How to Watermark Cryptographic Functions

Ryo Nishimaki

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

Abstract. We propose a scheme for watermarking cryptographic functions. Informally speaking, a digital watermarking scheme for cryptographic functions embeds information, called a *mark*, into functions such as (trapdoor) one-way functions and decryption functions of public-key encryption. It is required that a mark-embedded function is functionally equivalent to the original function and it is difficult for adversaries to remove the embedded mark without damaging the function. In spite of its importance and usefulness, there have only been a few theoretical studies on watermarking for functions (or program), and 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. 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 the 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

1 Introduction

1.1 Background

Digital watermarking is a method of embedding information, called a "mark", in digital objects such as images, movies, and audio files. Marked objects look similar to the original objects and it is difficult to remove embedded marks without destroying the object. One of the applications of watermarking is protecting copyright, i.e., we can prevent unauthorized copying of digital content by detecting watermarks. Another application is tracing and identifying owners of digital content, that is, if we find 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, and only a few studies have focused on watermarking for non-perceptual objects (e.g., software, program). Software is digital content, so it can be easily copied. Software piracy is a serious problem today. Watermarking for programs is one of tools to solve the problem and has very useful, attractive, and practical applications, but 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" [14]. This is related to watermarking schemes for program (functions), but their security definition is a bit weak and not sufficient for program watermarking. 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 [1]. They proposed a formalization of software watermarking and its security definition, but the definition is simulation based and too strong. They gave an impossibility result for general-purpose program watermarking by using impossibility results of general-purpose program obfuscation [1]. "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 [23]. 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 [24], but their proof is heuristic and they showed no reduction.

Hopper, Molnar, and Wagner proposed a rigorous complexity-theoretic (game-based) definition of security for watermarking schemes, but they focused on watermarking for only *perceptual* objects [8]. 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 that is provably secure in a complexity-theoretic definition of security. We consider functions as a kind of program. 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 [14].

Traceable cryptographic primitives. One application of watermarking for cryptographic functions (we sometimes 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 [14]:

- If we consider software or program that generates ciphertexts of the Feistel cipher based on a one-way function [13], signatures of Rompel's signature scheme [21], or decrypted values of ciphertexts under PKE schemes based on a trapdoor one-way function, and a malicious user illegally makes copies of such software and distributes them, then a company that sold the software can trace them and identify the guilty users.
- If a company sells MAC-functions based on watermarked one-way functions to users and records user IDs and marked functions in a database and the users use them to log-in a member web site, then they do not need to disclose 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 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 [10]. When we want to 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. Traitor tracing enables us to identify such malicious subscribers called traitor [3]. 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 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 [4], 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

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.

Our security notion is based on the notion of strong watermarking introduced by Hopper et al. [8]. 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 assume 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) [20]. 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) [20]. 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 [16, 17, 19]. We use the IBE scheme of Okamoto and Takashima [19] (which is a special case of their inner-product predicate encryption (IPE) scheme) and that of Lewko [11] 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 [22]. The methodology is a key technique to achieve a watermarking scheme.

We apply the dual system encryption technique to not only security proofs but also *constructions of cryptographic primitives*. In the dual system encryption, we can use *semi-functional ciphertexts* and *semi-functional keys*. Semi-functional ciphertexts can be decrypted using normal keys and normal ciphertext can be decrypted using semi-functional keys, but 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* [5, 7, 20], 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.

Our watermarking scheme is based on DPVS. We can set a hidden linear subspace by concealing the basis of a subspace from public parameters due to a nice property of DPVS. A pair of dual orthonormal bases, $\mathbb{B}$ and $\mathbb{B}^*$, are generated using a random linear transformation matrix. We use a hidden linear subspace spanned by a subset of $\mathbb{B}/\mathbb{B}^*$ for semi-functional ciphertexts/keys (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 [15, 17]. 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 [17, 19] and these problems are deeply related to the VD problem [24] 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 [11], which is implied by the DLIN assumption and highly related to the decisional subspace assumption introduced by Okamoto and Takashima [15] and the VD problem. All assumptions introduced by Okamoto-Takashima [17, 19] and Lewko [11] 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 [6, 24]. 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 [15, 17], that is, we can remove a mark if we have the matrix. Such an algorithm was proposed by Okamoto and Takashima [15].

Our construction is a new application of DPVS. DPVS has been used to construct fully secure functional encryption, IPE, IBE and attribute-based signatures [11,12,16–19], 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.

Remark. In this extended abstract, we do not have enough space to give complete proofs and all definitions, so we omitted some of them.

