldsymbol{y}) = \mathsf{HYB}_{j-1}$. Therefore, $\mathsf{HYB}_j \overset{\mathsf{C}}{\approx} \mathsf{HYB}_{j-1}$. As a corollary, $\mathsf{HYB}_0 \overset{\mathsf{C}}{\approx} \mathsf{HYB}^{\ell}$.

Second, We define new hybrid distributions $\mathsf{HYB}'_0, \ldots, \mathsf{HYB}'_{\ell'}$ over matrices $\boldsymbol{C} \in \mathbb{G}_T^{\ell' \times (\ell+1)}$. We define HYB'_i: We set elements in the first i rows of $\boldsymbol{C}$ as an output of $\mathsf{GenConceal}$. On the other hand, we choose uniformly random elements in the last $(\ell' - i)$ rows over $\mathbb{G}_T^{\ell+1}$. We can easily verify that $\mathsf{HYB}'_{\ell'}$ is the same as $\mathsf{GenConceal}$ and HYB'_0 is the uniform distribution over $\mathbb{G}_T^{\ell' \times (\ell+1)}$. We show that for each $i \in [\ell']$, $\mathsf{HYB}'_i \overset{\mathsf{C}}{\approx} \mathsf{HYB}'_{i-1}$ if the DBDH assumption holds. We construct PPT algorithm $\mathcal{D}$ that uses distinguisher $\mathcal{D}'$ for HYB'_i and HYB'_{i-1}. $\mathcal{D}$ is given instance $(g_T^{\boldsymbol{\psi}}, \boldsymbol{y} \in \mathbb{G}_T^{\ell+1})$, $\mathcal{D}$ generates matrix $\boldsymbol{C}$ as follows.

- For $k \in [i-1]$, chooses $\zeta_k \xleftarrow{\mathsf{U}} \mathbb{Z}_p$ and set the k-th row of $\boldsymbol{C}$ be $\boldsymbol{c}_k := (g_T^{\boldsymbol{\psi}})^{\zeta_k} = g_T^{\zeta_k \boldsymbol{\psi}}$.
- For $k = i+1, \ldots, \ell'$, $\mathcal{B}$ chooses uniformly random elements over $\mathbb{G}_T^{(\ell+1)}$.
- For $k = i$, sets the i-th row of $\boldsymbol{C}$ be $\boldsymbol{c}_i := \boldsymbol{y}$. That is, $\mathcal{D}$ embeds the instance.

If $\boldsymbol{y} = g_T^{\zeta \boldsymbol{\psi}}$, then the distribution is the same as HYB'_i, else if $\boldsymbol{y}$ is uniformly random, then the distribution is the same as HYB'_{i-1}. Therefore, $\mathsf{HYB}'_0 \overset{\mathsf{C}}{\approx} \mathsf{HYB}'_{\ell'}$.

Finally, we prove the lemma. In the above simulation, we can replace $(\ell+1)$-th column element g_T^{ζ} with $g^{\zeta \vec{d}_1 + \zeta \mathsf{tag} \vec{d}_2 + s_2 \vec{d}_3 + s_2 \mathsf{tag} \vec{d}_4}$ where $(\mathfrak{D}, \mathfrak{D}^*) \xleftarrow{\mathsf{R}} \mathsf{Dual}(\mathbb{Z}_p^8)$. We should simulate $\boldsymbol{V} = g^{\zeta \vec{d}_1 + \zeta \mathsf{tag} \vec{d}_2 + s_2 \vec{d}_3 + s_2 \mathsf{tag} \vec{d}_4}$ and $\boldsymbol{k}_{\mathsf{tag}} = g^{(\alpha + r_1 \mathsf{tag}) \theta \vec{d}_1^* - r_1 \theta \vec{d}_2^* r_2 \mathsf{tag} \sigma \vec{d}_3^* - r_2 \sigma \vec{d}_4^*}$ for $s_2 \xleftarrow{\mathsf{U}} \mathbb{Z}_p$. If we have $g^{\zeta} = g^c$ and matrix $\mathfrak{D} = (\vec{d}_1, \ldots, \vec{d}_8)$, then we can compute $g^{\zeta \vec{d}_1 + \zeta \mathsf{tag} \vec{d}_2} = (g^{\zeta})^{\vec{d}_1 + \mathsf{tag} \vec{d}_2}$ without knowing ζ since simulator $\mathcal{B}$ generates $(\mathfrak{D}, \mathfrak{D}^*) \xleftarrow{\mathsf{R}} \mathsf{Dual}(\mathbb{Z}_p^8)$ by itself. $\mathcal{B}$ can also generate $\boldsymbol{k}_{\mathsf{tag}}$ since it has $\mathfrak{D}^*$. Therefore, in the above simulation we can replace $(\ell+1)$-th column element g_T^{ζ} with $g^{\zeta \vec{d}_1 + \zeta \mathsf{tag} \vec{d}_2 + s_2 \vec{d}_3 + s_2 \mathsf{tag} \vec{d}_4}$ and add $\boldsymbol{k}_{\mathsf{tag}} = g^{(\alpha + r_1 \mathsf{tag}) \theta \vec{d}_1^* - r_1 \theta \vec{d}_2^* r_2 \mathsf{tag} \sigma \vec{d}_3^* - r_2 \sigma \vec{d}_4^*}$ (We can set $\alpha_0 := \alpha \theta \vec{d}_1 \cdot \vec{d}_1^*$). Therefore, $\mathsf{LTF}_{\mathsf{mult}}$ satisfies indistinguishability. $\square$

4.2 Watermarking Scheme for $\mathsf{LTF}_{\mathsf{mult}}$

In this section, we present our watermarking scheme. First, we give an overview of our construction.

We added extra two dimensions of DPVS to the original Lewko IBE scheme since we use the extra dimensions to embed watermarks. Even if we add a vector spanned by $\vec{d}_7^*$ and $\vec{d}_8^*$ to conversion key $\boldsymbol{k}_{\mathsf{tag}}$, which is spanned by $\vec{d}_1^*, \ldots, \vec{d}_4^*$, the conversion key is indistinguishable from the original one since vectors $\vec{d}_7, \vec{d}_8, \vec{d}_7^*, \vec{d}_8^*$ are hidden. Moreover, the marked index works as the original non-marked index since elements in function index $\boldsymbol{V}$ are spanned by $\vec{d}_1, \ldots, \vec{d}_4$ and components $\vec{d}_7^*, \vec{d}_8^*$ are canceled. However, if we have a vector which is spanned by $\vec{d}_7, \vec{d}_8$, then we can detect the mark which is generated by $\vec{d}_7^*, \vec{d}_8^*$. If we have complete dual orthonormal bases $(\mathfrak{D}, \mathfrak{D}^*)$, then we can use the vector decomposition algorithm introduced in Section 2.3 and eliminate the vector spanned by $\vec{d}_7^*, \vec{d}_8^*$, i.e., watermarks.

Our watermarking scheme $\mathsf{CWM}_{\mathsf{mult}}$ for $\mathsf{LTF}_{\mathsf{mult}}$ is as follows:

$\mathsf{WMGen}(1^\lambda)$**:** It generates $(\mathfrak{D}, \mathfrak{D}^*) \xleftarrow{\cup} \mathsf{Dual}(\mathbb{Z}_p^8)$, chooses $\alpha, \theta, \sigma \xleftarrow{\cup} \mathbb{Z}_p$, and sets $g_T := e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*}$, $\widehat{\mathbb{D}} := (g_T, g^{\vec{d}_1}, \ldots, g^{\vec{d}_4})$, $pk := (\widehat{\mathbb{D}}, \mathsf{Samp})$, $sk := (g^{\alpha\theta\vec{d}_1^*}, g^{\theta\vec{d}_1^*}, g^{\theta\vec{d}_2^*}, g^{\sigma\vec{d}_3^*}, g^{\sigma\vec{d}_4^*})$, $mk := (g^{\vec{d}_7^*}, g^{\vec{d}_8^*})$, $dk := (g^{\vec{d}_7}, g^{\vec{d}_8})$, and $rk := (\mathfrak{D}, \mathfrak{D}^*)$. The sampling algorithm $\mathsf{Samp}(\mathsf{param}_{\mathbb{V}}, \widehat{\mathbb{D}}, sk)$ chooses $\mathsf{tag} \in \mathbb{Z}_p$ and $\boldsymbol{\psi} \xleftarrow{\cup} \mathbb{Z}_p^\ell$, $\vec{s}_1, \vec{s}_2 \xleftarrow{\cup} \mathbb{Z}_p^\ell$, and generates $(ek, ik) := ((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{\mathsf{tag}}), \boldsymbol{\psi})$ as $\mathsf{LTF.IGen}$. It computes $\boldsymbol{k}_{\mathsf{tag}} := g^{(\alpha + r_1\mathsf{tag})\theta\vec{d}_1^* - r_1\theta\vec{d}_2^* + r_2\mathsf{tag}\sigma\vec{d}_3^* - r_2\sigma\vec{d}_4^*}$. Note that sk is *not* included in the description of Samp. Keys sk, mk, and rk are secret. Key dk can be disclosed.

$\mathsf{Mark}(mk, ek)$**:** It parses $ek = (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{\mathsf{tag}}) = (\boldsymbol{U}, \boldsymbol{V}, g^{(\alpha + r_1\mathsf{tag})\theta\vec{d}_1^* - r_1\theta\vec{d}_2^* + r_2\mathsf{tag}\sigma\vec{d}_3^* - r_2\sigma\vec{d}_4^*})$, chooses $t_7, t_8 \xleftarrow{\cup} \mathbb{Z}_p$, and computes $\widetilde{\boldsymbol{k}}_{\mathsf{tag}} := \boldsymbol{k}_{\mathsf{tag}} \cdot g^{t_7\vec{d}_7^* + t_8\vec{d}_8^*}$ by using $g^{\vec{d}_7^*}$ and $g^{\vec{d}_8^*}$. It outputs marked function index $\mathsf{WM}(ek) = (\boldsymbol{U}, \boldsymbol{V}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}})$.

$\mathsf{Detect}(dk, \widetilde{ek})$**:** It parses $\widetilde{ek} = (\boldsymbol{U}, \boldsymbol{V}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}})$, chooses $u_7, u_8 \xleftarrow{\cup} \mathbb{Z}_p^\times$, and computes $\boldsymbol{c} := g^{u_7\vec{d}_7 + u_8\vec{d}_8}$. Next, it computes $\Delta := \boldsymbol{e}(\boldsymbol{c}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}})$. If it holds that $\Delta = \boldsymbol{e}(\boldsymbol{c}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}}) \neq 1$, then it outputs marked. Else if, it holds that $\Delta = 1$, then it outputs unmarked. *NOTE:* If we allow public detection, we publish detection key $g^{u_7\vec{d}_7 + u_8\vec{d}_8}$.

$\mathsf{Remove}(rk, \widetilde{ek})$**:** It parses $rk = (\mathfrak{D}, \mathfrak{D}^*)$ and $\widetilde{ek} = (\boldsymbol{U}, \boldsymbol{V}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}})$, runs the decomposition algorithm. That is, it computes $\sum_{j=1}^m g^{z_j\vec{d}_j^*} = \mathsf{Decomp}(\widetilde{\boldsymbol{k}}_{\mathsf{tag}}, (g^{\vec{d}_1^*}, \ldots, g^{\vec{d}_m^*}), \mathfrak{D}^*, (g^{\vec{d}_1^*}, \ldots, g^{\vec{d}_8^*}))$ for all $m < 8$, where $z_j \in \mathbb{Z}_p$ and obtains $g^{z_j\vec{d}_j^*}$ for $j = 1, \ldots, 8$. It holds $\widetilde{\boldsymbol{k}}_{\mathsf{tag}} = g^{z_1\vec{d}_1^* + \cdots + z_8\vec{d}_8^*}$. It computes $\boldsymbol{k}'_{\mathsf{tag}} := \widetilde{\boldsymbol{k}}_{\mathsf{tag}} / g^{z_7\vec{d}_7^* + z_8\vec{d}_8^*}$ and outputs $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}'_{\mathsf{tag}})$ as an unmarked index.

We can easily understand that meaningfulness, correctness, and polynomial blowup hold.

Preserving functionality holds since elements in $\boldsymbol{U}$ and $\boldsymbol{V}$ do not include vectors $g^{\vec{d}_7}$ and $g^{\vec{d}_8}$ and vector $g^{t_1\vec{d}_7^* + t_2\vec{d}_8^*}$ does not interfere the computation of $\mathsf{LTF.Eval}$.

Note that if we do not have secret key $(g^{\vec{d}_1^*}, \ldots, g^{\vec{d}_4^*})$, then we cannot compute a complete function index, that is, we cannot compute conversion key $\boldsymbol{k}_{\mathsf{tag}}$. This seems to be a restriction, but in the scenario of watermarking schemes, this is acceptable by following reasons. We use watermarking schemes to authorize objects and such objects are privately generated by authors. For example, movies, music files, and software are generated by some companies and they do not distribute unauthorized (unmarked) objects. Moreover, in the experiment on security, the adversary is given a oracle which gives marked function indices. Thus, it is reasonable that unauthorized parties cannot efficiently sample functions by themselves.

4.3 Security Proofs for $\mathsf{CWM}_{\mathsf{mult}}$

Theorem 4.4 *Our watermarking scheme* $\mathsf{CWM}_{\mathsf{mult}}$ *is secure under the DLIN assumption.*

We prove the theorem by proving Theorems 4.5 and 4.6.

Theorem 4.5 (Non-Removability) *Our scheme* $\mathsf{CWM}_{\mathsf{mult}}$ *satisfies non-removability under the subspace assumption.*

Proof. If $\mathcal{A}$ outputs $(0, ek^*)$, where $\mathsf{Detect}(dk, ek^*) \to \mathsf{unmarked}$ and $\mathsf{IdealDtc}(ek^*) \to \mathsf{marked}$, then we construct algorithm $\mathcal{B}$, which solves the subspace problem with $k = 1$ and $n = 8$. $\mathcal{B}$ is given Γ, $D = (g^{\vec{b}_1}, g^{\vec{b}_2}, g^{\vec{b}_4}, \ldots, g^{\vec{b}_8}, g^{\eta\vec{b}_1^*}, g^{\beta\vec{b}_2^*}, g^{\vec{b}_3^*}, \ldots, g^{\vec{b}_8^*}, U_1 = g^{\mu_1\vec{b}_1+\mu_2\vec{b}_2+\mu_3\vec{b}_3}, \mu_3)$, and Q_{b} where Q_{b} is $V_1 = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*}$ or $W_1 = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*+\tau_3\vec{b}_3^*}$. $\mathcal{B}$ chooses $\theta, \alpha', \sigma \xleftarrow{\cup} \mathbb{Z}_p$, sets

$$\begin{array}{llllllll}
\vec{d}_1 := \vec{b}_3^* & \vec{d}_2 := \vec{b}_4^* & \vec{d}_3 := \vec{b}_5^* & \vec{d}_4 := \vec{b}_6^* & \vec{d}_5 := \vec{b}_7^* & \vec{d}_6 := \vec{b}_8^* & \vec{d}_7 := \vec{b}_1^* & \vec{d}_8 := \vec{b}_2^* \\
\vec{d}_1^* := \vec{b}_3 & \vec{d}_2^* := \vec{b}_4 & \vec{d}_3^* := \vec{b}_5 & \vec{d}_4^* := \vec{b}_6 & \vec{d}_5^* := \vec{b}_7 & \vec{d}_6^* := \vec{b}_8 & \vec{d}_7^* := \vec{b}_1 & \vec{d}_8^* := \vec{b}_2,
\end{array}$$

and can generate $pk = (e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*}, g^{\vec{d}_1}, \ldots, g^{\vec{d}_4}) := (e(g^{\vec{b}_4^*}, g^{\vec{b}_4})^{\alpha'\mu_3\theta}, g^{\vec{b}_3^*}, g^{\vec{b}_4^*}, g^{\vec{b}_5^*}, g^{\vec{b}_6^*})$ and $mk = (g^{\vec{d}_7^*}, g^{\vec{d}_8^*}) := (g^{\vec{b}_1}, g^{\vec{b}_2})$. $\mathcal{B}$ has a detect key, which is essentially the same as $g^{\vec{d}_7}$ and $g^{\vec{d}_8}$ since $g^{\eta\vec{b}_1^*}, g^{\beta\vec{b}_2^*}$ are given. (In a case of public detection, we publish $(g^{\eta\vec{b}_1^*})^{u_1'}(g^{\beta\vec{b}_2^*})^{u_2'}$.) Coefficients β and η do not affect the detect algorithm. $\mathcal{B}$ has $(g^{\vec{d}_2^*}, \ldots, g^{\vec{d}_8^*})$ but does not have $g^{\vec{d}_1^*}$ since $g^{\vec{b}_3}$ is not given. That is, $\mathcal{B}$ has the mark key and perfectly simulates the mark oracle. On the other hand, the secret key is incomplete as follows, $sk = (\perp, \perp, g^{\theta\vec{d}_2^*}, g^{\sigma\vec{d}_3^*}, g^{\sigma\vec{d}_4^*}) := (\perp, \perp, g^{\theta\vec{b}_4}, g^{\sigma\vec{b}_5}, g^{\sigma\vec{b}_6})$.

