Certificateless Homomorphic Signature Scheme for Network Coding

Jinyong Chang, Bilin Shao, Yanyan Ji, and Genqing Bian

Abstract—Homomorphic signature is an extremely important public key cryptographic technique for network coding to defend against pollution attacks. As a public key cryptographic primitive, it also encounters the same problem that how to confirm the relationship between some public key pk and the identity ID of its owner. In the setting of network coding, the intermediate and destination nodes need to use source node S's public key to check the validity of vector-signature pairs. Therefore, the binding of S and its corresponding public key becomes crucial. The popular and traditional solution is based on certificates which is issued by a trusted certification authority (CA) center. However, the generation and management of certificates is extremely cumbersome. Hence, in recent work [20], Lin et al. proposed a new notion of identity-based homomorphic signature, which intends to avoid using certificates. But the key escrow problem is inevitable for identity-based primitives. In this paper, we propose another new notion (for network coding): certificateless homomorphic signature (CLHS), which is a compromise for the above two techniques. In particular, we first describe the definition and security model of certificateless homomorphic signature. Then based on bilinear map and the computational Diffie-Hellman (CDH) assumption, give a concrete implementation and detailedly analyze its security. Finally, performance analysis illustrates that our construction is practical.

Index Terms—Homomorphic Signature, Certificateless Signature, CDH Assumption, Network Coding.

1 Introduction

Unlike traditional store-and-forward routing mechanisms, network coding allows its intermediate nodes encode their incoming packets before forwarding them, which has been mathematically proven to enhance the network robustness and maximize the network throughput [1], [18], [26]. Hence, in recent years, it has received extensive attentions and been applied to various computer network systems, including wireless networks [22], P2P systems [9] and multicast networks [28]. However, it is also well-known that it is extremely vulnerable under pollution attack [7], since the polluted packets will further contaminate other packets after the random combinations of intermediate nodes, which not only leads to incorrect decoding for the destination nodes but also wastes network resources.

Linearly homomorphic signature scheme is a popular cryptographic primitive, which can detect and filter the polluted packets and hence defend against pollution attack for network coding [4], [12]. In contrast to homomorphic message authentication code scheme [10], it is in the public key setting. Therefore, all the nodes in the network only need to obtain public key of source node S instead of sharing a common secret key (with S), which is more appropriate for offline systems, such as robust distributed file storage [15].

It is also well-known that for any public key cryptographic primitives, how to assure the corresponding relationship between a public key and user's identity is an extremely important and complicated problem. The most popular and traditional way is based on public key infrastructure (PKI), in which this problem is resolved by issuing certificates (essentially a signature for the public key) by a trusted certification authority (CA). Any user, who wants to use someone's public key, first checks the validity of this public key's certificate so that confirming the relationship between public key and its corresponding identity. However, in practice, how to generate and manage so many certificates is a cumbersome task for PKI. In addition, the verification for each certificate also somewhat degrades the performances of original algorithms.

To avoid the using of certificates, in 1984, Shamir first introduced identity-based cryptography [23]. The basic idea is using user's information, such as email address, ID card number or telephone number etc., as his/her public key, which removes the necessity of public key certificate. But the user's secret key is generated by combining this public key with a master key owned by an entity named Key-Generation-Center (KGC).

Therefore, in 2018, Lin et al. introduced the notion of identity-based homomorphic signature scheme, presented an elegant construction based on the computational Diffie-Hellman (CDH) assumption, and tried to use it to Blockchain [20]. Of course, this scheme can also be used in network coding to simplify the management of public key. In the next section, we will present the model of applying identity-based homomorphic signature to network coding.

However, the inherent problem for any identity-based primitive lies in the escrow of key, which means that any user's private key is known by the entity of KGC. Hence, KGC can literally forge any signature of any user if he wants. Obviously, this is the case that any signer does not want to see. As a result, the network coding system based on identity-based homomorphic signature also encounters the same problem. How to tackle it seems to be not mentioned in the existing literatures.