2 Preliminaries

Notations. For any $n \in \mathbb{N} \setminus \{0\}$, let $[n]$ be the set $\{1, \dots, n\}$. When D is a random variable or distribution, $y \overset{\mathsf{R}}{\leftarrow} D$ means that y is randomly selected from D according to its distribution. If S is a set, then $x \overset{\mathsf{U}}{\leftarrow} S$ means that x is uniformly selected from S. We denote y is set, defined or substituted by z by $y := z$. When b is a fixed value, $A(x) \to b$ (e.g., $A(x) \to 1$) denotes the event that probabilistic polynomial-time (PPT) 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 $f(\lambda) = \lambda^{-\omega(1)}$ (We write $f < \mathsf{negl}(\lambda)$). We denote the finite field of order q by $\mathbb{F}_q$, and $\mathbb{F}_q \setminus \{0\}$ by $\mathbb{F}_q^{\times}$. A bold face small letter denotes an element of vector space $\mathbb{V}$, e.g., $\boldsymbol{x} \in \mathbb{V}$. Set $GL(n, \mathbb{F}_q)$ denotes the general linear group of degree n over $\mathbb{F}_q$. Let $\mathcal{G}_{\mathsf{bm}}$ be a parameter generation algorithm that takes as input security parameter λ and outputs $(q, \mathbb{G}, \mathbb{G}_T, e, g)$. If we use g/G to denote a generator in $\mathbb{G}$, then we use multiplicative/additive notation, respectively.

2.1 Function Family of Lossy Trapdoor Functions

Definition 1 (Lossy Trapdoor Functions [20]). *A lossy trapdoor function* LTF *with domain* D *consists of four efficient algorithms satisfying three properties*

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

Evaluation: $\mathsf{LTF.Eval}_{ek}(X)$ *(*$X \in \mathsf{D}$*) 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* $(ek', \perp)$.

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

Indistinguishability: *Let* λ *be a security parameter. For all PPT* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND}}_{\mathsf{LTF},\mathcal{A}}(\lambda)$ $:= \left|\Pr[\mathcal{A}(1^\lambda, [\mathsf{LTF.IGen}(1^\lambda)]_1)] - \Pr[\mathcal{A}(1^\lambda, [\mathsf{LTF.LGen}(1^\lambda)]_1)]\right| < \mathsf{negl}(\lambda)$.

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

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

2.2 Dual Pairing Vector Space [12,16,17]

Definition 2. *"Dual pairing vector space (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* $\phi_{i,j}(\boldsymbol{x}) := (0, \ldots, 0, G_j, 0, \ldots, 0)$ *(only the* i*-th coordinate is* G*) 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* $\mathsf{pp}'_{\mathbb{V}} := (q, \mathbb{V}, \mathbb{G}_T, \mathbb{A}, e)$ *with security parameter* λ *and* N*-dimensional* $\mathbb{V}$*. It can be constructed using* $\mathcal{G}_{\mathsf{bm}}$.

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 $\mathbb{V}$. We describe random dual orthonormal bases generator $\mathcal{G}_{\mathsf{ob}}(1^\lambda, N)$: Generate $\mathsf{pp}'_{\mathbb{V}} := (q, \mathbb{V}, \mathbb{G}_T, \mathbb{A}, e) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{dpvs}}(1^\lambda, N)$, $\boldsymbol{X} := (\chi_{i,j}) \xleftarrow{\mathsf{U}} GL(N, \mathbb{F}_q)$, $\psi \xleftarrow{\mathsf{U}} \mathbb{F}_q^\times$, $(\vartheta_{i,j}) := \psi(\boldsymbol{X}^\top)^{-1}$, $g_T := e(G, G)^\psi$, $\mathsf{pp}_{\mathbb{V}} :=$

$(\mathsf{pp}'_{\mathbb{V}}, g_T)$, $\boldsymbol{b}_i := \sum_{j=1}^{N} \chi_{i,j}\boldsymbol{a}_j, \mathbb{B} := (\boldsymbol{b}_1, \dots, \boldsymbol{b}_N)$, $\boldsymbol{b}_i^* := \sum_{j=1}^{N} \vartheta_{i,j}\boldsymbol{a}_j, \mathbb{B}^* := (\boldsymbol{b}_1^*, \dots, \boldsymbol{b}_N^*)$, return $(\mathsf{pp}_{\mathbb{V}}, \mathbb{B}, \mathbb{B}^*)$. It holds that $\boldsymbol{e}(\boldsymbol{b}_i, \boldsymbol{b}_j^*) = (g_T)^{\delta_{i,j}}$.

Vector decomposition problem. The VD problem was originally introduced by Yoshida, Mitsunari, and Fujiwara [24]. We present the definition of a higher dimensional version by Okamoto and Takashima [15] to fit the VD problem into DPVS. Note that a specific class of the CVDP instances that are specified over canonical basis $\mathbb{A}$ are tractable.

Definition 3 (CVDP: (ℓ_1, ℓ_2)-Computational Vector Decomposition Problem [15]). *For $\ell_1 > \ell_2$ and all $\lambda \in \mathbb{N}$, we define the advantage*

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

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

Trapdoor. If we have a trapdoor, 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 [15]. The input is $(\boldsymbol{v}, (\boldsymbol{b}_1, \dots, \boldsymbol{b}_{\ell_2}), \boldsymbol{X}, \mathbb{B})$ such that $\boldsymbol{v} := \sum_{i=1}^{\ell_1} y_i\boldsymbol{b}_i$ is a target vector for decomposition, $(\boldsymbol{b}_1, \dots, \boldsymbol{b}_{\ell_2})$ is a subspace to be decomposed into, $\boldsymbol{X}$ is a trapdoor, and $\mathbb{B} := (\boldsymbol{b}_1, \dots, \boldsymbol{b}_{\ell_1})$ is a basis generated by using $\boldsymbol{X}$. Algorithm $\mathsf{Decomp}(\boldsymbol{v}, (\boldsymbol{b}_1, \dots, \boldsymbol{b}_{\ell_2}), \boldsymbol{X}, \mathbb{B})$: 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, $(\chi_{i,j}) = \boldsymbol{X}$ and $(\tau_{i,j}) := (\boldsymbol{X})^{-1}$.