It implicitly holds $\alpha = \alpha'\mu_3$. To simulate the challenge oracle without the complete sk, for tag, $\mathcal{B}$ chooses $r_1', r_2, t_7, t_8 \xleftarrow{\cup} \mathbb{Z}_p$ and computes

$$\begin{aligned}
\widetilde{\boldsymbol{k}}_{\mathsf{tag}} &:= (U_1)^{(\alpha'+r_1'\mathsf{tag})\theta} g^{-r_1'\mu_3\theta\vec{d}_2^*+r_2\mathsf{tag}\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*+t_7\vec{d}_7^*+t_8\vec{d}_8^*} \\
&= g^{(\alpha+r_1\mathsf{tag})\theta\vec{d}_1^*-r_1\theta\vec{d}_2^*+r_2\mathsf{tag}\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*+(t_7-\theta(\alpha'+r_1'\mathsf{tag})\mu_1)\vec{d}_7^*+(t_8-\theta(\alpha'+r_1'\mathsf{tag})\mu_2)\vec{d}_8^*}.
\end{aligned}$$

We set $r_1 := \mu_3 r_1'$. This is a valid marked index. If $\mathcal{A}$ outputs valid unmarked index $ek^* = (\boldsymbol{U}^*, \boldsymbol{V}^*, \boldsymbol{k}^*_{\mathsf{tag}^*})$ where $\boldsymbol{k}^*_{\mathsf{tag}^*} = g^{(\alpha+r_1^*\mathsf{tag}^*)\theta\vec{d}_1^*-r_1^*\theta\vec{d}_2^*+r_2^*\mathsf{tag}^*\sigma\vec{d}_3^*-r_2^*\sigma\vec{d}_4^*}$, then $\mathcal{B}$ computes $\Delta := \boldsymbol{e}(Q_{\mathsf{b}}, \boldsymbol{k}^*_{\mathsf{tag}^*})$. If $\Delta = 1$, then $\mathcal{B}$ outputs 0 ($\mathsf{b} = 0$), otherwise, it outputs 1. $\mathcal{B}$ can output correct b. Analysis is as follows.

- If $Q_0 = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*} = g^{\tau_1\eta\vec{d}_7+\tau_2\beta\vec{d}_8}$ is given, then $\Delta = 1$ since $\boldsymbol{k}^*_{\mathsf{tag}^*}$ does not include vectors $\vec{d}_7^*$ and $\vec{d}_8^*$.
- If $Q_1 = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*+\tau_3\vec{b}_3^*} = g^{\tau_1\eta\vec{d}_7+\tau_2\beta\vec{d}_8+\tau_3\vec{d}_1}$ is given, then $\Delta = e(g,g)^{(\alpha+r_1\mathsf{tag})\theta\tau_3\vec{d}_1\cdot\vec{d}_1^*} \neq 1$.

Thus, $\mathcal{B}$ breaks the problem. □

Before we prove the unforgeability, we give a few remarks. Note that the adversary is not allowed to output a function index whose identity is equal to those of indices generated by the challenge oracle or are queried to the mark oracle. This is justified by the following fact. If it is allowed, then it means the adversary has *already had a (functionally equivalent) marked index* for the given or queried identity, that is, an IBE private key for the same identity. This is inevitable. For simplicity, we prove the unforgeability explained above.

Of course, it may be possible that the adversary outputs a function index whose tag is the same as an queried tag and functionality is different from given functions by the oracles. We can extend the unforgeability to stronger one by using known techniques that convert standard unforgeable signature schemes into *strongly unforgeable* signature schemes. We now define algorithm $\mathsf{Xtr}(pk, sk, \mathsf{tag})$. It chooses $r_1, r_2 \xleftarrow{\mathsf{U}} \mathbb{Z}_p$ and outputs $\boldsymbol{k}_{\mathsf{tag}} := g^{(\alpha+r_1\mathsf{tag})\theta\vec{d}_1^*-r_1\theta\vec{d}_2^*+r_2\mathsf{tag}\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*}$. We can consider $\boldsymbol{k}_{\mathsf{tag}}$ be a signature for tag. Naor pointed out that signature schemes can be derived from IBE schemes [BF03]. Thus, we can prove the unforgeability of our watermarking schemes by using the unforgeability of signature schemes derived from IBE schemes of Okamoto-Takashima and Lewko. Huang, Wong, and Zhao proposed a generic transformation technique for converting unforgeable signature schemes into strongly unforgeable ones [HWZ07]. Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be strong one-time signature scheme. The conversion is as follows: generates $(vk_{\mathsf{ot}}, sk_{\mathsf{ot}}) \xleftarrow{\mathsf{R}} \mathsf{Gen}(1^\lambda)$, $\boldsymbol{k}_{vk_{\mathsf{ot}}} \xleftarrow{\mathsf{R}} \mathsf{Xtr}(pk, sk, vk_{\mathsf{ot}})$, that is, tag is replaced by vk_{ot}, and $\mathsf{sig} := \mathsf{Sign}(sk_{\mathsf{ot}}, \mathsf{tag} \parallel \boldsymbol{k}_{vk_{\mathsf{ot}}})$ and outputs $(\boldsymbol{k}_{vk_{\mathsf{ot}}}, \mathsf{sig}, vk_{\mathsf{ot}})$ as a signature. If we show that the adversary cannot forge a marked index for tag which is not queried to the oracle and $\boldsymbol{k}_{\mathsf{tag}}$ in our scheme is replaced with $(\boldsymbol{k}_{vk_{\mathsf{ot}}}, \mathsf{sig}.vk_{\mathsf{ot}})$ (of course, ek includes tag though we omit it), then our watermarking scheme satisfies strong unforgeability of watermarking schemes by the strongly unforgeable property. That is, we can show that the adversary cannot output a function index with a queried tag. In this paper, we prove the standard unforgeability for simplicity.

Next, we prove unforgeability.

Theorem 4.6 *Our scheme* $\mathsf{CWM}_{\mathsf{mult}}$ *satisfies unforgeability under the subspace assumption.*

Proof. Let q_{M} and q_{C} be the number of queries to the mark oracle and the challenge oracle, respectively. There are two types of conversion keys in our scheme [Wat09].

Normal: $\boldsymbol{k}_{\mathsf{tag}} = g^{(\alpha+r_1\mathsf{tag})\theta\vec{d}_1^*-r_1\theta\vec{d}_2^*+r_2\mathsf{tag}\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*+t_7\vec{d}_7^*+t_8\vec{d}_8^*}$

Semi-functional: $\boldsymbol{k}_{\mathsf{tag}} = g^{(\alpha+r_1\mathsf{tag})\theta\vec{d}_1^*-r_1\theta\vec{d}_2^*+r_2\mathsf{tag}\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*+t_5\vec{d}_5^*+t_6\vec{d}_6^*+t_7\vec{d}_7^*+t_8\vec{d}_8^*}$. We can generate semi-functional conversion keys if we have the secret key and $(g^{\vec{d}_5^*}, g^{\vec{d}_6^*})$.

Both types of conversion keys give a correct output (we can check this by simple calculation). To show that our scheme satisfies unforgeability, we introduce the following games: We consider game Game-i where the challenge oracle generates semi-functional conversion keys for the first $i \in [q_{\mathsf{C}}]$ queries and semi-functional conversion keys for the remaining $(q_{\mathsf{C}} - i)$ queries. Note that the mark oracle does not generate function indices. It only embeds marks for queried indices. Let $\mathsf{Adv}_i^{\mathsf{forge\text{-}N}}$ (resp. $\mathsf{Adv}_i^{\mathsf{forge\text{-}S}}$) denote the advantage of the adversary in Game-(i) for outputting a normal (resp. semi-functional) conversion key for a non-given or non-queried tag.

1. In Game-(0), the challenge oracles return normal conversion keys. First, We can show Lemma 4.7: If $\mathcal{A}$ outputs a valid semi-functional conversion key, then we can construct algorithm $\mathcal{B}_1$ (simulator for $\mathcal{A}$) that solves the subspace problem

2. Next, we consider Game-(i). We can show Lemma 4.8: If $\mathcal{A}$ detects the change of answers by the challenge oracle (from normal answer to semi-functional answer), we can construct algorithm $\mathcal{B}$ (simulator for $\mathcal{A}$) which solves the subspace problem.

3. Last, we consider Game-(q_{C}), where all answers of the challenge oracle to $\mathcal{A}$ are semi-functional conversion keys. We can show Lemma 4.9: If adversary $\mathcal{A}$ outputs a valid normal conversion key, then we can construct algorithm $\mathcal{B}_2$ which solves the subspace problem.

By Lemma 4.7, 4.8, and 4.9, we can show the following:

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Forge}}(\lambda) = \mathsf{Adv}_0^{\mathsf{forge\text{-}N}} + \mathsf{Adv}_0^{\mathsf{forge\text{-}S}} &< \mathsf{Adv}_0^{\mathsf{forge\text{-}N}} + q_{\mathsf{C}}\mathsf{Adv}_{\mathcal{B}}^{\mathsf{SubS}} + \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{SubS}} \\ &< \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{SubS}} + q_{\mathsf{C}}\mathsf{Adv}_{\mathcal{B}}^{\mathsf{SubS}} + \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{SubS}}\end{aligned}$$

Lemma 4.7 *If $\mathcal{A}$ outputs a semi-functional marked index in* Game-(0)*, then we can construct an algorithm that break the subspace assumption with $k = 2$ and $n = 8$.*

Lemma 4.8 *If there exists $\mathcal{A}$ that distinguishes* Game-$(i-1)$ *from* Game-(i)*, then we can construct an algorithm that break the subspace assumption with $k = 2$ and $n = 8$.*

Lemma 4.9 *If $\mathcal{A}$ outputs a normal marked index in* Game-(q_{C})*, then we can construct an algorithm that break the subspace assumption with $k = 1$ and $n = 8$.*

The theorem follows from these lemmas (the DLIN assumption implies the subspace assumption).

First, we give a proof of Lemma 4.7.

Proof of lemma. If $\mathcal{A}$ outputs $(1, ek^*)$ where $\mathsf{Detect}(dk, ek^*) \rightarrow \mathsf{marked}$ and $\mathsf{IdealDtc}(ek^*) \rightarrow \mathsf{unmarked}$, then we construct algorithm $\mathcal{B}$ which solves the subspace problem $k = 2$ and $n = 8$. $\mathcal{B}$ is given Γ, $D = (g^{\vec{b}_1}, \ldots, g^{\vec{b}_4}, g^{\vec{b}_7}, g^{\vec{b}_8}, g^{\eta\vec{b}_1^*}, g^{\eta\vec{b}_2^*}, g^{\beta\vec{b}_3^*}, g^{\beta\vec{b}_4^*}, g^{\vec{b}_5^*}, \ldots, g^{\vec{b}_8^*}, U_1 = g^{\mu_1\vec{b}_1+\mu_2\vec{b}_3+\mu_3\vec{b}_5}$, $U_2 = g^{\mu_1\vec{b}_2+\mu_2\vec{b}_4+\mu_3\vec{b}_6}, \mu_3)$, and Q_{b} where Q_{b} is $(V_1, V_2) = (g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_3^*}, g^{\tau_1\eta\vec{b}_2^*+\tau_2\beta\vec{b}_4^*})$ or $(W_1,$

$W_2) = (g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_3^*+\tau_3\vec{b}_5^*}, g^{\tau_1\eta\vec{b}_2^*+\tau_2\beta\vec{b}_4^*+\tau_3\vec{b}_6^*})$. For dual orthonormal bases $(\mathfrak{D}, \mathfrak{D}^*)$, we first consider dual orthonormal bases $(\mathfrak{F}, \mathfrak{F}^*)$ as follows:

$$\begin{array}{llllllll}
\vec{f}_1 := \eta\vec{b}_1^* & \vec{f}_2 := \eta\vec{b}_2^* & \vec{f}_3 := \beta\vec{b}_3^* & \vec{f}_4 := \beta\vec{b}_4^* & \vec{f}_5 := \vec{b}_5^* & \vec{f}_6 := \vec{b}_6^* & \vec{f}_7 := \vec{b}_7^* & \vec{f}_8 := \vec{b}_8^* \\
\vec{f}_1^* := \eta^{-1}\vec{b}_1 & \vec{f}_2^* := \eta^{-1}\vec{b}_2 & \vec{f}_3^* := \beta^{-1}\vec{b}_3 & \vec{f}_4^* := \beta^{-1}\vec{b}_4 & \vec{f}_5^* := \vec{b}_5 & \vec{f}_6^* := \vec{b}_6 & \vec{f}_7^* := \vec{b}_7 & \vec{f}_8^* := \vec{b}_8.
\end{array}$$

$\mathcal{B}$ chooses random matrix $A \in \mathbb{Z}_p^{2\times 2}$, which is invertible except negligible probability. Matrix $A \in \mathbb{Z}_p^{2\times 2}$ and $(A^{-1})^\top$ are applied to $\vec{f}_5, \vec{f}_6$ and $\vec{f}_5^*, \vec{f}_6^*$ as changes of basis matrix, respectively. That is, $\mathcal{B}$ sets $\vec{d}_i = \vec{f}_i$ and $\vec{d}_i^* = \vec{f}_i^*$ for $i = 1, \ldots, 4, 7, 8$ and it implicitly sets $\mathfrak{D} = \mathfrak{F}_A$, $\mathfrak{D}^* = \mathfrak{F}_A^*$. By Lemma 2.8, $(\mathfrak{D}, \mathfrak{D}^*)$ are correct dual orthonormal bases.

In order to set $\theta := \theta'\eta$, $\sigma := \sigma'\beta$ implicitly, $\mathcal{B}$ chooses $\alpha, \theta', \sigma' \xleftarrow{\cup} \mathbb{Z}_p$, and computes $pk = (e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*}, g^{\vec{d}_1}, \ldots, g^{\vec{d}_4}) := (e(g^{\vec{b}_1}, g^{\eta\vec{b}_1^*})^{\alpha\theta'}, g^{\eta\vec{b}_1^*}, g^{\eta\vec{b}_2^*}, g^{\beta\vec{b}_3^*}, g^{\beta\vec{b}_4^*})$ and $sk = (g^{\alpha\theta\vec{d}_1^*}, g^{\theta\vec{d}_1^*}, g^{\theta\vec{d}_2^*}, g^{\sigma\vec{d}_3^*}, g^{\sigma\vec{d}_4^*}) := (g^{\alpha\theta'\vec{b}_1}, g^{\theta'\vec{b}_1}, g^{\theta'\vec{b}_2}, g^{\sigma'\vec{b}_3}, g^{\sigma'\vec{b}_4})$ since it has $(g^{\eta\vec{b}_1^*}, g^{\eta\vec{b}_2^*}, g^{\beta\vec{b}_3^*}, g^{\beta\vec{b}_4^*})$, $(g^{\vec{b}_1}, \ldots, g^{\vec{b}_4})$, θ', and σ'. $\mathcal{B}$ has $dk = (g^{\vec{d}_7}, g^{\vec{d}_8})$, which is calculated by A and $(g^{\vec{b}_7^*}, g^{\vec{b}_8^*})$ and mark key $mk = (g^{\vec{d}_7^*}, g^{\vec{d}_8^*})$, which is calculated by A and $(g^{\vec{b}_7}, g^{\vec{b}_8})$ and can simulate the mark oracle since it has $(g^{\vec{b}_7}, g^{\vec{b}_8})$ and $(g^{\vec{b}_7^*}, g^{\vec{b}_8^*})$. Note that $\mathcal{B}$ does not have $g^{\vec{d}_5^*}$ and $g^{\vec{d}_6^*}$ since $g^{\vec{b}_5}$ and $g^{\vec{b}_6}$ are not given.