Our Contributions. In this paper, we try to give a solution of the above problem. Concretely, we introduce the notion of certificateless homomorphic signature (CLHS) scheme, which is a compromise primitive between identity-based homomorphic signature and a normal homomorphic signature, and apply it to

- *Jinyong Chang and Genqing Bian are with school of information and control engineering, Xi'An University of Architecture and Technology, Xi'An, 710055, Shaanxi, P.R. China and also with department of mathematics, Changzhi University, Changzhi, 046011, Shanxi, P.R. China.*
- *Bilin Shao and Yanyan Ji are with school of management, Xi'An University of Architecture and Technology, Xi'An, 710055, Shaanxi, P.R. China.*

*Manuscript received *; revised *.*

network coding. In particular, our contributions consist of the following aspects.

- For the first time, we introduce the notion of CLHS, define its security model.
- Based on the CDH assumption and the known certificateless signature (CLS), we present a concrete instantiation of CLHS and the corresponding security proof.
- By fixing series of parameters, we give the simulations of the algorithms, including the signature, verification as well as combination for different packets. The performance analysis illustrates that our construction is practical.

Related Works. Homomorphic signature scheme was firstly proposed by Johnson et al. in [16]. This definition is applied to network coding to defend against pollution attack by Boneh et al. in [7]. The classic construction of homomorphic signature is based on pairings, random oracle, and the CDH assumption. After that, some other constructions based on lattice, standard model also appear, such as [4], [6], [8], [25]. The property of homomorphic is extended to polynomial function in [5]. In recent works [19], [20], Lin et al. respectively considered the new definitions of identity-based homomorphic signature and homomorphic proxy signature schemes. Later, Chang et al. further considered the related-key-attack on identity-based homomorphic signature [11].

Certificateless public key cryptography is proposed by Al-Riyami and Paterson in 2003 [3], who intends to eliminate the key escrow problem for KGC. In this work, they suggested certificateless public key encryption and certificateless signature schemes. After this pioneering result, many certificateless techniques were developed in [13], [24]. In [27], Zhang et al. further considered the security model for certificateless signature and then presented an elegant construction based on the CDH assumption and pairing. In [14], Huang et al. also constructed a certificateless signature scheme with enough short-size signature based on bilinear map. In recent work [17], Karati et al., tried to apply certificateless signature scheme to industrial internet of things (IIoT) to realize the lightweight authentication for IIoT.

Organizations. The later parts will be organized as follows. First, in Section 2, we will introduce some basic notations, system model of network coding and some basic notions, which mainly includes certificateless signature scheme, certificateless homomorphic signature scheme as well as their security models. Then a concrete construction of CLHS and its security proof will be given in Section 3. Finally, the performance analysis and conclusions can be found in Section 4 and Section 5, respectively.

2 Preliminaries

Basic Notations. In this paper, we denote by λ the security parameter. For a set S, $s \xleftarrow{\$} S$ means that randomly choose an element s from S. For a natural number q, $[q]$ denotes the set of $\{1, 2, \cdots, q\}$. If q is a prime, then $\mathbb{Z}_q$ is a finite field and $\mathbb{Z}_q^* = \mathbb{Z}_q \backslash \{0\}$. PPT means probabilistic polynomial time. A function $f(\lambda)$ is called negligible if for any $c > 0$, there exists a $k_0 \in \mathbb{Z}$ satisfying that, for all $\lambda > k_0$, it holds that $f(\lambda) < \lambda^{-c}$. We use a boldface type $\mathbf{v}$ to denote a vector and v_i to denote its i-th component.

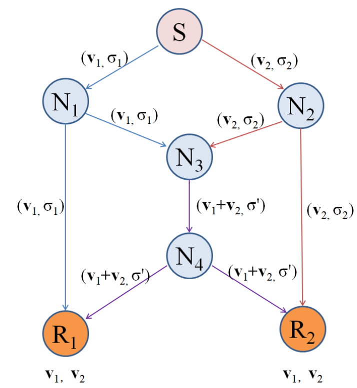

Fig. 1. Secure Network Coding

2.1 System Model

2.1.1 Secure Network Coding

In the model of combining network coding with homomorphic signature, there are three types of nodes: Source nodes, intermediate nodes and destination nodes. (A lite version can be found in Fig. 1)