Lemma 1 (Okamoto-Takashima [15]). *Algorithm* Decomp *efficiently 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 DPVS by Lewko [11]. We introduce a multiplicative notation by Lewko [11]. For $\vec{v}, \vec{w} \in \mathbb{F}_q^n$, $a \in \mathbb{F}_q$, and $g \in \mathbb{G}$, we define $g^{\vec{v}} := (g^{v_1}, \dots, g^{v_n})$, $g^{a\vec{v}} := (g^{av_1}, \dots, g^{av_n})$, $g^{\vec{v}+\vec{w}} := (g^{v_1+w_n}, \dots, g^{v_n+w_n})$, and $\boldsymbol{e}(g^{\vec{v}}, g^{\vec{w}}) := \prod_{i=1}^{n} e(g^{v_i}, g^{w_i})$. Lewko defined algorithm $\mathsf{Dual}(\mathbb{F}_q^n)$ as follows: It chooses $\vec{b}_i, \vec{b}_j^* \in \mathbb{F}_q^n$ and $\psi \xleftarrow{\mathsf{U}} \mathbb{F}_q$ such that $\vec{b}_i \cdot \vec{b}_j^* = 0 \bmod q$ for $i \neq j$, $\vec{b}_i \cdot \vec{b}_i^* = \psi \bmod q$ for all $i \in [n]$ and outputs $(\mathfrak{B}, \mathfrak{B}^*)$ where $\mathfrak{B} := (\vec{b}_1, \dots, \vec{b}_n)$ and $\mathfrak{B}^* := (\vec{b}_1^*, \dots, \vec{b}_n^*)$. We use the notation $(\mathfrak{B}, \mathfrak{B}^*)$ to express dual orthonormal bases in $\mathbb{F}_q$ to distinguish from bases $(\mathbb{B}, \mathbb{B}^*)$ in $\mathbb{V}$. We can consider $\boldsymbol{b}_i = g^{\vec{b}_i}$, $\boldsymbol{b}_j^* = g^{\vec{b}_j^*}$, $\vec{b}_i = (\chi_{i,1}, \dots, \chi_{i,n})$, $\vec{b}_i^* = (\vartheta_{i,1}, \dots, \vartheta_{i,n})$ where $\boldsymbol{X} = (\chi_{i,j})$ and $\psi(\boldsymbol{X}^{-1})^\top = (\vartheta_{i,j})$.

2.3 Complexity Assumptions

Definition 4 (DLIN Assumption). *Let* $\mathcal{G}_{\mathsf{b}}^{\mathsf{dlin}}(1^\lambda)$ *be an algorithm that generates* $\Gamma := (q, \mathbb{G}, \mathbb{G}_T, e, g) \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{bm}}(1^\lambda)$*, chooses* $\xi, \kappa, \delta, \sigma, \zeta \xleftarrow{\mathsf{U}} \mathbb{F}_q$*, lets* $Q_0 := g^{\delta+\sigma}$*,* $Q_1 := g^{\zeta}$*, and outputs* $\mathcal{I} := (\Gamma, f, h, f^\delta, h^\sigma, Q_{\mathsf{b}})$*. The advantage is* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{dlin}}(\lambda) := |\Pr[\mathcal{A}(\mathcal{I}) \to 1 | \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}_0^{\mathsf{dlin}}(1^\lambda)] - \Pr[\mathcal{A}(\mathcal{I}) \to 1 | \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}_1^{\mathsf{dlin}}(1^\lambda)]|$*. We say that the DLIN assumption holds if for all PPT* $\mathcal{A}$*,* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{dlin}}(\lambda) < \mathsf{negl}(\lambda)$*.*