In order to simulate the challenge oracle, for tag, $\mathcal{B}$ chooses $r_1, r_2, t_7, t_8 \xleftarrow{\cup} \mathbb{Z}_p$ and computes $\widetilde{\boldsymbol{k}}_{\mathsf{tag}} := g^{(\alpha+r_1\mathsf{tag})\theta\vec{d}_1^*-r_1\theta\vec{d}_2^*+r_2\mathsf{tag}\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*+t_7\vec{d}_7^*+t_8\vec{d}_8^*}$. This is a valid marked index. Let $Q_{\mathsf{b}} = (T_1, T_2)$. If $\mathcal{A}$ outputs valid $ek^* = (\boldsymbol{U}^*, \boldsymbol{V}^*, \boldsymbol{k}_{\mathsf{tag}^*}^*)$ where

$$\boldsymbol{k}_{\mathsf{tag}^*}^* = g^{(\alpha+r_1^*\mathsf{tag}^*)\theta\vec{d}_1^*-\theta\vec{d}_2^*+r_2^*\mathsf{tag}^*\sigma^*\vec{d}_3^*-r_2^*\sigma^*\vec{d}_4^*+t_5^*\vec{d}_5^*+t_6^*\vec{d}_6^*+t_7^*\vec{d}_7^*+t_8^*\vec{d}_8^*},$$

then computes $C_0 := \boldsymbol{e}(T_1, g^{\vec{b}_1})^{\theta'\alpha} = e(g,g)^{\alpha\theta\tau_1\vec{d}_1\cdot\vec{d}_1^*}$, $C := T_1(T_2)^{\mathsf{tag}^*}$, and $\Delta := \boldsymbol{e}(C, \boldsymbol{k}_{\mathsf{tag}^*})$. In this game, we assume that $\mathcal{A}$ outputs a semi-functional marked index, so $(t_5^*, t_6^*) \neq \vec{0}$ and $(t_7^*, t_8^*) \neq \vec{0}$.

- If $T_1 = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_3^*} = g^{\tau_1\vec{d}_1+\tau_2\vec{d}_3}$ and $T_2 = g^{\tau_1\eta\vec{b}_2^*+\tau_2\beta\vec{b}_4^*} = g^{\tau_1\vec{d}_2+\tau_2\vec{d}_4}$, then it holds that $C = g^{\tau_1\vec{d}_1+\tau_1\mathsf{tag}\vec{d}_2+\tau_2\vec{d}_3+\tau_2\mathsf{tag}\vec{d}_4}$ and $\Delta := \boldsymbol{e}(C, \boldsymbol{k}_{\mathsf{tag}^*}^*) = e(g,g)^{\alpha\theta\tau_1\vec{d}_1\cdot\vec{d}_1^*}$.
- If $T_1 = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_3^*+\tau_3\vec{b}_5^*} = g^{\tau_1\vec{d}_1+\tau_2\vec{d}_3+\tau_3\vec{b}_5^*}$ and $T_2 = g^{\tau_1\eta\vec{b}_2^*+\tau_2\beta\vec{b}_4^*+\tau_3\vec{b}_6^*} = g^{\tau_2\vec{d}_1+\tau_2\vec{d}_4+\tau_3\vec{b}_6^*}$, then $C = g^{\tau_1\vec{d}_1+\tau_1\mathsf{tag}\vec{d}_2+\tau_2\vec{d}_3+\tau_2\mathsf{tag}\vec{d}_4+\tau_3\vec{b}_5^*+\tau_3\mathsf{tag}\vec{b}_6^*}$ and $\Delta := \boldsymbol{e}(C, \boldsymbol{k}_{\mathsf{tag}^*}^*) = e(g,g)^{\alpha\theta\tau_1\vec{d}_1\cdot\vec{d}_1^*+\gamma^*}$ where $\gamma^* \neq 0$.

In the latter case, the coefficient vector of $(\vec{b}_5^*, \vec{b}_6^*)$ is $(\tau_3, \mathsf{tag}\tau_3)$, thus the coefficient vector of $(\vec{d}_5, \vec{d}_6)$ is $\tau_3 A^{-1}(1, \mathsf{tag}^*)^\top$ and $\gamma = (t_5^*, t_6^*)\tau_3 A^{-1}(1, \mathsf{tag}^*)^\top$. These coefficients are uniformly random since $\mathcal{B}$ chose uniformly random A. If $\Delta/C_0 = 1$, then $\mathcal{B}$ outputs 0 ($\mathsf{b} = 0$). Else if $\Delta/C_0 \neq 1$, then it outputs 1. Thus, $\mathcal{B}$ can break the assumption. ■

Next, we prove Lemma 4.8.

Proof of lemma. If $\mathcal{A}$ outputs $(1, ek^*)$ where $\mathsf{Detect}(dk, ek^*) \to \mathsf{marked}$ and $\mathsf{IdealDtc}(ek^*) \to \mathsf{unmarked}$, then we construct algorithm $\mathcal{B}$ which solves the subspace problem $k = 2$ and $n = 8$. $\mathcal{B}$ is given Γ, $D = (g^{\vec{b}_1}, \ldots, g^{\vec{b}_4}, g^{\vec{b}_7}, g^{\vec{b}_8}, g^{\eta\vec{b}^*_1}, g^{\eta\vec{b}^*_2}, g^{\beta\vec{b}^*_3}, g^{\beta\vec{b}^*_4}, g^{\vec{b}^*_5}, \ldots, g^{\vec{b}^*_8}, U_1 = g^{\mu_1\vec{b}_1+\mu_2\vec{b}_3+\mu_3\vec{b}_5}$, $U_2 = g^{\mu_1\vec{b}_2+\mu_2\vec{b}_4+\mu_3\vec{b}_6}, \mu_3)$, and Q_{b} where Q_{b} is $(V_1, V_2) = (g^{\tau_1\eta\vec{b}^*_1+\tau_2\beta\vec{b}^*_3}, g^{\tau_1\eta\vec{b}^*_2+\tau_2\beta\vec{b}^*_4})$ or $(W_1$, $W_2) = (g^{\tau_1\eta\vec{b}^*_1+\tau_2\beta\vec{b}^*_3+\tau_3\vec{b}^*_5}, g^{\tau_1\eta\vec{b}^*_2+\tau_2\beta\vec{b}^*_4+\tau_3\vec{b}^*_6})$. For dual orthonormal bases $(\mathfrak{D}, \mathfrak{D}^*)$, we set

$$\begin{array}{llllll} \vec{d}_1 := \vec{b}_1 & \vec{d}_2 := \vec{b}_2 & \vec{d}_3 := \vec{b}_3 & \vec{d}_4 := \vec{b}_4 & \vec{d}_7 := \vec{b}_7 & \vec{d}_8 := \vec{b}_8 \\ \vec{d}^*_1 := \vec{b}^*_1 & \vec{d}^*_2 := \vec{b}^*_2 & \vec{d}^*_3 := \vec{b}^*_3 & \vec{d}^*_4 := \vec{b}^*_4 & \vec{d}^*_7 := \vec{b}^*_7 & \vec{d}^*_8 := \vec{b}^*_8 \end{array}$$

$\mathcal{B}$ chooses random matrix $A \in \mathbb{Z}_p^{2\times 2}$, which is invertible except negligible probability. Matrix A and $(A^{-1})^\top$ are applied as changes of basis matrix to $\vec{f}_5, \vec{f}_6$ and $\vec{f}^*_5, \vec{f}^*_6$, respectively, that is, it implicitly holds that $\mathfrak{D} := \mathfrak{B}_A$ and $\mathfrak{D}^* := \mathfrak{B}^*_A$ and they are correct distribution and reveal no information about A by Lemma 2.8. In order to set $\theta := \eta$ and $\sigma := \beta$ implicitly, $\mathcal{B}$ chooses $\alpha \xleftarrow{\cup} \mathbb{Z}_p$ and compute $e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}^*_1} := e(g^{\vec{b}_1}, g^{\eta\vec{b}^*_1})^\alpha$. Here it holds that $g^{\theta\vec{d}^*_1} = g^{\eta\vec{b}^*_1}$ and $g^{\sigma\vec{d}^*_3} = g^{\beta\vec{b}^*_3}$.

$\mathcal{B}$ can generate $pk = (e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}^*_1}, g^{\vec{d}_1}, \ldots, g^{\vec{d}_4}) := (g^{\vec{b}_1}, \ldots, g^{\vec{b}_4})$ and $sk = (g^{\alpha\theta\vec{d}^*_1}, g^{\theta\vec{d}^*_1}, g^{\theta\vec{d}^*_2}$, $g^{\sigma\vec{d}^*_3}, g^{\sigma\vec{d}^*_4}) := ((g^{\eta\vec{b}^*_1})^\alpha, g^{\eta\vec{b}^*_1}, g^{\eta\vec{b}^*_2}, g^{\beta\vec{b}^*_3}, g^{\beta\vec{b}^*_4})$. $\mathcal{B}$ has $mk = (g^{\vec{d}^*_7}, g^{\vec{d}^*_8})$, which is calculated by A and $(g^{\vec{b}^*_7}, g^{\vec{b}^*_8})$ and $dk = (g^{\vec{d}_7}, g^{\vec{d}_8})$, which is calculated by A and $((g^{\vec{b}_7}, g^{\vec{b}_8}))$ and can simulate the mark oracle since it has $(g^{\vec{b}^*_7}, g^{\vec{b}^*_8})$, $(g^{\vec{b}_7}, g^{\vec{b}_8})$, and α.

$\mathcal{B}$ can generate normal conversion keys by using sk and mk, so it returns normal conversion keys for j-th query where $i < j$. It can also compute semi-functional conversion key since it has $(g^{\vec{b}^*_5}, g^{\vec{b}^*_6})$ and can compute random linear combination of them, that is, $g^{t_5\vec{b}^*_5+t_6\vec{b}^*_6}$. It is equivalent to a vector spanned by $g^{\vec{d}^*_5}$ and $g^{\vec{d}^*_6}$. Thus, $\mathcal{B}$ returns semi-functional conversion keys for j-th query where $j < i$. Let $Q_{\mathsf{b}} = (T_1, T_2)$. In order to simulate i-th query, for tag_i, $\mathcal{B}$ chooses $t_7, t_8 \xleftarrow{\cup} \mathbb{Z}_p$ and computes $\widetilde{\boldsymbol{k}}_{\mathsf{tag}_i} := (g^{\eta\vec{b}^*_1})^\alpha T_1^{\mathsf{tag}_i}(T_2)^{-1} g^{t_7\vec{d}^*_7+t_8\vec{d}^*_8}$. Here, we can consider as $r_1 := \tau_1, r_2 := \tau_2$.

- If $T_1 = g^{\tau_1\eta\vec{b}^*_1+\tau_2\beta\vec{b}^*_3}$ and $T_2 = g^{\tau_1\eta\vec{b}^*_2+\tau_2\beta\vec{b}^*_4}$, then

$$\widetilde{\boldsymbol{k}}_{\mathsf{tag}_i} = g^{(\alpha+\tau_1\mathsf{tag}_i)\theta\vec{d}^*_1-\tau_1\theta\vec{d}^*_2+\tau_2\sigma\mathsf{tag}_i\vec{d}^*_3-\tau_2\sigma\vec{d}^*_4+t_7\vec{d}^*_7+t_8\vec{d}^*_8}.$$

  This is exactly Game-$(i-1)$.

- If $T_1 = g^{\tau_1\eta\vec{b}^*_1+\tau_2\beta\vec{b}^*_3+\tau_3\vec{b}^*_5}$ and $T_2 = g^{\tau_1\eta\vec{b}^*_2+\tau_2\beta\vec{b}^*_4+\tau_3\vec{b}^*_6}$, then

$$\widetilde{\boldsymbol{k}}_{\mathsf{tag}_i} = g^{(\alpha+\tau_1\mathsf{tag}_i)\theta\vec{d}^*_1-\tau_1\theta\vec{d}^*_2+\tau_2\sigma\mathsf{tag}_i\vec{d}^*_3-\tau_2\sigma\vec{d}^*_4+\mathsf{tag}_i\tau_3\vec{b}^*_5-\tau_3\vec{b}^*_6+t_7\vec{d}^*_7+t_8\vec{d}^*_8}.$$

  This is exactly Game-(i).

In the latter case, the coefficient vector of $\vec{b}^*_5$ and $\vec{b}^*_6$ is $\vec{X} := (\mathsf{tag}_i\tau_3, -\tau_3)$. If $\mathcal{A}$ outputs valid $ek^* = (\boldsymbol{U}^*, \boldsymbol{V}^*, \boldsymbol{k}^*_{\mathsf{tag}^*})$ where

$$\boldsymbol{k}^*_{\mathsf{tag}^*} = g^{(\alpha+r^*_1\mathsf{tag}^*)\theta\vec{d}^*_1-\theta\vec{d}^*_2+r^*_2\mathsf{tag}^*\sigma^*\vec{d}^*_3-r^*_2\sigma^*\vec{d}^*_4+t^*_5\vec{d}^*_5+t^*_6\vec{d}^*_6+t_7\vec{d}^*_7+t_8\vec{d}^*_8}$$

(maybe $(t_5^*, t_6^*) = \vec{0}$), then $\mathcal{B}$ computes $C := U_1(U_2)^{\mathsf{tag}^*} = g^{\mu_1\vec{d}_1+\mu_1\mathsf{tag}^*\vec{d}_2+\mu_2\vec{d}_3+\mu_2\mathsf{tag}^*\vec{d}_4+\mu_3\vec{b}_5+\mu_3\mathsf{tag}^*\vec{b}_6}$ and $C_0 := (\boldsymbol{e}(g^{\eta\vec{b}_1^*}, U_1))^\alpha = (e(g,g)^{\alpha\theta\vec{d}_1^*})^{\mu_1}$. The coefficient vector of $(\vec{b}_5, \vec{b}_6)$ is $\vec{Y} := (\mu_3, \mathsf{tag}^*\mu_3)$. The adversary must output a *new* function index and make it marked, so tag^* is different from all tag_i which are given (resp. queried) from the challenge oracle (resp. to the mark oracle). The coefficient vector of $(\vec{d}_5^*, \vec{d}_6^*)$ and $(\vec{d}_5, \vec{d}_6)$ are $\vec{X}' := \tau_3 A^\top(\mathsf{tag}_i, -1)^\top$ and $\vec{Y}' := \mu_3 A^{-1}(1, \mathsf{tag}^*)$, respectively. The distribution of all values except $\widetilde{\boldsymbol{k}}_{\mathsf{tag}_i}$ and (C_0, C) is independent of transformation matrix A and $\mathsf{tag}^* \neq \mathsf{tag}_i$, so coefficient $\vec{X}'$ and $\vec{Y}'$ are uniformly random by Lemma 2.8. $\mathcal{B}$ computes $\Delta := \boldsymbol{e}(C, \boldsymbol{k}_{\mathsf{tag}^*})$.

If $t_5^* = t_6^* = 0$ (i.e., normal conversion key) and $\Delta/C_0 = 1$, then $\mathcal{B}$ outputs 0. If $(t_5^*, t_6^*) \neq \vec{0}$ (i.e., semi-functional conversion key) and $\Delta/C_0 \neq 1$, then $\mathcal{B}$ outputs 1. Thus, if $\mathcal{A}$ detects the change, that is, if there is non-negligible difference between that $\mathcal{A}$ outputs a normal conversion key and a semi-functional conversion key, then $\mathcal{B}$ can break the subspace assumption. ■

Last, we prove Lemma 4.9.

Proof of lemma. If $\mathcal{A}$ outputs $(1, ek^*)$ where $\mathsf{Detect}(dk, ek^*) \to \mathsf{marked}$ and $\mathsf{IdealDtc}(ek^*) \to \mathsf{unmarked}$, then we construct algorithm $\mathcal{B}$ which solves Subspace problem $k = 1$ and $n = 8$. $\mathcal{B}$ is given Γ, $D = (g^{\vec{b}_1}, g^{\vec{b}_2}, g^{\vec{b}_4}, \ldots, g^{\vec{b}_8}, g^{\eta\vec{b}_1^*}, g^{\beta\vec{b}_2^*}, g^{\vec{b}_3^*}, \ldots, g^{\vec{b}_8^*}, U_1 = g^{\mu_1\vec{b}_1+\mu_2\vec{b}_2+\mu_3\vec{b}_3}, \mu_3)$, and Q_b where Q_b is $V_1 = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*}$ or $W_1 = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*+\tau_3\vec{b}_3^*}$. For dual orthonormal bases $(\mathfrak{D}, \mathfrak{D}^*)$, we set

$$\begin{array}{llllllll}
\vec{d}_1 := \vec{b}_3^* & \vec{d}_2 := \vec{b}_4^* & \vec{d}_3 := \vec{b}_5^* & \vec{d}_4 := \vec{b}_6^* & \vec{d}_5 := \vec{b}_1^* & \vec{d}_6 := \vec{b}_2^* & \vec{d}_7 := \vec{b}_7^* & \vec{d}_8 := \vec{b}_8^* \\
\vec{d}_1^* := \vec{b}_3 & \vec{d}_2^* := \vec{b}_4 & \vec{d}_3^* := \vec{b}_5 & \vec{d}_4^* := \vec{b}_6 & \vec{d}_5^* := \vec{b}_1 & \vec{d}_6^* := \vec{b}_2 & \vec{d}_7^* := \vec{b}_7 & \vec{d}_8^* := \vec{b}_8.
\end{array}$$

$\mathcal{B}$ chooses $\theta, \alpha', \sigma \xleftarrow{\cup} \mathbb{Z}_p$, and can compute $pk = (e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*}, g^{\vec{d}_1}, \ldots, g^{\vec{d}_4}) := (\boldsymbol{e}(g^{\vec{b}_4^*}, g^{\vec{b}_4})^{\alpha'\mu_3\theta}, g^{\vec{b}_3^*}, g^{\vec{b}_4^*}, g^{\vec{b}_5^*}, g^{\vec{b}_6^*})$ where $\alpha := \alpha'\mu_3$, $dk = (g^{\vec{d}_7}, g^{\vec{d}_8}) := (g^{\vec{b}_7^*}, g^{\vec{b}_8^*})$ and $mk = (g^{\vec{d}_7^*}, g^{\vec{d}_8^*}) := (g^{\vec{b}_7}, g^{\vec{b}_8})$ (i.e., can simulate the mark oracle), but does not have $g^{\vec{d}_1^*}$ since $g^{\vec{b}_3}$ is not given. That is, it has incomplete $sk = (g^{\alpha\theta\vec{d}_1^*}, g^{\theta\vec{d}_1^*}, g^{\theta\vec{d}_2^*}, g^{\sigma\vec{d}_3^*}, g^{\sigma\vec{d}_4^*}) := (\perp, \perp, g^{\theta\vec{b}_4}, g^{\sigma\vec{b}_5}, g^{\sigma\vec{b}_6})$. $\mathcal{B}$ also has $(g^{\vec{d}_5^*}, g^{\vec{d}_6^*}) = (g^{\vec{b}_1}, g^{\vec{b}_2})$. Here, $U_1 = g^{\mu_1\vec{d}_5^*+\mu_2\vec{d}_6^*+\mu_3\vec{d}_1^*}$ and we use it to simulate conversion keys without the complete secret key. In order to simulate the challenge oracle, for tag, $\mathcal{B}$ chooses $r_1', r_2, t_5', t_6', t_7, t_8 \xleftarrow{\cup} \mathbb{Z}_p$ and computes

$$\begin{aligned}
\widetilde{\boldsymbol{k}}_\mathsf{tag} &:= (U_1)^{(\alpha'-r_1'\mathsf{tag})\theta} g^{-r_1'\mu_3\theta\vec{d}_2^*+r_2\mathsf{tag}\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*+t_5'\vec{d}_5^*+t_6'\vec{d}_6^*+t_7\vec{d}_7^*+t_8\vec{d}_8^*} \\
&= g^{(\alpha+r_1\mathsf{tag})\theta\vec{d}_1^*-r_1\theta\vec{d}_2^*+r_2\mathsf{tag}\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*+(t_5'-\theta(\alpha'+r_1'\mathsf{tag})\mu_1)\vec{d}_5^*+(t_6'-\theta(\alpha'+r_1'\mathsf{tag})\mu_2)\vec{d}_6^*+t_7\vec{d}_7^*+t_8\vec{d}_8^*}.
\end{aligned}$$

It implicitly holds $r_1 := \mu_3 r_1'$. This is a valid marked semi-functional conversion key. If $\mathcal{A}$ outputs normal conversion key $ek^* = (\boldsymbol{U}^*, \boldsymbol{V}^*, \boldsymbol{k}_{\mathsf{tag}^*}^*)$ where

$$\boldsymbol{k}_{\mathsf{tag}^*}^* = g^{(\alpha+r_1^*\mathsf{tag}^*)\theta\vec{d}_1^*-r_1^*\theta\vec{d}_2^*+r_2^*\mathsf{tag}^*\sigma^*\vec{d}_3^*-r_2^*\sigma^*\vec{d}_4^*+t_7^*\vec{d}_7^*+t_8^*\vec{d}_8^*},$$

then $\mathcal{B}$ chooses $s_1, s_2 \xleftarrow{\cup} \mathbb{Z}_p$ and computes $C := g^{s_1\vec{d}_1+s_1\mathsf{tag}^*\vec{d}_2+s_2\vec{d}_3+s_2\mathsf{tag}^*\vec{d}_4}$, $C_0 := e(g,g)^{s_1\alpha\theta\vec{b}_4\cdot\vec{b}_4^*}$, and $\Delta := \boldsymbol{e}(C\cdot T, \boldsymbol{k}^*_{\mathsf{tag}^*})$. If $T = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*} = g^{\tau_1\eta\vec{d}_5+\tau_2\beta\vec{d}_6}$, then $\Delta/C_0 = 1$. If $T = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*+\tau_3\vec{b}_3^*} = g^{\tau_1\eta\vec{d}_5+\tau_2\beta\vec{d}_6+\tau_3\vec{d}_1}$, then $\Delta/C_0 = e(g,g)^{\tau_3(\alpha+r_1^*\mathsf{tag}^*)\theta\vec{d}_1\cdot\vec{d}_1^*} \neq 1$. Thus, $\mathcal{B}$ can break the subspace assumption. ■

Thus, we finished the proof of Theorem 4.6 □

As we see in the proof of Theorem 4.6 above, even if forgeries by the adversary do not include marks, the proof holds. That is, the adversary cannot output an un-marked index.

5 Proposed Scheme Based on Okamoto-Takashima scheme

We can obtain a LTF and its watermarking scheme by using Okamoto-Takashima IPE scheme based on DPVS (with additive notation) [OT10, OT12]. The construction idea is the same as that of $\mathsf{LTF}_{\mathsf{mult}}$ and $\mathsf{CWM}_{\mathsf{mult}}$. In this section, we present a scheme based on Okamoto-Takashima IPE scheme.

5.1 LTF based on Okamoto-Takashima IPE scheme

We propose our LTF construction, $\mathsf{LTF}_{\mathsf{add}} := (\mathsf{LTF.IGen}, \mathsf{LTF.LGen}, \mathsf{LTF.Eval}, \mathsf{LTF.Invert})$ based on Okamoto-Takashima IPE [OT10, OT12]. Instead of using the inner-product predicate, we use the equality-test predicate for simplicity.

$\mathsf{LTF.IGen}(1^\lambda)$ **:** It generates $(\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{R} \mathcal{G}_{\mathsf{ob}}(1^\lambda, 10)$ (See Section 2.3 for $\mathcal{G}_{\mathsf{ob}}$), chooses $\boldsymbol{\psi} \xleftarrow{\cup} \mathbb{F}_q^\ell$, and sets $\widehat{\mathbb{B}} := (\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_9)$ and $g_{T_j} := \boldsymbol{e}(\boldsymbol{b}_0, \boldsymbol{b}_0^*)^{\psi_j}$ for all $j \in [\ell]$. It chooses $\zeta_i, \varphi_i \xleftarrow{\cup} \mathbb{F}_q$ and $\omega_i \xleftarrow{\cup} \mathbb{F}_q$ for all $i \in [\ell]$ and generates $u_{i,j} := g_{T_j}^{\zeta_i} \cdot g_T^{m_{i,j}}$ and $\boldsymbol{v}_i := (\zeta_i, \omega_i(1, \mathsf{tag}), 0^6, \varphi_i)_{\mathbb{B}}$ for all $i, j \in [\ell]$ where $m_{i,j} = 1$ if $i = j$ and $m_{i,j} = 0$ otherwise. For a conversion key, it chooses $\sigma \xleftarrow{\cup} \mathbb{F}_q$, $\vec{\eta} \xleftarrow{\cup} \mathbb{F}_q^2$ and generates $\boldsymbol{k}_{\mathsf{tag}} := (1, \sigma(\mathsf{tag}, -1), 0^2, \vec{\eta}, 0^2, 0)_{\mathbb{B}^*}$. It returns $ek := (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{\mathsf{tag}}) := (\{u_{i,j}\}_{i,j}, \{\boldsymbol{v}_i\}_i, \boldsymbol{k}_{\mathsf{tag}})$ $(i, j = 1, \ldots, \ell)$, $ik := \boldsymbol{\psi}$.

$\mathsf{LTF.LGen}(1^\lambda)$ **:** This is the same as $\mathsf{LTF.IGen}$ except that for all $i, j \in [\ell]$, $m_{i,j} = 0$ and $ik := \perp$.

$\mathsf{LTF.Eval}(ek, \vec{x})$**:** For input $\vec{x} \in \{0,1\}^\ell$, it computes

$$
\begin{aligned}
y_j &:= \prod_i u_{i,j}^{x_i} &&= \prod_i g_{T_j}^{x_i\zeta_i} \cdot g_T^{x_i m_{i,j}} && = g_{T_j}^{\langle\vec{x},\vec{\zeta}\rangle} g_T^{x_j} \\
y_{\ell+1} &:= \sum_i x_i \boldsymbol{v}_i &&= \sum_i (x_i\zeta_i, x_i\omega_i(1,\mathsf{tag}), 0^6, x_i\varphi_i)_{\mathbb{B}} \\
& && = (\langle\vec{x},\vec{\zeta}\rangle, \langle\vec{x},\vec{\omega}\rangle(1,\mathsf{tag}), 0^6, \langle\vec{x},\vec{\varphi}\rangle)_{\mathbb{B}}
\end{aligned}
$$

where $\vec{\zeta} := (\zeta_1, \ldots, \zeta_\ell)$, $\vec{\omega} := (\omega_1, \ldots, \omega_\ell)$, and $\vec{\varphi} := (\varphi_1, \ldots, \varphi_\ell)$, and $y'_{\ell+1} := \boldsymbol{e}(y_{\ell+1}, \boldsymbol{k}_0) = \boldsymbol{e}(\boldsymbol{b}_0, \boldsymbol{b}_0^*)^{\langle\vec{x},\vec{\zeta}\rangle}$ and returns output $\boldsymbol{y} := (y_1, \ldots, y_\ell, y'_{\ell+1})$.

$\mathsf{LTF.Invert}(ik, \boldsymbol{y})$**:** For input $\boldsymbol{y}$, it computes $x'_j := y_j/(y'_{\ell+1})^{\psi_j} = g_{T_j}^{\langle \vec{x}, \vec{\zeta} \rangle} g_T^{x_j} / g_T^{\langle \vec{x}, \vec{\zeta} \rangle \cdot \psi_j}$ and let $x_j \in \{0,1\}$ be such that $x'_j = g_T^{x_j}$. It returns $\vec{x} = (x_1, \ldots, x_\ell)$.

Theorem 5.1 $\mathsf{LTF}_{\mathsf{dpvs}}$ *is a lossy trapdoor function if the DBDH assumption holds.*

The proof is followed from the next two lemmas.

Lemma 5.2 (Lossiness of $\mathsf{LTF}_{\mathsf{add}}$**)** $\mathsf{LTF}_{\mathsf{add}}$ *is* $(\ell - \log q)$*-lossy.*

Proof. We compute lossiness ℓ'. For a lossy function index generated by $\mathsf{LTF.LGen}$, every output $\boldsymbol{y} = (g_T^{\zeta' \boldsymbol{\psi}}, e(G,G)^{\langle \vec{x}, \vec{\zeta} \rangle}) = (g_{T_1}^{\zeta'}, \ldots, g_{T_\ell}^{\zeta'}, e(G,G)^{\zeta'})$ where $\zeta' = \langle \vec{x}, \vec{\zeta} \rangle \in \mathbb{F}_q$. Here, $\boldsymbol{\psi}$ is fixed by the function index. For any given image $\boldsymbol{y}$, there are at least q possible values for $\langle \vec{x}, \vec{\zeta} \rangle$ and pre-images. Therefore, $|\mathsf{D}|\,/2^{\ell'} = q$, equivalently $\ell' = \ell - \log q$ since $|\mathsf{D}| = 2^\ell$. □

Lemma 5.3 (Indistinguishability of $\mathsf{LTF}_{\mathsf{add}}$**)** *If the DBDH assumption holds, then* $\mathsf{LTF}_{\mathsf{add}}$ *satisfies indistinguishability.*

Proof. This proof is essentially the same as that of $\mathsf{LTF}_{\mathsf{mult}}$. The difference is as follows: In the simulation, we can replace $(\ell+1)$-th column element g_T^ζ with $(\zeta, \omega(1, \mathsf{tag}), 0^6, \varphi)_{\mathbb{B}}$ where $(\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{ob}}(1^\lambda, 10)$ and $\widehat{\mathbb{B}} := (\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_9)$. We should simulate $\boldsymbol{V} = (\zeta, \omega(1, \mathsf{tag}), 0^6, \varphi)_{\mathbb{B}} = \zeta \boldsymbol{b}_0 + \omega \boldsymbol{b}_1 + \omega \mathsf{tag} \boldsymbol{b}_2 + \varphi \boldsymbol{b}_9$ and $\boldsymbol{k}_{\mathsf{tag}} = (1, \sigma(\mathsf{tag}, -1), 0^4, \vec{\eta}, 0)_{\mathbb{B}^*}$ for $\vec{\eta} \xleftarrow{\mathsf{U}} \mathbb{F}_q^2$ and $\omega, \varphi \xleftarrow{\mathsf{U}} \mathbb{F}_q$. If we have $\zeta G = cG$ and matrix $X = (\chi_{i,j})$ behind $(\mathbb{B}, \mathbb{B}^*)$, then we can compute $\zeta \boldsymbol{b}_0$ without knowing ζ since simulator $\mathcal{B}$ generates $(\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{ob}}(1^\lambda, 10)$ by itself and has matrix $\boldsymbol{X}$ and $\boldsymbol{b}_0 = \sum_{j=0}^{9} \chi_{0,j} \boldsymbol{a}_j = (\chi_{0,0} G, \ldots, \chi_{0,9} G)$, and compute $\zeta \chi_{0,j} G = \chi_{0,j}(\zeta G)$ for any $j \in \{0, \ldots, 9\}$. $\mathcal{B}$ can also generate $\boldsymbol{k}_{\mathsf{tag}}$ since it has $\mathbb{B}^*$. Therefore, in the above simulation we can replace $(\ell+1)$-th column element g_T^ζ with $(\zeta, \omega(1, \mathsf{tag}), 0^6, \varphi)_{\mathbb{B}}$ and add $\boldsymbol{k}_0 = (1, \sigma(\mathsf{tag}, -1), 0^4, \vec{\eta}, 0)_{\mathbb{B}^*}$. Therefore, $\mathsf{LTF}_{\mathsf{add}}$ satisfies indistinguishability. □

5.2 Watermarking Scheme for $\mathsf{LTF}_{\mathsf{add}}$

Next, we present watermarking scheme $\mathsf{CWM}_{\mathsf{add}}$ for LTF based on Okamoto-Takashima IPE.