Definition 5 (Subspace Assumption). *Let* $\mathcal{G}_{\mathsf{b}}^{\mathsf{dss}}(1^\lambda)$ *be an algorithm that generates* $\Gamma \xleftarrow{\mathsf{R}} \mathcal{G}_{\mathsf{bm}}(1^\lambda)$*, chooses* $\eta, \beta, \tau_1, \tau_2, \tau_3, \mu_1, \mu_2, \mu_3 \xleftarrow{\mathsf{U}} \mathbb{F}_q$*,* $(\mathfrak{B}, \mathfrak{B}^*) \xleftarrow{\mathsf{R}} \mathsf{Dual}(\mathbb{F}_q^n)$*, for* $i \in [k]$ *where* $3k \leq n$ *lets* $U_i := g^{\mu_1 \vec{b}_i + \mu_2 \vec{b}_{k+i} + \mu_3 \vec{b}_{2k+i}}$*,* $V_i := g^{\tau_1 \eta \vec{b}_i^* + \tau_2 \beta \vec{b}_{k+i}^*}$*,* $W_i := g^{\tau_1 \eta \vec{b}_i^* + \tau_2 \beta \vec{b}_{k+i}^* + \tau_3 \vec{b}_{2k+i}^*}$*,* $D := (g^{\vec{b}_1}, \ldots, g^{\vec{b}_{2k}}, g^{\vec{b}_{3k+1}}, \ldots, g^{\vec{b}_n}, g^{\eta \vec{b}_1^*}, \ldots, g^{\eta \vec{b}_k^*}, g^{\beta \vec{b}_{k+1}^*}, \ldots, g^{\beta \vec{b}_{2k}^*}, g^{\vec{b}_{2k+1}^*}, \ldots, g^{\vec{b}_n^*}, U_1, \ldots, U_k, \mu_3)$*,* $Q_0 := (V_1, \ldots, V_k)$*,* $Q_1 := (W_1, \ldots, W_k)$*, and outputs* $\mathcal{I} := (\Gamma, D, Q_{\mathsf{b}})$*. The advantage is* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{dss}}(\lambda) := |\Pr[\mathcal{A}(\mathcal{I}) \to 1 | \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}_0^{\mathsf{dss}}(1^\lambda)] - \Pr[\mathcal{A}(\mathcal{I}) \to 1 | \mathcal{I} \xleftarrow{\mathsf{R}} \mathcal{G}_1^{\mathsf{dss}}(1^\lambda)]|$*. We say that the subspace assumption holds if for all PPT* $\mathcal{A}$*,* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{dss}}(\lambda) < \mathsf{negl}(\lambda)$*.*

Theorem 1 (Lewko [11]). *The DLIN assumption implies the subspace assumption.*

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 [23]. We consider a family of functions $\mathcal{F} := \{\mathcal{F}_\lambda\}_\lambda$. For example, LTFs are cryptographic functions and function F is sampled from family $\mathsf{LTF}_\lambda := \{f_{ek}(\cdot) | (ek, ik) \xleftarrow{\mathsf{R}} \mathsf{LTF.IGen}(1^\lambda)\}$. A watermarking key generation algorithm takes as inputs security parameter λ and family $\mathcal{F}$ and outputs public parameter pk, secret key sk, mark key mk, detect key dk, and remove key rk. That is, our watermarking schemes is an *asymmetric key* watermarking scheme. Public parameter pk includes sampling algorithm $\mathsf{Samp}_{\mathcal{F}}$, which outputs a function $F \xleftarrow{\mathsf{R}} \mathcal{F}_\lambda$ (note that we include the case in which the sampling algorithm takes sk as an input). Note that the description of $\mathsf{Samp}_{\mathcal{F}}$ does not include sk. Our cryptographic watermarking scheme for cryptographic functions $\mathcal{F}$ uses public parameter pk and secret key sk to choose a function $F \xleftarrow{\mathsf{R}} \mathcal{F}_\lambda$ from the function family. A mark key allows us to embed a mark in function F. A marked function F' should be similar to original function F. A detect/remove key allows us to detect/remove a mark in marked function F'.

Definition 6. *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 function family $\mathcal{F}$, 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,* $(pk, sk, mk, dk, rk) \overset{\mathsf{R}}{\leftarrow} \mathsf{WMGen}(1^{\lambda}, \mathcal{F})$.

Mark**:** *The mark algorithm takes as inputs mk and unmarked function F and outputs marked function $\widetilde{F}$, that is,* $\widetilde{F} \overset{\mathsf{R}}{\leftarrow} \mathsf{Mark}(pk, mk, F)$ *(hereafter, we often omit pk from inputs).*

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

Remove**:** *The remove algorithm takes as inputs rk and marked function $\widetilde{F}$ and outputs unmarked function* $F := \mathsf{Remove}(pk, rk, \widetilde{F})$*, that is functionally equivalent to the original one.*

As Hopper et al. noted [8], we do not allow any online communication between the Detect and Mark procedures. We sometimes use notation $\mathsf{WM}(F)$ to denote a marked function of F.

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. [8] and software watermarking proposed by Barak et al. [1]. We borrow some terms from these studies. 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, but we do not use it since we do not consider perceptual objects.

Basically, the following properties should be satisfied: Most objects $F \in \mathcal{F}_{\lambda}$ must be unmarked. We define similarity by a 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 are game-based definitions and based on the notion of strong watermarking by Hopper et al. [8]. Our definitions are specialized to focus on cryptographic functions (do not consider 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. [8]. 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. [8].