$\mathsf{WMGen}(\mathsf{LTF}_{\mathsf{add}})$**:** It generates $(\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{ob}}(1^\lambda, 10)$. It sets $\widehat{\mathbb{B}} := (\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_9)$, $pk := (\mathsf{param}_{\mathbb{V}}, \widehat{\mathbb{B}}, \mathsf{Samp})$, $sk := (\boldsymbol{b}_0^*, \boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \boldsymbol{b}_5^*, \boldsymbol{b}_6^*)$, $mk := (\boldsymbol{b}_7^*, \boldsymbol{b}_8^*)$, $dk := (\boldsymbol{b}_7, \boldsymbol{b}_8)$, and $rk := \psi(\boldsymbol{X}^\top)^{-1} = (\vartheta_{i,j})$. Sampling algorithm $\mathsf{Samp}(\mathsf{param}_{\mathbb{V}}, \widehat{\mathbb{B}}, sk)$ chooses $\boldsymbol{\psi} \xleftarrow{\mathsf{U}} \mathbb{F}_q^\ell$, $\vec{\zeta}, \vec{\omega}, \vec{\varphi} \xleftarrow{\mathsf{U}} \mathbb{F}_q^\ell$, and generates $(ek, ik) := ((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{\mathsf{tag}}), \boldsymbol{\psi})$ as $\mathsf{LTF.IGen}$ (or $\mathsf{LTF.LGen}$). Note that sk is not included in the description of Samp. Keys sk, mk, and rk are secret. Key dk can be disclosed.

$\mathsf{Mark}(mk, ek)$**:** It parses $ek = (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{\mathsf{tag}}) = \big(\boldsymbol{U}, \boldsymbol{V}, (1, \sigma(\mathsf{tag}, -1), 0^2, \eta, 0^2, 0)_{\mathbb{B}^*}\big)$, chooses $\mu_7, \mu_8 \xleftarrow{\mathsf{U}} \mathbb{F}_q$, and computes $\widetilde{\boldsymbol{k}}_{\mathsf{tag}} := \boldsymbol{k}_{\mathsf{tag}} + (0^7, \mu_7, \mu_8, 0)_{\mathbb{B}^*}$ by using $(\boldsymbol{b}_7^*, \boldsymbol{b}_8^*)$. It outputs marked function index $\mathsf{WM}(ek) = (\boldsymbol{U}, \boldsymbol{V}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}})$.

$\mathsf{Detect}(dk, \widetilde{ek})$**:** It parses $\widetilde{ek} = (\boldsymbol{U}, \boldsymbol{V}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}})$, chooses $\delta_7, \delta_8 \xleftarrow{\mathsf{U}} \mathbb{F}_q$, and computes $\boldsymbol{c} := \delta_7 \boldsymbol{b}_7 + \delta_8 \boldsymbol{b}_8$. Next, computes $\Delta := \boldsymbol{e}(\boldsymbol{c}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}})$. If it holds that $\Delta = \boldsymbol{e}(\boldsymbol{c}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}}) \neq 1$, then outputs marked. Else if, it holds that $\Delta = 1$, then outputs unmarked. *NOTE:* If we allow public detection, we publish detection key $\delta_7 \boldsymbol{b}_7 + \delta_8 \boldsymbol{b}_8$.

$\mathsf{Remove}(rk, \widetilde{ek})$**:** It parses $rk = (\psi(\boldsymbol{X}^\top)^{-1})$ and $\widetilde{ek} = (\boldsymbol{U}, \boldsymbol{V}, \widetilde{\boldsymbol{k}}_{\mathsf{tag}})$ and runs the decomposition algorithm. That is, computes $\sum_{j=0}^{m} z_j \boldsymbol{b}_j = \mathsf{Decomp}(\widetilde{\boldsymbol{k}}_{\mathsf{tag}}, (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_m), \psi(\boldsymbol{X}^\top)^{-1}, \mathbb{B}^*)$ for all $m < 9$, where $z_j \in \mathbb{F}_q$ and obtains $z_j \boldsymbol{b}_j^*$ for all $j = 0, \ldots, 9$. It computes $\boldsymbol{k}'_{\mathsf{tag}} := \widetilde{\boldsymbol{k}}_{\mathsf{tag}} - (z_7 \boldsymbol{b}_7^* + z_8 \boldsymbol{b}_8^*)$ and outputs $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}'_{\mathsf{tag}})$ as an unmarked index.

We can easily understand that meaningfulness, correctness, and polynomial blowup hold. Preserving functionality holds since elements in $\boldsymbol{U}$ and $\boldsymbol{V}$ do not include vectors $\boldsymbol{b}_7$ and $\boldsymbol{b}_8$ and $\mu_7 \boldsymbol{b}_7^* + \mu_8 \boldsymbol{b}_8^*$ does not interfere the computation of $\mathsf{LTF.Eval}$. We can prove the security of this scheme as the proof of $\mathsf{CWM}_{\mathsf{mult}}$.

5.3 Security Proofs for $\mathsf{CWM}_{\mathsf{add}}$

Theorem 5.4 *Our watermarking scheme* $\mathsf{CWM}_{\mathsf{add}}$ *is secure watermarking scheme under the DLIN assumption.*

We introduce some assumptions that are implied by the DLIN assumption to prove the security of $\mathsf{CWM}_{\mathsf{add}}$.

5.3.1 New Complexity Problems from DLIN.

In this section, we introduce new complexity problems, Another Basic Problem 0, 1, and 2 (ABP0, ABP1, ABP2) based on the DLIN problem to prove the security of the additive version of our scheme. The new problems can be considered as a variants of Basic Problem 0 (BP0) introduced by Okamoto and Takashima [OT10].

First, we review some complexity problems over DPVS that are reduced to the DLIN problem [OT10, OT12].

Definition 5.5 (Basic Problem 0 [OT10]) Basic Problem 0 (BP0) is to guess β, given $(\mathsf{param}_{\mathsf{BP0}},$

$\widehat{\mathbb{B}}, \mathbb{B}^*, \boldsymbol{y}_\beta, \boldsymbol{f}, \kappa G, \xi G, \delta\xi G) \xleftarrow{R} \mathcal{G}^{\mathsf{BP0}}(1^\lambda)$, where $\mathcal{G}^{\mathsf{BP0}}(1^\lambda)$:

$\mathsf{param}_{\mathbb{G}} := (q, \mathbb{G}, \mathbb{G}_T, G, e) \xleftarrow{R} \mathcal{G}_{\mathsf{bmp}}(1^\lambda)$, $\mathsf{param}_{\mathbb{V}} := (q, \mathbb{V}, \mathbb{G}_T, \mathbb{A}, e) \xleftarrow{R} \mathcal{G}_{\mathsf{dpvs}}(1^\lambda, 3, \mathsf{param}_{\mathbb{G}})$,

$X := \begin{pmatrix} \vec{\chi}_1 \\ \vec{\chi}_2 \\ \vec{\chi}_3 \end{pmatrix} := (\chi_{i,j})_{i,j} \xleftarrow{U} GL(3, \mathbb{F}_q)$, $(\vartheta_{i,j})_{i,j} := \begin{pmatrix} \vec{\vartheta}_1 \\ \vec{\vartheta}_2 \\ \vec{\vartheta}_3 \end{pmatrix} := (X^\top)^{-1}$,

$\kappa, \xi \xleftarrow{U} \mathbb{F}_q^\times$, $g_T := e(G, G)^{\kappa\xi}$,

$\boldsymbol{b}_i := \kappa(\vec{\chi}_i)_{\mathbb{A}} = \kappa \sum_{j=1}^3 \chi_{i,j} \boldsymbol{a}_j (i = 1, 3)$, $\widehat{\mathbb{B}} := (\boldsymbol{b}_1, \boldsymbol{b}_3)$,

$\boldsymbol{b}_i^* := \xi(\vec{\vartheta}_i)_{\mathbb{A}} = \xi \sum_{j=1}^3 \vartheta_{i,j} \boldsymbol{a}_j (i = 1, 2, 3)$, $\mathbb{B}^* := (\boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \boldsymbol{b}_3^*)$,

$\mathsf{param}_{\mathsf{BP0}} := (\mathsf{param}_{\mathbb{V}}, g_T)$, $\delta, \sigma, \omega \xleftarrow{U} \mathbb{F}_q,\ \rho, \tau \xleftarrow{U} \mathbb{F}_q^\times$,

$\boldsymbol{y}_0^* := (\delta, 0, \sigma)_{\mathbb{B}^*}$, $\boldsymbol{y}_1^* := (\delta, \rho, \sigma)_{\mathbb{B}^*}$,

$\boldsymbol{f} := (\omega, \tau, 0)_{\mathbb{B}}$,

return $(\mathsf{param}_{\mathsf{BP0}}, \widehat{\mathbb{B}}, \mathbb{B}^*, \boldsymbol{y}_\beta, \boldsymbol{f}, \kappa G, \xi G, \delta\xi G)$

Lemma 5.6 ([OT10]) *For any adversary $\mathcal{B}$, there is PPT machine $\mathcal{D}$, whose running time is essentially the same as that of $\mathcal{D}$, such that for any security parameter λ, $\mathsf{Adv}^{\mathsf{BP0}}_{\mathcal{B}}(\lambda) \leq \mathsf{Adv}^{\mathsf{DLIN}}_{\mathcal{D}}(\lambda) + 5/q$.*

Definition 5.7 (Problem 1 [OT10, OT12]) Problem 1 is to guess β, given $(\mathsf{param}, \mathbb{B}, \widehat{\mathbb{B}}^*, \boldsymbol{e}_{\beta,1}, \{\boldsymbol{e}_i\}_{i=2,\ldots,n}) \xleftarrow{R} \mathcal{G}^{\mathsf{P1}}_\beta(1^\lambda, n)$ where

$$\begin{aligned}
\mathcal{G}^{\mathsf{P1}}_\beta(1^\lambda, n) :& (\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{R} \mathcal{G}_{\mathsf{ob}}(1^\lambda, 4n+2) \\
& \widehat{\mathbb{B}}^* := (\boldsymbol{b}_0^*, \boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*, \boldsymbol{b}_{2n+1}^*, \ldots, \boldsymbol{b}_{4n+1}^*), \omega, \gamma \xleftarrow{U} \mathbb{F}_q, \vec{z} \xleftarrow{U} \mathbb{F}_q^2 \\
& \boldsymbol{e}_{0,1} := (0, \omega\vec{e}_1, 0^{2n}, 0^n, \gamma)_{\mathbb{B}}, \\
& \boldsymbol{e}_{1,1} := (0, \omega\vec{e}_1, \vec{z}, 0^n, 0^n, \gamma)_{\mathbb{B}}, \\
& \boldsymbol{e}_i := \omega \boldsymbol{b}_i, i = 2, \ldots, n,
\end{aligned}$$

return $(\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \widehat{\mathbb{B}}^*, \boldsymbol{e}_{\beta,1}, \{\boldsymbol{e}_i\}_{i=2,\ldots,n})$.

Lemma 5.8 ([OT10, OT12]) *For any adversary $\mathcal{B}$, there is PPT machine $\mathcal{D}$, whose running time is essentially the same as that of $\mathcal{B}$, such that for any security parameter λ, $\mathsf{Adv}^{\mathsf{P1}}_{\mathcal{B}}(\lambda) \leq \mathsf{Adv}^{\mathsf{DLIN}}_{\mathcal{D}}(\lambda) + 6/q$.*

Definition 5.9 (Problem 2 [OT12]) Problem 2 is to guess β, given $(\mathsf{param}, \widehat{\mathbb{B}}, \mathbb{B}^*, \{\boldsymbol{h}^*_{\beta,i}, \boldsymbol{e}_i\}_{i=1,\ldots,n}) \xleftarrow{R} \mathcal{G}^{\mathsf{P2}}_\beta(1^\lambda, n)$ where

$$
\begin{aligned}
\mathcal{G}^{\mathsf{P2}}_\beta(1^\lambda, n) :& (\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{R} \mathcal{G}_{\mathsf{ob}}(1^\lambda, 4n+2) \\
& \widehat{\mathbb{B}} := (\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_n, \boldsymbol{b}_{2n+1}, \boldsymbol{b}_{4n+1}), \omega, \tau, \delta, \delta_0, \sigma \xleftarrow{U} \mathbb{F}_q, \\
& i = 1, \ldots, n, \vec{e}_i := (0^{i-1}, 1, 0^{n-i}), \\
& \boldsymbol{h}^*_{0,i} := (0, \delta\vec{e}_i, 0^n, 0^n, \delta_0\vec{e}_i, 0)_{\mathbb{B}^*}, \\
& \boldsymbol{h}^*_{1,i} := (0, \delta\vec{e}_i, \tau\vec{e}_i, 0^n, \delta_0\vec{e}_i, 0)_{\mathbb{B}^*}, \\
& \boldsymbol{e}_i := (0, \omega\vec{e}_i, \sigma\vec{e}_i, 0^n, 0^n, 0)_{\mathbb{B}},
\end{aligned}
$$

return $(\mathsf{param}_{\mathbb{V}}, \widehat{\mathbb{B}}, \mathbb{B}^*, \{\boldsymbol{h}^*_{\beta,i}, \boldsymbol{e}_i\}_{i=1,\ldots,n})$.

Lemma 5.10 ([OT12]) *For any adversary $\mathcal{B}$, there is PPT machine $\mathcal{D}$, whose running time is essentially the same as that of $\mathcal{B}$, such that for any security parameter λ, $\mathsf{Adv}^{\mathsf{P2}}_{\mathcal{B}}(\lambda) \leq \mathsf{Adv}^{\mathsf{DLIN}}_{\mathcal{D}}(\lambda) + 5/q$.*

Next, we introduce ABP0. ABP0 seems to be somewhat stronger than BP0 since a coefficient for $\boldsymbol{b}_2$ in $\boldsymbol{f}$ is given to the adversary. However, we can show that if the DLIN problem is hard, then ABP0 is also hard. This is proved by a technique that is similar to that of Okamoto and Takashima.

Definition 5.11 (Another Basic Problem 0) Another Basic Problem 0 (ABP0) is to guess $\beta \in \{0,1\}$, given $(\mathsf{param}_{\mathsf{ABP0}}, \widehat{\mathbb{B}}, \widetilde{\mathbb{B}}^*, \boldsymbol{y}_\beta, \boldsymbol{f}, \tau, \kappa G, \xi G, \delta\xi G) \xleftarrow{R} \mathcal{G}^{\mathsf{ABP0}}(1^\lambda)$, where $\mathcal{G}^{\mathsf{ABP0}}_\beta(1^\lambda)$:

$$
\begin{aligned}
& \mathsf{param}_{\mathbb{G}} := (q, \mathbb{G}, \mathbb{G}_T, G, e) \xleftarrow{R} \mathcal{G}_{\mathsf{bmp}}(1^\lambda), && \mathsf{param}_{\mathbb{V}} := (q, \mathbb{V}, \mathbb{G}_T, \mathbb{A}, e) \xleftarrow{R} \mathcal{G}_{\mathsf{dpvs}}(1^\lambda, 3), \\
& X := \begin{pmatrix} \vec{\chi}_1, \\ \vec{\chi}_2 \\ \vec{\chi}_3 \end{pmatrix} := (\chi_{i,j})_{i,j} \xleftarrow{U} GL(3, \mathbb{F}_q), && (\vartheta_{i,j})_{i,j} := \begin{pmatrix} \vec{\vartheta}_1 \\ \vec{\vartheta}_2 \\ \vec{\vartheta}_3 \end{pmatrix} := (X^\top)^{-1}, \kappa, \xi \xleftarrow{U} \mathbb{F}_q^\times, \\
& \boldsymbol{b}_i := (\vec{\chi}_i)_{\mathbb{A}} = \textstyle\sum_{j=1}^3 \chi_{i,j}\boldsymbol{a}_j \ (i = 1, 3), && \widehat{\mathbb{B}} := (\boldsymbol{b}_1, \boldsymbol{b}_3), \\
& \boldsymbol{b}^*_i := (\vec{\vartheta}_i)_{\mathbb{A}} = \textstyle\sum_{j=1}^3 \vartheta_{i,j}\boldsymbol{a}_j \ (i = 1, 2, 3), && \widetilde{\mathbb{B}}^* := (\xi\boldsymbol{b}^*_1, \boldsymbol{b}^*_2, \kappa\boldsymbol{b}^*_3), \ \mathbb{B}^* := (\boldsymbol{b}^*_1, \boldsymbol{b}^*_2, \boldsymbol{b}^*_3), \\
& g_T := e(G, G), && \mathsf{param}_{\mathsf{ABP0}} := (\mathsf{param}_{\mathbb{V}}, g_T), \\
& \delta, \sigma, \omega, \theta \xleftarrow{U} \mathbb{F}_q, && \rho, \tau \xleftarrow{U} \mathbb{F}_q^\times, \\
& \boldsymbol{y}^*_0 := (\delta\xi, 0, \sigma\kappa)_{\mathbb{B}^*} = (\delta, 0, \sigma)_{\widetilde{\mathbb{B}}^*}, && \boldsymbol{y}^*_1 := (\delta\xi, \rho, \sigma\kappa)_{\mathbb{B}^*} = (\delta, \rho, \sigma)_{\widetilde{\mathbb{B}}^*}, \\
& \boldsymbol{f} := (\omega, \tau, \theta)_{\mathbb{B}} && \\
& \text{return} && (\mathsf{param}_{\mathsf{ABP0}}, \widehat{\mathbb{B}}, \widetilde{\mathbb{B}}^*, \boldsymbol{y}_\beta, \boldsymbol{f}, \tau, \kappa G, \xi G, \delta\xi G)
\end{aligned}
$$

Lemma 5.12 *For any adversary $\mathcal{B}$, there is a PPT algorithm $\mathcal{E}$, whose running time is essentially the same as that of $\mathcal{B}$ such that for any security parameter λ, $\mathsf{Adv}^{\mathsf{ABP0}}_{\mathcal{B}} \leq \mathsf{Adv}^{\mathsf{DLIN}}_{\mathcal{E}} + 5/q$*

Before we prove the lemma above, we first introduce a lemma that was proved by Okamoto and Takashima [OT10].

Lemma 5.13 ([OT10]) *Canonical maps are defined as follows, $\phi_{i,j}(\boldsymbol{x}) := (\overbrace{1,\ldots,1}^{i-1}, G_j, \overbrace{1,\ldots,1}^{N-i})$ where $\boldsymbol{x} = (G_1,\ldots,G_N)$. We can efficiently sample a random linear transformation $W := \sum_{i=1}^{N}\sum_{j=1}^{N} r_{i,j}\phi_{i,j}$ of $\mathbb{V}$ with random coefficients $\{r_{i,j}\}_{i,j\in[N]} \xleftarrow{\cup} GL(N,\mathbb{F}_q)$ by using $\{\phi_{i,j}\}$. Matrix $(r^*_{i,j}) := (\{r_{i,j}\}^{-1})^{\top}$ defines the adjoint action on $\mathbb{V}$ for pairing e, i.e., $e(W(\boldsymbol{x}),(W^{-1})^{\top}(\boldsymbol{y})) = e(\boldsymbol{x},\boldsymbol{y})$ for any $\boldsymbol{x},\boldsymbol{y}\in\mathbb{V}$, via $(W^{-1})^{\top} := \sum_{i=1}^{N}\sum_{j=1}^{N} r^*_{i,j}\phi_{i,j}$.*

Next, we prove Lemma 5.12 by using the lemma above.

Proof. Assume that there exists an adversary $\mathcal{B}$ that solves ABP0 for contradiction. We construct an algorithm $\mathcal{E}$ that solves the DLIN problem. Algorithm $\mathcal{E}$ is given a DLIN instance $(\mathsf{param}_{\mathbb{G}}, G, \xi G, \kappa G, \delta\xi G, \sigma\kappa G, Y_\beta)$. To use $\mathcal{B}$, $\mathcal{E}$ set up parameters. $\mathcal{E}$ generates $\mathsf{param}_{\mathbb{V}} := (q,\mathbb{V},\mathbb{G}_T,\mathbb{A},e) := \mathcal{G}_{\mathsf{dpvs}}(1^\lambda,3)$, computes $g_T := e(G,G)$ and $\mathsf{param}_{\mathsf{ABP0}} := (\mathsf{param}_{\mathbb{V}}, g_T)$, and sets two (3×3) matrices

$$\Pi^* := \begin{pmatrix} 1 & 0 & 1/\xi \\ 0 & 0 & 1 \\ 0 & 1 & 1/\kappa \end{pmatrix}, \Pi := \begin{pmatrix} 1 & 0 & 0 \\ -1/\xi & -1/\kappa & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

It is easily verified that $\Pi\cdot(\Pi^*)^{\top} = \boldsymbol{I}_3$. To simulate bases of ABP0, $\mathcal{E}$ computes following vectors.

$$\begin{aligned} \xi\boldsymbol{u}^*_1 &:= (\xi,0,1)_{\mathbb{A}}, & \boldsymbol{u}^*_2 &:= (0,0,1)_{\mathbb{A}}, & \kappa\boldsymbol{u}^*_3 &:= (0,\kappa,1)_{\mathbb{A}}, \\ \boldsymbol{u}_1 &:= (1,0,0)_{\mathbb{A}}, & \boldsymbol{u}_2 &:= (-1/\xi,-1/\kappa,1)_{\mathbb{A}}, & \boldsymbol{u}_3 &:= (0,1,0)_{\mathbb{A}}. \end{aligned}$$

Note that $\mathcal{E}$ cannot calculate $\boldsymbol{u}_2$ since it does not have $1/\kappa$ and $1/\xi$. Next, $\mathcal{E}$ chooses $\eta,\varphi,\tau \xleftarrow{\cup} \mathbb{F}_q$, such that $\tau\neq 0$ and sets $\boldsymbol{v} := (\varphi G, \eta G, \tau G) = (\varphi,\eta,\tau)_{\mathbb{A}}$ and $\boldsymbol{w}^*_\beta := (\delta\xi G, \sigma\kappa G, Y_\beta)$. By Lemma 5.13, $\mathcal{E}$ generates random linear transformation W on $\mathbb{V}$ and sets $\boldsymbol{b}_i := W(\boldsymbol{u}_i)$ for $i=1,2,3$,

$$\begin{aligned} \xi\boldsymbol{b}^*_1 &:= (W^{-1})^{\top}(\xi\boldsymbol{u}^*_1), & \boldsymbol{b}^*_2 &:= (W^{-1})^{\top}(\boldsymbol{u}^*_2), & \kappa\boldsymbol{b}^*_3 &:= (W^{-1})^{\top}(\kappa\boldsymbol{u}^*_3), \\ \widehat{\mathbb{B}} &:= (\boldsymbol{b}_1,\boldsymbol{b}_3), & \widetilde{\mathbb{B}}^* &:= (\xi\boldsymbol{b}^*_1,\boldsymbol{b}^*_2,\kappa\boldsymbol{b}^*_3), \\ \boldsymbol{f} &:= W(\boldsymbol{v}), & \boldsymbol{y}_\beta &:= (W^{-1})^{\top}(\boldsymbol{w}^*_\beta), \end{aligned}$$

and $\mathsf{param}_{\mathsf{ABP0}} := (\mathsf{param}_{\mathbb{V}}, g_T)$. To use adversary $\mathcal{B}$ of ABP0, $\mathcal{E}$ gives $(\mathsf{param}_{\mathsf{ABP0}}, \widehat{\mathbb{B}}, \widetilde{\mathbb{B}}^*, \boldsymbol{y}_\beta, \boldsymbol{f}, \tau, \kappa G, \xi G, \delta\xi G)$ to $\mathcal{B}$ as an input. If $\mathcal{B}$ outputs β', then $\mathcal{E}$ outputs $\hat{\beta} := \beta'$. Note that $\mathcal{E}$ can give τ to $\mathcal{B}$ since $\mathcal{E}$ chose τ.

Next, we analyze the success probability of $\mathcal{E}$. We must argue that the the simulated input is a valid instance of ABP0. If we set $\mathbb{U} := (\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3)$, $\theta := \eta + \tau/\kappa$ and $\omega := \varphi + \tau/\xi$, then it holds $\theta \neq 0$ since $\tau \neq 0$, and we can rewrite

$$\begin{aligned}\boldsymbol{v} &= (\varphi, \eta, \tau)_{\mathbb{A}} \\ &= (\omega - \tau/\xi, \theta - \tau/\kappa, \tau)_{\mathbb{A}} \\ &= \omega\boldsymbol{u}_1 + \tau\boldsymbol{u}_2 + \theta\boldsymbol{u}_3 \\ &= (\omega, \tau, \theta)_{\mathbb{U}}\end{aligned} \qquad \begin{aligned}\boldsymbol{f} &= W(\boldsymbol{v}) = W((\omega, \tau, \theta)_{\mathbb{U}}) \\ &= (\omega, \tau, \theta)_{\mathbb{B}}\end{aligned}$$

Thus, $\boldsymbol{f}$ is a valid input. We analyze two cases.

Case $\beta = 0$**:** In this case, it holds $Y_\beta = Y_0 = (\delta + \sigma)G$. Thus, it holds

$$\begin{aligned}\boldsymbol{w}_0^* &= (\delta\xi G, \sigma\kappa G, (\delta + \sigma)G) \\ &= (\delta\xi, \sigma\kappa, \delta + \sigma)_{\mathbb{A}} \\ &= \delta\xi\boldsymbol{u}_1^* + \sigma\kappa\boldsymbol{u}_3^* \\ &= (\delta\xi, 0, \sigma\kappa)_{\mathbb{U}^*},\end{aligned} \qquad \begin{aligned}\boldsymbol{y}_0 &= (W^{-1})^\top(\boldsymbol{w}_0^*) \\ &= (W^{-1})^\top((\delta\xi, 0, \sigma\kappa)_{\mathbb{U}^*}) \\ &= (\delta\xi, 0, \sigma\kappa)_{\mathbb{B}^*} \\ &= (\delta, 0, \sigma)_{\widetilde{\mathbb{B}}^*},\end{aligned}$$

where $\mathbb{U}^* := (\boldsymbol{u}_1^*, \boldsymbol{u}_2^*, \boldsymbol{u}_3^*)$. In this case, we can have $(\delta, 0, \sigma)_{\widetilde{\mathbb{B}}^*}$ and the distribution of $(\mathsf{param}_{\mathsf{ABP0}}, \widehat{\mathbb{B}}, \widetilde{\mathbb{B}}^*, \boldsymbol{y}_0, \boldsymbol{f}, \tau, \kappa G, \xi G, \delta\xi G)$ is exactly the same as an output of $\mathcal{G}_0^{\mathsf{ABP0}}(1^\lambda)$ when $\kappa \neq 0$ and $\xi \neq 0$. The event that $\kappa = 0$ or $\xi = 0$ happens with probability $2/q$.

Case $\beta = 1$**:** In this case, $Y_\beta = Y_1 = \psi G$ is uniformly distributed in $\mathbb{G}$. We set $\rho := \psi - \delta - \sigma$. In this case, it holds

$$\begin{aligned}\boldsymbol{w}_1^* &= (\delta\xi G, \sigma\kappa G, (\delta + \rho + \sigma)G) \\ &= (\delta\xi, \sigma\kappa, \delta + \rho + \sigma)_{\mathbb{A}} \\ &= \delta\xi\boldsymbol{u}_1^* + \rho\boldsymbol{u}_{2^*} + \sigma\kappa\boldsymbol{u}_3^* \\ &= (\delta\xi, \rho, \sigma\kappa)_{\mathbb{U}^*}\end{aligned} \qquad \begin{aligned}\boldsymbol{y}_1 &= (W^{-1})^\top(\boldsymbol{w}_1^*) \\ &= (W^{-1})^\top((\delta\xi, \rho, \sigma\kappa)_{\mathbb{U}^*}) \\ &= (\delta\xi, \rho, \sigma\kappa)_{\mathbb{B}^*} \\ &= (\delta, \rho, \sigma)_{\widetilde{\mathbb{B}}^*}\end{aligned}$$

where ρ is uniformly distributed. In this case, we can have $(\delta, \rho, \sigma)_{\widetilde{\mathbb{B}}^*}$ and the distribution of $(\mathsf{param}_{\mathsf{ABP0}}, \widehat{\mathbb{B}}, \widetilde{\mathbb{B}}^*, \boldsymbol{y}_1, \boldsymbol{f}, \tau, \kappa G, \xi G, \delta\xi G)$ is exactly the same as an output of $\mathcal{G}_1^{\mathsf{ABP0}}(1^\lambda)$ when $\kappa \neq 0$, $\xi \neq 0$, and $\rho \neq 0$. The event that $\kappa = 0$ or $\xi = 0$ or $\rho = 0$ happens with probability $3/q$.

Therefore, if $\mathcal{B}$ outputs β', then $\mathcal{E}$ can correctly output $\hat{\beta} = \beta$. □

Next, we introduce "Another Basic Problem 1", that is similar to the "Basic Problem 1" introduced by Okamoto and Takashima [OT10]

Definition 5.14 (Another Basic Problem 1) Another Basic Problem 1 (ABP1) is to guess $\beta \in \{0,1\}$, given $(\mathsf{param}_{\mathsf{ABP1}}, \widetilde{\mathbb{B}}, \widehat{\mathbb{B}}^*, \boldsymbol{e}_\beta, \boldsymbol{h}, \tau) \xleftarrow{\mathsf{R}} \mathcal{G}_\beta^{\mathsf{ABP1}}(1^\lambda)$, where

$$
\begin{aligned}
\mathcal{G}_\beta^{\mathsf{ABP1}}(1^\lambda): \quad & (\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{ob}}(1^\lambda, 10) \\
& \delta, \sigma, \omega, \theta \xleftarrow{\mathsf{U}} \mathbb{F}_q,\ \tau, \rho, \xi, \kappa \xleftarrow{\mathsf{U}} \mathbb{F}_q^\times \\
& \widetilde{\mathbb{B}} := (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_6, \xi\boldsymbol{b}_7, \kappa\boldsymbol{b}_8, \boldsymbol{b}_9),\ \widehat{\mathbb{B}}^* := (\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_9^*), \\
& \boldsymbol{e}_0 := (0, 0^2, 0^2, 0^2, \delta\xi, \sigma\kappa, 0)_{\mathbb{B}} = (0, 0^2, 0^2, 0^2, \delta, \sigma, 0)_{\widetilde{\mathbb{B}}} \\
& \boldsymbol{e}_1 := (\rho, 0^2, 0^2, 0^2, \delta\xi, \sigma\kappa, 0)_{\mathbb{B}} = (\rho, 0^2, 0^2, 0^2, \delta, \sigma, 0)_{\widetilde{\mathbb{B}}} \\
& \boldsymbol{h} := (\tau, 0^2, 0^2, 0^2, \omega, \theta, 0)_{\mathbb{B}^*}
\end{aligned}
$$

return $(\mathsf{param}_{\mathbb{V}}, \widetilde{\mathbb{B}}, \widehat{\mathbb{B}}^*, \boldsymbol{e}_\beta, \boldsymbol{h}, \tau)$.

Lemma 5.15 *For any adversary $\mathcal{B}$, there is a PPT algorithm $\mathcal{E}$, whose running time is essentially the same as that of $\mathcal{B}$ such that for any security parameter λ, $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{ABP1}} \leq \mathsf{Adv}_{\mathcal{E}}^{\mathsf{ABP0}}$*

Proof. Assume that there exists adversary $\mathcal{B}$ that solves ABP1 for contradiction. We construct algorithm $\mathcal{E}$. $\mathcal{E}$ is given the ABP0 instance $(\mathsf{param}_{\mathsf{ABP0}}, \widehat{\mathbb{B}}, \widetilde{\mathbb{B}}^*, \boldsymbol{y}_\beta, \boldsymbol{f}, \tau, \kappa G, \xi G, \delta\xi G)$. For vector $\boldsymbol{v} = (G_1, G_2, G_3) \in \mathbb{G}^3$, we use notation $(\boldsymbol{v}, 0^{n-3}) := (G_1, G_2, G_3, 0^{n-3})$. $\mathcal{E}$ generates random linear transformation W on $\mathbb{V}$, parses $\widehat{\mathbb{B}} = (\boldsymbol{b}_1, \boldsymbol{b}_3)$, $\widetilde{\mathbb{B}}^* = (\xi\boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \kappa\boldsymbol{b}_3^*)$, chooses $\psi \xleftarrow{\mathsf{U}} \mathbb{F}_q^\times$, and sets

$$
\begin{aligned}
\boldsymbol{d}_0 &:= W(\boldsymbol{b}_2^*, 0^7) & \boldsymbol{d}_0^* &:= \psi(W^{-1})^\top(\boldsymbol{b}_2, 0^7) \\
\boldsymbol{d}_i &:= W(0^{2+i}, G, 0^{7-i})\ (i = 1, \ldots 6) & \boldsymbol{d}_i^* &:= \psi(W^{-1})^\top(0^{2+i}, G, 0^{7-i})\ (i = 1, \ldots, 6) \\
\xi\boldsymbol{d}_7 &:= W(\xi\boldsymbol{b}_1^*, 0^7) & \boldsymbol{d}_7^* &:= \psi(W^{-1})^\top(\boldsymbol{b}_1, 0^7) \\
\kappa\boldsymbol{d}_8 &:= W(\kappa\boldsymbol{b}_3^*, 0^7) & \boldsymbol{d}_8^* &:= \psi(W^{-1})^\top(\boldsymbol{b}_3, 0^7) \\
\boldsymbol{d}_9 &:= W(0^9, G) & \boldsymbol{d}_9^* &:= \psi{W^{-1}}^\top(0^9, G), \\
\boldsymbol{g}_\beta &:= W(\boldsymbol{y}_\beta, 0^7) & \boldsymbol{p} &:= \psi(W^{-1})^\top(\boldsymbol{f}, 0^7) \\
& & &= (\tau, 0^6, \omega, \theta, 0)_{\mathbb{D}^*} \\
\mathbb{D} &:= (\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_6, \boldsymbol{d}_7, \boldsymbol{d}_8, \boldsymbol{d}_9) & \widehat{\mathbb{D}}^* &:= (\boldsymbol{d}_1^*, \ldots, \boldsymbol{d}_9^*) \\
\widetilde{\mathbb{D}} &:= (\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_6, \xi\boldsymbol{d}_7, \kappa\boldsymbol{d}_8, \boldsymbol{d}_9). & &
\end{aligned}
$$

$\mathcal{E}$ gives $(\mathsf{param}_{\mathsf{ABP0}}, \widetilde{\mathbb{D}}, \widehat{\mathbb{D}}^*, \boldsymbol{g}_\beta, \boldsymbol{p}, \tau)$ to $\mathcal{B}$ as an input of ABP0 and outputs whatever $\mathcal{B}$ outputs.