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

```
**Oracle** $\mathcal{MO}(F)$
$F' \overset{\mathsf{R}}{\leftarrow} \mathsf{Mark}(mk, F)$;
$\mathsf{MList} := \mathsf{MList} \cup \{F'\}$;
return $F'$;
```

```
**Oracle** $\mathcal{CO}_{\mathcal{F}_\lambda}()$
$F \overset{\mathsf{R}}{\leftarrow} \mathcal{F}_\lambda$;
$F' \overset{\mathsf{R}}{\leftarrow} \mathsf{Mark}(mk, F)$;
$\mathsf{CList} := \mathsf{CList} \cup \{F'\}$;
$\mathsf{MList} := \mathsf{MList} \cup \{F'\}$;
return $F'$
```

```
**Oracle** $\mathcal{DO}(F)$
$\mathsf{Detect}(dk, F) \to b$;
return $b$
```

```
**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 $\{\mathsf{unmarked}\}$
```

Fig. 1. Experiment for non-removability and unforgeability

Definition 7 (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 $F \in \mathcal{F}_\lambda$,* $\mathsf{Detect}(dk, F) \to \mathsf{unmarked}$.

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

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

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

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

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. 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 [8], 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 $\mathsf{unmarked}$ but the ideal detect procedure returns $\{\mathsf{marked}, \mathsf{unmarked}\}$ since an equivalent function is recorded in MList.

4 Proposed Watermarking Scheme

We present LTFs and a watermarking scheme for LTFs that are secure under the DLIN assumption. Generally speaking, LTFs can be constructed from homomorphic encryption schemes as discussed in many papers [5,20]. Lewko/Okamoto-Takashima proposed an IBE/IPE scheme based on DPVS, which is homomorphic and secure under the DLIN assumption. We can easily construct an LTF from the IBE scheme by applying the matrix encryption technique introduced by Peikert and Waters [20]. In this extended abstract, we present a scheme based on only the Lewko IBE scheme due to page limitations (since cryptographers are used to multiplicative notation). See a full version of this paper for a scheme based on the Okamoto-Takashima IPE scheme.

Basically, we use homomorphic *PKE* schemes to construct LTFs, but we use homomorphic *IBE* schemes to achieve watermarking scheme since we want 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*. Our LTF $\mathsf{LTF}_{\mathsf{mult}}$ based on the Lewko IBE is as follows:

$\mathsf{LTF.IGen}(1^\lambda)$**:** It generates $(\mathfrak{D}, \mathfrak{D}^*) \overset{\mathsf{U}}{\leftarrow} \mathsf{Dual}(\mathbb{F}_q^8)$, chooses $\alpha, \theta, \sigma \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q$, $\boldsymbol{\psi} := (\psi_1, \ldots, \psi_\ell) \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q^\ell$, 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 $s_{i,1}, s_{i,2} \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q$ for all $i \in [\ell]$ and generates

$u_{i,j} := g_{T_j}^{s_{i,1}} \cdot g_T^{m_{i,j}}$ and $\boldsymbol{v}_i := g^{s_{i,1}\vec{d}_1 + s_{i,1}ID\vec{d}_2 + s_{i,2}\vec{d}_3 + s_{i,2}ID\vec{d}_4}$ for all $i, j \in [\ell]$ where $m_{i,i} = 1$ and $m_{i,j} = 0$ (if $i \neq j$). For a conversion key, it chooses $r_1, r_2 \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q$ and generates $\boldsymbol{k}_{ID} := g^{(\alpha + r_1 ID)\theta\vec{d}_1^* - r_1\theta\vec{d}_2^* + r_2 ID\sigma\vec{d}_3^* - r_2\sigma\vec{d}_4^*}$. It returns $ek := (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{ID}) := (\{u_{i,j}\}_{i,j}, \{\boldsymbol{v}_i\}_i, \boldsymbol{k}_{ID})$ $(i, j \in [\ell])$, $ik := \boldsymbol{\psi}$. Note ek includes ID, 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 computes $y_j := \prod_i u_{i,j}^{x_i} = g_{T_j}^{\vec{x}\cdot\vec{s}_1} g_T^{x_j}$, $y_{\ell+1} := \prod_i \boldsymbol{v}_i^{x_i} = g^{\vec{x}\cdot\vec{s}_1\vec{d}_1 + \vec{x}\cdot\vec{s}_1 ID\vec{d}_2 + \vec{x}\cdot\vec{s}_2\vec{d}_3 + \vec{x}\cdot\vec{s}_2 ID\vec{d}_4}$ where $\vec{s}_1 := (s_{1,1}, \ldots, s_{1,\ell})$, $\vec{s}_2 := (s_{2,1}, \ldots, s_{2,\ell})$, and $y'_{\ell+1} := \boldsymbol{e}(y_{\ell+1}, \boldsymbol{k}_0) = e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*\vec{x}\cdot\vec{s}_1}$ 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 $x'_j := y_j/(y'_{\ell+1})^{\psi_j} = g_{T_j}^{\vec{x}\cdot\vec{s}_1} g_T^{x_j} / g_T^{\vec{x}\cdot\vec{s}_1\cdot\psi_j}$ 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 2. $\mathsf{LTF}_{\mathsf{mult}}$ *is a lossy trapdoor function if the DBDH assumption holds.*

We omit the proof and the definition of the DBDH assumption (this assumption is implied by the DLIN assumption).

Next, we present our watermarking scheme. 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 $\boldsymbol{k}_{ID}$, which is spanned by $\vec{d}_1^*, \ldots, \vec{d}_4^*$, it is indistinguishable from the original 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 $\boldsymbol{V}$ is 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 generated by $\vec{d}_7^*, \vec{d}_8^*$. If we have complete dual orthonormal bases $(\mathfrak{D}, \mathfrak{D}^*)$, then we can use the decomposition algorithm introduced in Section 2.2 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:

$\mathsf{WMGen}(\mathsf{LTF}_{\mathsf{mult}})$**:** It generates $(\mathfrak{D}, \mathfrak{D}^*) \overset{\mathsf{U}}{\leftarrow} \mathsf{Dual}(\mathbb{F}_q^8)$, chooses $\alpha, \theta, \sigma \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q$, $g_T := e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*}$, and sets $\widehat{\mathfrak{D}} := (g_T, g^{\vec{d}_1}, \ldots, g^{\vec{d}_4})$ $pk := (\widehat{\mathfrak{D}}, \mathsf{Samp})$, $sk := (g^{\alpha\theta\vec{d}_1^*}, g^{\theta\vec{d}_1^*}, g^{\sigma\vec{d}_2^*}, g^{\sigma\vec{d}_3^*}, g^{\vec{d}_4^*})$, $mk := (g^{\vec{d}_7^*}, g^{\vec{d}_8^*})$, $dk := (g^{\vec{d}_7}, g^{\vec{d}_8})$, and $rk := (\mathfrak{D}, \mathfrak{D}^*)$. The sampling algorithm $\mathsf{Samp}(\mathsf{pp}_{\mathbb{V}}, \widehat{\mathfrak{D}}, sk)$ chooses $\boldsymbol{\psi} \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q^\ell$, $\vec{s}_1, \vec{s}_2 \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q^\ell$, and generates $(ek, ik) := ((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{ID}), \boldsymbol{\psi})$ as $\mathsf{LTF.IGen}$. It computes $\boldsymbol{k}_{ID} := g^{(\alpha + r_1 ID)\theta\vec{d}_1^* - r_1\theta\vec{d}_2^* + r_2 ID\sigma\vec{d}_3^* - r_2\sigma\vec{d}_4^*}$. 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 $ek = (\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{k}_{ID})$, chooses $t_1, t_2 \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q$, and computes $\widetilde{\boldsymbol{k}}_{ID} := \boldsymbol{k}_{ID} \cdot g^{t_1\vec{d}_7^* + t_2\vec{d}_8^*}$ by using $g^{\vec{d}_7^*}$ and $g^{\vec{d}_8^*}$. It outputs marked function index $\mathsf{WM}(ek) = (\boldsymbol{U}, \boldsymbol{V}, \widetilde{\boldsymbol{k}}_{ID})$.

$\mathsf{Detect}(dk, \widetilde{ek})$**:** It parses $\widetilde{ek} = (\boldsymbol{U}, \boldsymbol{V}, \widetilde{\boldsymbol{k}}_{ID})$, chooses $z_1, z_2 \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q^{\times}$, and computes $\boldsymbol{c} := g^{z_1\vec{d}_7 + z_2\vec{d}_8}$. Next, it computes $\Delta := \boldsymbol{e}(\boldsymbol{c}, \widetilde{\boldsymbol{k}}_{ID})$. If it holds that $\Delta = \boldsymbol{e}(\boldsymbol{c}, \widetilde{\boldsymbol{k}}_{ID}) \neq 1$, then it outputs marked. Else if, it holds that $\Delta = 1$, then it outputs $\mathsf{unmarked}$.

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

Correctness, preserving functionality, and polynomial slowdown are easily followed. Meaningfulness follows since $(g^{\vec{d}_1^*}, \ldots, g^{\vec{d}_8^*})$ are hidden. 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 $\boldsymbol{k}_{ID}$. This seems to be a restriction, but in the scenario of watermarking schemes, this is acceptable. 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.

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

Our watermarking scheme $\mathsf{CWM}_{\mathsf{mult}}$ is secure under the DLIN assumption. We prove this by proving Theorems 3 and 4.

Theorem 3. $\mathsf{CWM}_{\mathsf{mult}}$ *is non-removable 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, $D = (g^{\vec{b}_1}, g^{\vec{b}_2}, g^{\vec{b}_4}, \ldots, g^{\vec{b}_8}, g^{\eta\vec{b}_1^*}, g^{\beta\vec{b}_2^*}, g^{\vec{b}_3^*}, \ldots, g^{\vec{b}_8^*}, U_1)$, and Q_{b} for $\mathsf{b} \in \{0,1\}$. We set $Q_0 := V_1 = g^{\tau_1\eta\vec{b}_1^* + \tau_2\beta\vec{b}_2^*}$ and $Q_1 := W_1 = g^{\tau_1\eta\vec{b}_1^* + \tau_2\beta\vec{b}_2^* + \tau_3\vec{b}_3^*}$. $\mathcal{B}$ sets

$$\vec{d}_1 := \vec{b}_3^* \quad \vec{d}_2 := \vec{b}_4^* \quad \vec{d}_3 := \vec{b}_5^* \quad \vec{d}_4 := \vec{b}_6^* \quad \vec{d}_5 := \vec{b}_7^* \quad \vec{d}_6 := \vec{b}_8^* \quad \vec{d}_7 := \vec{b}_1^* \quad \vec{d}_8 := \vec{b}_2^*$$
$$\vec{d}_1^* := \vec{b}_3 \quad \vec{d}_2^* := \vec{b}_4 \quad \vec{d}_3^* := \vec{b}_5 \quad \vec{d}_4^* := \vec{b}_6 \quad \vec{d}_5^* := \vec{b}_7 \quad \vec{d}_6^* := \vec{b}_8 \quad \vec{d}_7^* := \vec{b}_1 \quad \vec{d}_8^* := \vec{b}_2$$