Case $\beta = 0$: In this case, $\boldsymbol{y}_\beta = (\delta\xi, 0, \sigma\kappa)_{\mathbb{B}^*}$, and it holds $\boldsymbol{g}_\beta = (0, 0^6, \delta\xi, \sigma\kappa, 0)_{\mathbb{D}} = (0, 0^6, \delta, \sigma, 0)_{\widetilde{\mathbb{D}}}$.

Case $\beta = 1$: In this case, $\boldsymbol{y}_\beta = (\delta\xi, \rho, \sigma\kappa)_{\mathbb{B}^*}$, and it holds $\boldsymbol{g}_\beta = (\rho, 0^6, \delta\xi, \sigma\kappa, 0)_{\mathbb{D}} = (\rho, 0^6, \delta, \sigma, 0)_{\widetilde{\mathbb{D}}}$.

Thus, the simulation by $\mathcal{E}$ is valid and $\mathcal{B}$ works correctly. That is, $\mathcal{E}$ can guess correctly. □

Definition 5.16 (Another Basic Problem 2) Another Basic Problem 2 (ABP2) is to guess $\beta \in \{0,1\}$ given $(\mathsf{param}_{\mathsf{ABP2}}, \widetilde{\mathbb{B}}, \widehat{\mathbb{B}}^*, \boldsymbol{e}_\beta, \boldsymbol{h}, \tau) \xleftarrow{R} \mathcal{G}_\beta^{\mathsf{ABP2}}(1^\lambda)$, where

$$
\begin{aligned}
\mathcal{G}_\beta^{\mathsf{ABP2}}(1^\lambda): \quad & (\mathsf{param}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{R} \mathcal{G}_{\mathsf{ob}}(1^\lambda, 10), \\
& \delta, \sigma, \omega, \theta \xleftarrow{U} \mathbb{F}_q,\ \tau, \rho, \xi, \kappa \xleftarrow{U} \mathbb{F}_q^\times, \\
& \widetilde{\mathbb{B}} := (\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \xi\boldsymbol{b}_3, \kappa\boldsymbol{b}_4, \boldsymbol{b}_5, \ldots, \boldsymbol{b}_9),\ \widehat{\mathbb{B}}^* := (\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_9^*), \\
& \boldsymbol{e}_0 := (0, 0^2, \delta\xi, \sigma\kappa, 0^2, 0)_{\mathbb{B}} = (0, 0^2, \delta, \sigma, 0^2, 0^2, 0)_{\widetilde{\mathbb{B}}}, \\
& \boldsymbol{e}_1 := (\rho, 0^2, \delta\xi, \sigma\kappa, 0^2, 0)_{\mathbb{B}} = (\rho, 0^2, \delta, \sigma, 0^2, 0^2, 0)_{\widetilde{\mathbb{B}}}, \\
& \boldsymbol{h} := (\tau, 0^2, \omega, \theta, 0^2, 0^2, 0)_{\mathbb{B}^*}
\end{aligned}
$$

return $(\mathsf{param}_{\mathbb{V}}, \widetilde{\mathbb{B}}, \widehat{\mathbb{B}}^*, \boldsymbol{e}_\beta, \boldsymbol{h}, \tau)$.

Lemma 5.17 *For any adversary $\mathcal{B}$, there is a PPT algorithm $\mathcal{E}$, whose running time is essentially the same as that of $\mathcal{B}$ such that for any security parameter λ, $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{ABP2}} \leq \mathsf{Adv}_{\mathcal{E}}^{\mathsf{ABP0}}$*

This can be proved as Lemma 5.15.

5.3.2 Proof of Theorem 5.4

To prove Theorem 5.4, we prove Theorem 5.18 and 5.19 explained below since if the DLIN problem is hard, then BP1, BP2, ABP0, ABP1, and ABP2 are also hard.

Theorem 5.18 (Non-removability) *Our scheme* CWM *satisfies non-removability if Another Basic Problem 1 is hard.*

Proof. If $\mathcal{A}$ outputs $(0, ek^*)$ where $\mathsf{Detect}(dk, ek^*) \to \mathsf{unmarked}$ and $\mathsf{IdealDtc}(ek^*) \to \mathsf{marked}$, then we can construct an algorithm that solves Another Basic Problem 1.

$\mathcal{B}$ is given ABP1 instance $(\mathsf{param}, \widetilde{\mathbb{B}}, \widehat{\mathbb{B}}^*, \boldsymbol{e}_{\mathsf{b}}, \boldsymbol{h}, \tau) \xleftarrow{R} \mathcal{G}_{\mathsf{b}}^{\mathsf{ABP1}}(1^\lambda)$, parses $\widehat{\mathbb{B}} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_6, \xi\boldsymbol{b}_7, \kappa\boldsymbol{b}_8, \boldsymbol{b}_9)$ $\widehat{\mathbb{B}}^* := (\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_9^*)$, and sets $pk := (\mathsf{param}_{\mathbb{V}}, \boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_9)$, $sk := (\perp, \boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \boldsymbol{b}_5^*, \boldsymbol{b}_6^*)$, $dk := (\xi\boldsymbol{b}_7, \kappa\boldsymbol{b}_8)$, $mk := (\boldsymbol{b}_7^*, \boldsymbol{b}_8^*)$. $\mathcal{B}$ can simulate the mark oracle since it has mk. Here, $\mathcal{B}$ does not have $\boldsymbol{b}_0$ and cannot directly compute conversion key $\boldsymbol{k}_{\mathsf{tag}}$. However, it can use $\boldsymbol{h}$. In order to simulate the challenge algorithm for tag, $\mathcal{B}$ chooses $\sigma', \mu_7, \mu_8 \xleftarrow{U} \mathbb{F}_q^\times, \vec{\eta} \xleftarrow{U} \mathbb{F}_q^2$ and computes

$$
\begin{aligned}
\boldsymbol{k}_{\mathsf{tag}} &:= (1/\tau)\boldsymbol{h} + (0, \sigma'(\mathsf{tag}, -1), 0^2, \vec{\eta}, \mu_7, \mu_8, 0)_{\mathbb{B}^*} \\
&= (1, \sigma'(\mathsf{tag}, -1), 0^2, \vec{\eta}, \mu_7 + \omega/\tau, \mu_8 + \theta/\tau, 0)_{\mathbb{B}^*}.
\end{aligned}
$$

$\mathcal{B}$ can compute it since it has τ, $\boldsymbol{h}$, and $\widehat{\mathbb{B}}^*$.

If $\mathcal{A}$ outputs $ek = (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{\mathsf{tag}^*})$ where unmarked $\boldsymbol{k}_{\mathsf{tag}^*} = (1, \sigma^*(\mathsf{tag}^*, -1), 0^2, \vec{\eta}^*, 0^2, 0)_{\mathbb{B}^*}$ whose function is the same as an answer by the simulated challenge oracle, then $\mathcal{B}$ chooses $\varphi, \varpi \xleftarrow{\mathsf{U}} \mathbb{F}_q$ and computes $C := \boldsymbol{e}_{\mathsf{b}} + \varpi \boldsymbol{b}_1 + \varpi \mathsf{tag}^* \boldsymbol{b}_2 + \varphi \boldsymbol{b}_9$ and $\Delta := \boldsymbol{e}(C, \boldsymbol{k}_{\mathsf{tag}^*})$. If the coefficient of $\boldsymbol{b}_9^*$ in $\boldsymbol{k}_{\mathsf{tag}^*}$ is not 0, then its functionality is not preserved, so the coefficient should be 0. Even if the coefficients of $\boldsymbol{b}_3^*$ and $\boldsymbol{b}_4^*$ in $\boldsymbol{k}_{\mathsf{tag}^*}$ are not 0, this does not affect the computation of Δ since C does not include elements by $\boldsymbol{b}_3$ and $\boldsymbol{b}_4$.

If $\mathsf{b} = 0$, then $C = (0, \varpi(1, \mathsf{tag}^*), 0^4, \delta\xi, \sigma\kappa, \varphi)_{\mathbb{B}}$. In this case, it holds $\Delta = 1$ and $\mathcal{B}$ outputs 0.

If $\mathsf{b} = 1$, then $C = (\rho, \varpi(1, \mathsf{tag}^*), 0^4, \delta\xi, \sigma\kappa, \varphi)_{\mathbb{B}}$. In this case, it holds $\Delta \neq 1$ and $\mathcal{B}$ outputs 1. □

Theorem 5.19 (Unforgeability) *Our scheme* CWM *satisfies unforgeability under the DLIN assumption.*

Proof. We can prove this theorem by considering similar games in the proofs of Theorem 4.6 in Section 4.3.

Lemma 5.20 *If $\mathcal{A}$ outputs semi-functional marked index in* Game-0*, then we can construct an algorithm that break Problem 1 with $n = 2$.*

Proof of lemma. $\mathcal{B}$ is given instance $(\mathsf{param}, \mathbb{B}, \widehat{\mathbb{B}}^*, \boldsymbol{e}_{\mathsf{b},1}, \boldsymbol{e}_2) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{b}}^{\mathsf{P1}}(1^\lambda, 2)$, parses $\widehat{\mathbb{B}}^* = (\boldsymbol{b}_0^*, \boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \boldsymbol{b}_5^*, \ldots, \boldsymbol{b}_9^*)$ and $\mathbb{B} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_9)$ and sets $\widehat{\mathbb{B}} := (\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_9)$, $pk := (\mathsf{param}_{\mathbb{V}}, \widehat{\mathbb{B}})$, $sk := (\boldsymbol{b}_0^*, \boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \boldsymbol{b}_5^*, \boldsymbol{b}_6^*)$, $dk := (\boldsymbol{b}_7, \boldsymbol{b}_8)$, and $mk := (\boldsymbol{b}_7^*, \boldsymbol{b}_8^*)$. In order to simulate the challenge algorithm for tag, $\mathcal{B}$ computes $\boldsymbol{k}_{\mathsf{tag}} := (1, \sigma(\mathsf{tag}, -1), 0^2, \vec{\eta}, \mu_1, \mu_2, 0)_{\mathbb{B}^*}$ by using sk and mk. If $\mathcal{A}$ outputs $ek = (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{\mathsf{tag}^*})$, where $\boldsymbol{k}_{\mathsf{tag}^*} = (1, \sigma^*(\mathsf{tag}^*, -1), \vec{\tau}^*, \vec{\eta}^*, \vec{\mu}^*, 0)_{\mathbb{B}^*}$, then $\mathcal{B}$ computes $C := \boldsymbol{e}_{\mathsf{b},1} + \mathsf{tag}^* \boldsymbol{e}_2 + \zeta \boldsymbol{b}_0 + \varphi \boldsymbol{b}_9$ and $C_0 := g_T^{\zeta}$. Note that if the coefficient of $\boldsymbol{b}_9^*$ in $\boldsymbol{k}_{\mathsf{tag}^*}$ is 0, then the functionality is not preserved. We set $\varphi' := \varphi + \gamma$.

If $\mathsf{b} = 0$, then $C = (\zeta, \omega(1, \mathsf{tag}^*), 0^6, \varphi')_{\mathbb{B}}$.

If $\mathsf{b} = 1$, then $C = (\zeta, \omega(1, \mathsf{tag}^*), \vec{z}, 0^4, \varphi')_{\mathbb{B}}$.

Thus, $\mathcal{B}$ outputs the correct guess by computing $\Delta := \boldsymbol{e}(C, \boldsymbol{k}_{\mathsf{tag}^*})$. If $\Delta / C_0 = 1$, then $\mathcal{B}$ outputs 0. Otherwise, it outputs 1. ■

Lemma 5.21 *If there exists $\mathcal{A}$ which distinguishes* Game-$(i-1)$ *from* Game-(i)*, then we can construct an algorithm that break Problem 2 with $n = 2$.*

Proof of lemma. $\mathcal{B}$ is given instance $(\mathsf{param}, \widehat{\mathbb{B}}, \mathbb{B}^*, \boldsymbol{h}_{\mathsf{b},1}^*, \boldsymbol{h}_{\mathsf{b},2}^*, \boldsymbol{e}_1, \boldsymbol{e}_2) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{b}}^{\mathsf{P2}}(1^\lambda, 2)$, parses $\widehat{\mathbb{B}} = (\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_5, \ldots, \boldsymbol{b}_9)$ and $\mathbb{B}^* = (\boldsymbol{b}_0^*, \ldots, \boldsymbol{b}_9^*)$, and sets $\widehat{\mathbb{B}}' := (\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_9)$, $pk := (\mathsf{param}_{\mathbb{V}}, \widehat{\mathbb{B}}')$, $sk := (\boldsymbol{b}_0^*, \boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \boldsymbol{b}_5^*, \boldsymbol{b}_6^*)$, $dk := (\boldsymbol{b}_7, \boldsymbol{b}_8)$, and $mk := (\boldsymbol{b}_7^*, \boldsymbol{b}_8^*)$. In order to simulate the challenge algorithm for tag, $\mathcal{B}$ computes $\boldsymbol{k}_{\mathsf{tag}} := (1, \sigma(\mathsf{tag}, -1), \vec{\tau}, \vec{\eta}, \vec{\mu}, 0)_{\mathbb{B}^*}$. $\mathcal{B}$ can compute it as both normal and semi-functional since it has $\mathbb{B}^*$ and can set $\vec{\tau} := \vec{0}$.

When $\mathcal{B}$ answer i-th query, it computes $\boldsymbol{k}_{\mathsf{tag}_i} := \mathsf{tag}_i \boldsymbol{h}_{\mathsf{b},1}^* - \boldsymbol{h}_{\mathsf{b},2}^* + \boldsymbol{b}_0^* + \tau_1 \boldsymbol{b}_5^* + \tau_2 \boldsymbol{b}_6^* + \mu_1 \boldsymbol{b}_7^* + \mu_2 \boldsymbol{b}_8^*$.

If $\mathsf{b}=0$, then $\boldsymbol{k}_{\mathsf{tag}_i}=(1,\delta(\mathsf{tag}_i,-1),0^2,\tau_1+\delta_0\mathsf{tag}_i,\tau_2-\delta_0),\mu_1,\mu_2,0)_{\mathbb{B}}$.

If $\mathsf{b}=1$, then $\boldsymbol{k}_{\mathsf{tag}_i}=(1,\delta(\mathsf{tag}_i,-1),\tau(\mathsf{tag}_i,-1),\tau_1+\delta_0\mathsf{tag}_i,\tau_2-\delta_0,\mu_1,\mu_2,0)_{\mathbb{B}}$. These correspond to Game-$(i-1)$ and Game-(i), respectively.