$\mathcal{B}$ chooses $\theta, \alpha', \sigma \overset{\mathsf{U}}{\leftarrow} \mathbb{Z}_p$ and can generate public key $pk = (e(g,g)^{\alpha\theta\vec{d}_1\cdot\vec{d}_1^*}, g^{\vec{d}_1}, \ldots, g^{\vec{d}_4}) := (e(g^{\vec{b}_2^*}, g^{\vec{b}_2})^{\alpha'\mu_3\theta}, g^{\vec{b}_3^*}, g^{\vec{b}_4^*}, g^{\vec{b}_5^*}, g^{\vec{b}_6^*})$ and mark key $mk = (g^{\vec{d}_7^*}, g^{\vec{d}_8^*})$

$:= (g^{\vec{b}_1}, g^{\vec{b}_2})$. $\mathcal{B}$ 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. Coefficients β and η do not affect the detect algorithm. $\mathcal{B}$ can have $(g^{\vec{d}_2^*}, \dots, g^{\vec{d}_8^*})$ 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 but 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 ID, $\mathcal{B}$ chooses $r_1', r_2, t_7, t_8 \overset{\mathsf{U}}{\leftarrow} \mathbb{Z}_p$ and computes

$$\begin{aligned}\widetilde{\boldsymbol{k}}_{ID} &:= (U_1)^{(\alpha'+r_1'ID)\theta} g^{-r_1'\mu_3\theta\vec{d}_2^*+r_2ID\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*+t_7\vec{d}_7^*+t_8\vec{d}_8^*} \\ &= g^{(\alpha+r_1ID)\theta\vec{d}_1^*-r_1\theta\vec{d}_2^*+r_2ID\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*+t_7'\vec{d}_7^*+t_8'\vec{d}_8^*}\end{aligned}$$

where $t_7' = t_7 - \theta(\alpha' + r_1'ID)\mu_1$ and $t_8' = t_8 - \theta(\alpha' + r_1'ID)\mu_2$ We set $r_1 := \mu_3 r_1'$. This is a valid marked index. If $\mathcal{A}$ outputs valid unmarked index $ek^* = (\boldsymbol{U}^*, \boldsymbol{V}^*, \boldsymbol{k}_{ID^*}^*)$ where $\boldsymbol{k}_{ID^*}^* = g^{(\alpha+r_1^*ID^*)\theta\vec{d}_1^*-r_1^*\theta\vec{d}_2^*+r_2^*ID^*\sigma\vec{d}_3^*-r_2^*\sigma\vec{d}_4^*}$, then $\mathcal{B}$ computes $\varDelta := \boldsymbol{e}(Q_{\mathsf{b}}, \boldsymbol{k}_{ID^*}^*)$. If $\varDelta = 1$, then $\mathcal{B}$ outputs 0 ($\mathsf{b} = 0$), otherwise, it outputs 1. If $Q_{\mathsf{b}} = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*} = g^{\tau_1\eta\vec{d}_7+\tau_2\beta\vec{d}_8}$, then $\varDelta = 1$. If $Q_{\mathsf{b}} = g^{\tau_1\eta\vec{b}_1^*+\tau_2\beta\vec{b}_2^*+\tau_3\vec{b}_3^*} = g^{\tau_1\eta\vec{d}_7+\tau_2\beta\vec{d}_8+\tau_3\vec{d}_1}$, then $\varDelta = e(g,g)^{(\alpha+r_1ID)\theta\tau_3\vec{d}_1\cdot\vec{d}_1^*} \neq 1$. That is, $\mathcal{B}$ breaks the problem. $\square$

Next, we prove unforgeability. 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 unavoidable and in the experiment on unforgeability, IdealDtc always returns marked for identities that oracles used. For simplicity, we prove the unforgeability explained above, but we can extend it to stronger ones by using known techniques that convert standard unforgeable signature schemes into *strongly unforgeable* signature schemes. We now define algorithm $\mathsf{Xtr}(pk, sk, ID)$. It chooses $r_1, r_2 \overset{\mathsf{U}}{\leftarrow} \mathbb{F}_q$ and outputs $\boldsymbol{k}_{ID} := g^{(\alpha+r_1ID)\theta\vec{d}_1^*-r_1\theta\vec{d}_2^*+r_2ID\sigma\vec{d}_3^*-r_2\sigma\vec{d}_4^*}$. We can consider $\boldsymbol{k}_{ID}$ be a signature for ID. In fact, Naor pointed out that signature schemes can be derived from IBE schemes [2]. Thus, we can prove the unforgeability of our watermarking scheme 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 [9]. We can achieve strong unforgeability of watermarking schemes by using their technique and the strongly unforgeably property.