If $\mathcal{A}$ outputs $ek=(\boldsymbol{U},\boldsymbol{V},\boldsymbol{k}_{\mathsf{tag}^*})$ where $\boldsymbol{k}_{\mathsf{tag}^*}=(1,\sigma^*(\mathsf{tag}^*,-1),\vec{\tau}^*,\vec{\mu}^*,\vec{\eta}^*,0)_{\mathbb{B}^*}$, then $\mathcal{B}$ chooses $\zeta,\varphi\xleftarrow{\mathsf{U}}\mathbb{F}_q$ and computes $C_0:=g_T^{\zeta}$ and

$$\begin{aligned}C&:=\boldsymbol{e}_1+\mathsf{tag}^*\boldsymbol{e}_2+\zeta\boldsymbol{b}_0+\varphi\boldsymbol{b}_9\\&=(\zeta,\omega(1,\mathsf{tag}^*),\sigma(1,\mathsf{tag}^*),0^4,\varphi)_{\mathbb{B}}.\end{aligned}$$

If $\vec{\tau}^*=\vec{0}$ (normal), then $\Delta/C_0=1$ holds and $\mathcal{B}$ outputs 0.

If $\vec{\tau}^*\neq\vec{0}$ (semi-functional), then $\Delta/C_0\neq 1$ holds and $\mathcal{B}$ outputs 1 since $\mathsf{tag}^*\neq\mathsf{tag}_i$. ■

Lemma 5.22 *If $\mathcal{A}$ outputs normal marked index in* Game-q_{C}*, then we can construct an algorithm that break Another Basic Problem 2.*

Proof of lemma. $\mathcal{B}$ is given instance $(\mathsf{param},\widetilde{\mathbb{B}},\widehat{\mathbb{B}}^*,\boldsymbol{e}_{\mathsf{b}},\boldsymbol{h},\tau)\xleftarrow{\mathsf{R}}\mathcal{G}_{\mathsf{b}}^{\mathsf{ABP2}}(1^\lambda)$, parses $\widetilde{\mathbb{B}}=(\boldsymbol{b}_0,\boldsymbol{b}_1,\boldsymbol{b}_2,\xi\boldsymbol{b}_3,\kappa\boldsymbol{b}_4,\boldsymbol{b}_5,\ldots,\boldsymbol{b}_9)$ and $\widehat{\mathbb{B}}^*:=(\boldsymbol{b}_1^*,\ldots,\boldsymbol{b}_9^*)$, and sets $pk:=(\mathsf{param}_{\mathbb{V}},\boldsymbol{b}_0,\boldsymbol{b}_1,\boldsymbol{b}_2,\boldsymbol{b}_9)$, $sk:=(\perp,\boldsymbol{b}_1^*,\boldsymbol{b}_2^*,\boldsymbol{b}_5^*,\boldsymbol{b}_6^*)$, $dk:=(\boldsymbol{b}_7,\boldsymbol{b}_8)$, $mk:=(\boldsymbol{b}_7^*,\boldsymbol{b}_8^*)$. In order to simulate the challenge algorithm for tag, $\mathcal{B}$ chooses $\tau_1,\tau_2\xleftarrow{\mathsf{U}}\mathbb{F}_q^\times$ and $\vec{\eta},\vec{\mu}\xleftarrow{\mathsf{U}}\mathbb{F}_q^2$ and computes

$$\begin{aligned}\boldsymbol{k}_{\mathsf{tag}}&:=(1/\tau)\boldsymbol{h}+(0,\sigma(\mathsf{tag},-1),\tau_1,\tau_2,\vec{\eta},\vec{\mu},0)_{\mathbb{B}^*}\\&=(1,\sigma(\mathsf{tag},-1),\ \tau_1+\omega/\tau,\tau_2+\theta/\tau,\vec{\eta},\vec{\mu},0)_{\mathbb{B}^*}.\end{aligned}$$

$\mathcal{B}$ can compute it since it has $\widehat{\mathbb{B}}^*$, $\boldsymbol{h}$, and τ.

If $\mathcal{A}$ outputs $ek=(\boldsymbol{U},\boldsymbol{V},\boldsymbol{k}_{\mathsf{tag}^*})$ where $\boldsymbol{k}_{\mathsf{tag}^*}=(1,\sigma^*(\mathsf{tag}^*,-1),0^2,\vec{\eta}^*,\vec{\mu}^*,0)_{\mathbb{B}^*}$, then $\mathcal{B}$ chooses $\varphi\xleftarrow{\mathsf{U}}\mathbb{F}_q$ and computes $C:=\boldsymbol{e}_{\mathsf{b}}+\omega\boldsymbol{b}_1+\omega\mathsf{tag}^*\boldsymbol{b}_2+\varphi\boldsymbol{b}_9$ and $\Delta:=\boldsymbol{e}(C,\boldsymbol{k}_{\mathsf{tag}^*})$.

If $\mathsf{b}=0$, then $C=(0,\omega(1,\mathsf{tag}^*),\delta\xi,\sigma\kappa,0^4,\varphi)_{\mathbb{B}}$.

If $\mathsf{b}=1$, then $C=(\rho,\omega(1,\mathsf{tag}^*),\delta\xi,\sigma\kappa,0^4,\varphi)_{\mathbb{B}}$.

If $\mathsf{b}=0$, then $\Delta=1$ and $\mathcal{B}$ outputs 0. If $\mathsf{b}=1$, then $\Delta\neq 1$ and $\mathcal{B}$ outputs 1. Thus, $\mathcal{B}$ can guess correctly. ■

The theorem follows the lemmas. □

6 Concluding Remarks

We introduced the notion of cryptographic watermarking schemes, defined its security notion, and proposed two concrete constructions by using DPVS. Both of them are secure under the DLIN assumption in the standard model. This gives us the first positive result about provably secure watermarking schemes. We list a few remarks.

Constructions Based on the Symmetric External Diffie-Hellman Assumption. Chen, Lim, Ling, Wang, and Wee proposed an IBE scheme by using the subspace assumption based on the symmetric external Diffie-Hellman (SXDH) assumption, where the decisional Diffie-Hellman assumption holds in both groups of an asymmetric pairing group [CLL$^+$12]. Their IBE scheme is similar to Lewko's scheme and we can apply our technique to their scheme. Thus, we can construct a more efficient watermarking scheme based on the SXDH assumption.

Constructions Based on Composite-Order Pairing Groups. We use the canceling property of DPVS and sub-group decision type assumption to prove the security. Composite-order pairing groups also have such properties [LW10, LOS$^+$10]. Therefore, we can construct watermarking schemes based on composite-order pairing groups. However, we do not give concrete constructions in this paper since, generally speaking, schemes based on composite-order groups are less efficient than schemes based on prime-order groups due to large composites. One may think that we do not have remove algorithms if we use composite-order groups since we do not have trapdoor matrices of DPVS and the decomposition algorithm by Okamoto and Takashima. However, we note that if we use prime factors of composites as trapdoors, then we can also achieve remove algorithms in the composite-order group setting.

Open Issues. Our watermarking schemes are called the detection-type watermarking scheme, in which we can verify just one-bit information, embedded or not. We can consider a stronger variant called the extraction-type watermarking scheme, in which we can embed a message as a mark and extract it. In fact, our schemes can be modified into extraction-type schemes by adding extra $(2\mu - 2)$-dimension to our schemes for μ-bit messages since we can embed a one-bit message for each 2-dimension. However, this is quite inefficient. Thus, it is an open problem to construct more efficient extraction-type watermarking schemes.

Acknowledgements.

The author is grateful to Keita Xagawa for illuminating discussions and insightful comments about constructions of watermarking schemes. The author would like to thank Kazumaro Aoki and Atsushi Fujioka for helpful comments about applications of watermarking and the presentation of the introduction. The author would like to thank Shota Yamada for pointing out that we can use prime factors of a composite to construct a remove algorithm in the composite-order group setting. The author also would like to thank Masayuki Abe, Angelo De Caro, Akinori Kawachi, Katsuyuki Takashima, Maki Yoshida and anonymous reviewers of Asiacrypt 2012 and Eurocrypt 2013 for helpful comments.

References

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

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

[BW06] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 290–307. Springer, 2006.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 1994.

[CLL+12] Jie Chen, Hoon Wei Lim, San Ling, Huaxiong Wang, and Hoeteck Wee. Shorter ibe and signatures via asymmetric pairings. In *Pairing*, volume 7708 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 2012.

[CT02] Christian S. Collberg and Clark D. Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Trans. Software Eng.*, 28(8):735–746, 2002.

[FGK+10] David Mandell Freeman, Oded Goldreich, Eike Kiltz, Alon Rosen, and Gil Segev. More constructions of lossy and correlation-secure trapdoor functions. In *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 279–295. Springer, 2010.

[GV08] Steven D. Galbraith and Eric R. Verheul. An analysis of the vector decomposition problem. In *Public Key Cryptography*, volume 4939 of *LNCS*, pages 308–327. Springer, 2008.

[HMW07] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 362–382. Springer, 2007.

[HO12] Brett Hemenway and Rafail Ostrovsky. Extended-DDH and lossy trapdoor functions. In *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 627–643. Springer, 2012.

[HWZ07] Qiong Huang, Duncan S. Wong, and Yiming Zhao. Generic transformation to strongly unforgeable signatures. In *ACNS*, volume 4521 of *LNCS*, pages 1–17. Springer, 2007.

[KY02] Aggelos Kiayias and Moti Yung. Traitor tracing with constant transmission rate. In *EUROCRYPT*, volume 2332 of *LNCS*, pages 450–465. Springer, 2002.

[Lew12] Allison B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In *EUROCRYPT*, volume 7237 of *LNCS*, pages 318–335. Springer, 2012.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, volume 6110 of *LNCS*, pages 62–91. Springer, 2010.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, 1988.

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure hibe with short ciphertexts. In *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 455–479. Springer, 2010.

[NSS99] David Naccache, Adi Shamir, and Julien P. Stern. How to copyright a function? In *Public Key Cryptography*, volume 1560 of *LNCS*, pages 188–196. Springer, 1999.

[OT08] Tatsuaki Okamoto and Katsuyuki Takashima. Homomorphic encryption and signatures from vector decomposition. In *Pairing*, volume 5209 of *LNCS*, pages 57–74. Springer, 2008.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, volume 5912 of *LNCS*, pages 214–231. Springer, 2009.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, volume 6223 of *LNCS*, pages 191–208. Springer, 2010.

[OT11] Tatsuaki Okamoto and Katsuyuki Takashima. Efficient attribute-based signatures for non-monotone predicates in the standard model. In *Public Key Cryptography*, volume 6571 of *LNCS*, pages 35–52. Springer, 2011.

[OT12] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 591–608. Springer, 2012.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC'08*, pages 187–196. ACM, 2008.

[PW11] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. *SIAM J. Comput.*, 40(6):1803–1844, 2011.

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

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO'09*, volume 5677 of *LNCS*, pages 619–636. Springer, 2009. full version available from http://eprint.iacr.org/2009/385.

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

[YMF10] Maki Yoshida, Shigeo Mitsunari, and Toru Fujiwara. The vector decomposition problem. *IEICE Transactions*, 93-A(1):188–193, 2010.

A Lewko IBE and Okamoto-Takashima IPE Schemes

We present Lewko IBE scheme and Okamoto-Takashima IPE scheme for reference.

A.1 Identity-Based Encryption

We review Lewko IBE scheme, $\mathsf{IBE}_{\mathsf{L}}$ [Lew12] in this section.

$\mathsf{Setup}(1^\lambda)$**:** It generates $\Lambda := (q, \mathbb{G}, \mathbb{G}_T, e, g) \xleftarrow{R} \mathcal{G}_{\mathsf{bmp}}(1^\lambda)$ and $(\mathfrak{D}, \mathfrak{D}^*) \xleftarrow{U} \mathsf{Dual}(\mathbb{Z}_p^6)$, chooses $\alpha, \theta, \sigma \xleftarrow{U} \mathbb{Z}_p$, and sets $pk := (\Lambda, e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*}, g^{\vec{d}_1}, \ldots, g^{\vec{d}_4})$, $msk := (g^{\theta\vec{d}_1^*}, g^{\alpha\vec{d}_1^*}, g^{\theta\vec{d}_2^*}, g^{\sigma\vec{d}_3^*}, g^{\sigma\vec{d}_4^*})$. It outputs (pk, msk).

$\mathsf{Gen}(msk, ID)$**:** It chooses $r_1, r_2 \xleftarrow{U} \mathbb{Z}_p$ and generates $sk_{ID} := g^{(\alpha+r_1 ID)\theta\vec{d}_1^* - r_1\theta\vec{d}_2^* + r_2 ID\sigma\vec{d}_3^* - r_2\sigma\vec{d}_4^*}$.

$\mathsf{Enc}(pk, ID, M)$**:** It chooses $s_1, s_2 \xleftarrow{U} \mathbb{Z}_p$ and generates $C_0 := M \cdot (e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*})^{s_1}$ and $C := g^{s_1\vec{d}_1 + s_1 ID\vec{d}_2 + s_2\vec{d}_3 + s_2 ID\vec{d}_4}$. It outputs ciphertext $\mathsf{ct} := (C_0, C)$.

$\mathsf{Dec}(sk_{ID}, \mathsf{ct})$**:** It outputs $M := C_0 / \boldsymbol{e}(sk_{ID}, C)$.

Theorem A.1 ([Lew12]) *If the DLIN assumption holds, then* $\mathsf{IBE}_{\mathsf{L}}$ *is adaptively secure against chosen plaintext attacks.*

A.2 Attribute-Hiding Inner-Product Encryption [OT12]

We review an adaptively secure attribute-hiding inner-product encryption scheme by Okamoto and Takashima, $\mathsf{IPE}_{\mathsf{OT}}$ [OT12].

$\mathsf{Setup}(1^\lambda, n)$ **:** It generates $(\mathsf{param}_{\mathbb{G}}, \mathbb{B}, \mathbb{B}^*) \xleftarrow{R} \mathcal{G}_{\mathsf{ob}}^{\mathsf{asym}}(1^\lambda, 4n+2)$, sets $\bar{\mathbb{B}} := (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_n, \boldsymbol{b}_{4n+1})$ and $\bar{\mathbb{B}}^* := (\boldsymbol{b}_0^*, \ldots \boldsymbol{b}_n^*, \boldsymbol{b}_{3n+1}^*, \ldots, \boldsymbol{b}_{4n}^*)$, and returns $pk := (\mathsf{param}_{\mathbb{G}}, \bar{\mathbb{B}})$, $msk := \bar{\mathbb{B}}^*$.

$\mathsf{Gen}(pk, msk, \vec{v} \in \mathbb{F}_q^n \setminus \{\vec{0}\})$**:** It selects $\sigma \xleftarrow{\cup} \mathbb{F}_q$ and $\eta \xleftarrow{\cup} \mathbb{F}_q^n$, computes $\boldsymbol{k}^* := (\overbrace{1}^{1}, \overbrace{\sigma\vec{v}}^{n}, \overbrace{0^{2n}}^{2n}, \overbrace{\vec{\eta}}^{n}, \overbrace{0}^{1})_{\mathbb{B}^*}$, and returns $sk_{\vec{v}} := \boldsymbol{k}^*$.

$\mathsf{Enc}(pk, M, \vec{x} \in \mathbb{F}_q^n \setminus \{\vec{0}\})$**:** It selects $\omega, \varphi, \zeta \xleftarrow{\cup} \mathbb{F}_q$, computes $\boldsymbol{c}_1 := (\overbrace{\zeta}^{1}, \overbrace{\omega\vec{x}}^{n}, \overbrace{0^{2n}}^{2n}, \overbrace{0^{n}}^{n}, \overbrace{\varphi}^{1})_{\mathbb{B}}$ and $c_2 := M \cdot g_T^{\zeta}$, and returns $\boldsymbol{c}_{\vec{x}} := (\boldsymbol{c}_1, c_2)$.

$\mathsf{Dec}(pk, sk_{\vec{v}} := \boldsymbol{k}^*, \boldsymbol{c}_{\vec{x}} := (\boldsymbol{c}_1, c_2))$**:** It computes $M' := c_2 / e(\boldsymbol{c}_1, \boldsymbol{k}^*)$ and returns M'.

If $\vec{x} \cdot \vec{v} = 0$, then $e(\boldsymbol{c}_1, \boldsymbol{k}^*) = g_T^{\zeta + \omega\sigma\vec{x}\cdot\vec{v}} = g_T^{\zeta}$.

Theorem A.2 ([OT12]) *If the DLIN assumption holds, then* $\mathsf{IPE}_{\mathsf{OT}}$ *is adaptively attribute-hiding against chosen plaintext attacks.*