Theorem 4. $\mathsf{CWM}_{\mathsf{mult}}$ *is unforgeable 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 $\boldsymbol{k}_{ID}$.

Normal: $g^{(\alpha+r_1 ID)\theta\vec{d}_1^* - r_1\theta\vec{d}_2^* + r_2 ID\sigma\vec{d}_3^* - r_2\sigma\vec{d}_4^* + t_7\vec{d}_7^* + t_8\vec{d}_8^*}$
Semi-functional: $g^{(\alpha+r_1 ID)\theta\vec{d}_1^* - r_1\theta\vec{d}_2^* + r_2 ID\sigma\vec{d}_3^* - r_2\sigma\vec{d}_4^* + t_5\vec{d}_5^* + t_6\vec{d}_6^* + t_7\vec{d}_7^* + t_8\vec{d}_8^*}$.

We can generate them if we have the secret key, mark 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 $\mathsf{Game}\text{-}i$ 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 $\mathsf{Game}\text{-}(i)$ for outputting a normal (resp. semi-functional) conversion key for a non-given or non-queried ID.

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

Lemma 3. *If there exists $\mathcal{A}$, that distinguishes* $\mathsf{Game}\text{-}(i-1)$ *from* $\mathsf{Game}\text{-}(i)$*, then we can break the subspace assumption with $k = 2$ and $n = 8$.*

Lemma 4. *If $\mathcal{A}$ outputs a normal marked index in* $\mathsf{Game}\text{-}(q_{\mathsf{C}})$*, then we can break the subspace assumption with $k = 1$ and $n = 8$.*

By Lemmas 2, 3, and 4, we can show the following:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Forge}}(\lambda) = \mathsf{Adv}_0^{\mathsf{forge\text{-}N}} + \mathsf{Adv}_0^{\mathsf{forge\text{-}S}} < (q_{\mathsf{C}} + 2)\mathsf{Adv}_{\mathcal{B}}^{\mathsf{dss}}.$$

The theorem follows from the lemmas and Theorem 1. □

Acknowledgements. The author would like to thank Keita Xagawa for illuminating discussions and insightful comments about constructions of watermarking schemes and Kazumaro Aoki and Atsushi Fujioka for helpful comments about applications of watermarking and the presentation of the introduction.

References

1. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

2. D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.
3. B. Chor, A. Fiat, and M. Naor. Tracing traitors. In *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 1994.
4. C. S. Collberg and C. D. Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Trans. Software Eng.*, 28(8):735–746, 2002.
5. D. M. Freeman, O. Goldreich, E. Kiltz, A. Rosen, and G. 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.
6. S. D. Galbraith and E. R. Verheul. An analysis of the vector decomposition problem. In *Public Key Cryptography*, volume 4939 of *LNCS*, pages 308–327. Springer, 2008.
7. B. Hemenway and R. Ostrovsky. Extended-DDH and lossy trapdoor functions. In *Public Key Cryptography*, volume 7293 of *LNCS*, pages 627–643. Springer, 2012.
8. N. Hopper, D. Molnar, and D. Wagner. From weak to strong watermarking. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 362–382. Springer, 2007.
9. Q. Huang, D. S. Wong, and Y. Zhao. Generic transformation to strongly unforgeable signatures. In *ACNS*, volume 4521 of *LNCS*, pages 1–17. Springer, 2007.
10. A. Kiayias and M. Yung. Traitor tracing with constant transmission rate. In *EUROCRYPT*, volume 2332 of *LNCS*, pages 450–465. Springer, 2002.
11. A. 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.
12. A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, volume 6110 of *LNCS*, pages 62–91. Springer, 2010.
13. M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, 1988.
14. D. Naccache, A. Shamir, and J. P. Stern. How to copyright a function? In *Public Key Cryptography*, volume 1560 of *LNCS*, pages 188–196. Springer, 1999.
15. T. Okamoto and K. Takashima. Homomorphic encryption and signatures from vector decomposition. In *Pairing*, volume 5209 of *LNCS*, pages 57–74. Springer, 2008.
16. T. Okamoto and K. Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, volume 5912 of *LNCS*, pages 214–231. Springer, 2009.
17. T. Okamoto and K. Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, volume 6223 of *LNCS*, pages 191–208. Springer, 2010.
18. T. Okamoto and K. 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.
19. T. Okamoto and K. Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 591–608. Springer, 2012.
20. C. Peikert and B. Waters. Lossy trapdoor functions and their applications. *SIAM J. Comput.*, 40(6):1803–1844, 2011.
21. J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394. ACM, 1990.
22. B. 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.
23. M. Yoshida and T. Fujiwara. Toward digital watermarking for cryptographic data. *IEICE Transactions*, 94-A(1):270–272, 2011.
24. M. Yoshida, S. Mitsunari, and T. Fujiwara. The vector decomposition problem. *IEICE Transactions*, 93-A(1):188–193, 2010.