- A source node S first generates the signing key pair (pk, sk). For the data packets, which are depicted as some vectors $\mathbf{v}_1, \cdots, \mathbf{v}_m$, S respectively computes the corresponding signatures $\sigma_1, \cdots, \sigma_m$ and sends the pairs $(\mathbf{v}_1, \sigma_1), \cdots, (\mathbf{v}_m, \sigma_m)$ to the adjacent intermediate nodes.
- For some intermediate node N_i, it may receive $(\mathbf{v}_{i_1}, \sigma_{i_1})$, $\cdots, (\mathbf{v}_{i_\ell}, \sigma_{i_\ell})$ and first checks the validity of all the pairs. If any of them can not pass the verification, then this pair is seen as a "polluted" one and hence is discarded. Then, for the "unpolluted" pairs, use the combination algorithm (of the homomorphic signature scheme) to obtain a "combined" signature σ' for the vector $\mathbf{v}'$, which is a (random) linear combination of the "unpolluted" vector, and transmit the new pair $(\mathbf{v}', \sigma')$ to its adjacent nodes.
- For the destination node R_i, it must collect "enough" pairs $(\mathbf{v}'_1, \sigma'_1), \cdots (\mathbf{v}'_n, \sigma'_n)$ so that the original vectors $\mathbf{v}_1, \cdots, \mathbf{v}_m$ can be recovered from them. It also checks the validity of all the pairs. For the "polluted" ones, discard them and finally recover the original vectors $\mathbf{v}_1, \cdots, \mathbf{v}_m$.

2.1.2 Certification-Based Network Coding

In the above model of network coding, the implicit condition is that the intermediate and destination nodes *correctly* get the public key pk of the source node S. However, how to authenticate the relationship between pk and the target identity ID? The traditional and popular method is issuing a certification for S by a trusted CA. That is, S requests a certification C for its public key pk in advance and publishes (pk, C). For other nodes, who need to verify the signatures using S's public key, they first check the public-certification pair (pk, C). If it is valid, then pk is seen as true public key of S.

In fact, there may be many source nodes $\mathrm{S}_1, \cdots, \mathrm{S}_n$ in the whole system, who need to register and obtain the corresponding certifications for their public keys from the CA center. In this case, we call it certification-based network coding. In Fig. 2, we present a simplified version, in which there are two source nodes S_1 and S_2.

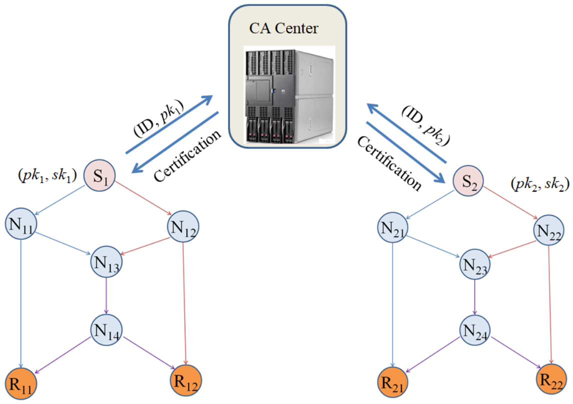

Fig. 2. Certification-Based Network Coding

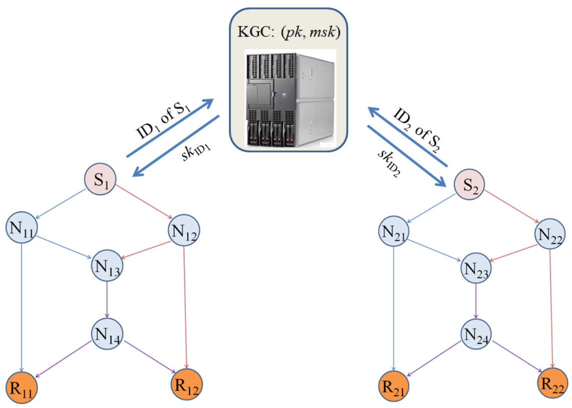

Fig. 3. Identity-Based Network Coding

2.1.3 Identity-Based Network Coding

As Lin et al. said in [20], the management of certifications is extremely cumbersome and the using of them sometimes degrades the performances of algorithms or schemes. Hence, they introduced the identity-based homomorphic signature scheme. Here, we remark that their scheme can also be used in network coding, which is called identity-based network coding.

In the model of identity-based network coding, the public key certification for each source node is not used since the only public key pk (of KGC) is universal. That is, for the intermediate and destination nodes in different coding process, the verifications of signatures only need the common public key pk of KGC.

Concretely, for any source node S_i, it submits identity ID_i to KGC. Then KGC will return a secret key sk_{ID_i}, which will be used to sign all the vectors. For the adjacent intermediate or destination nodes of S_i, they will verify the vector-signature pairs using pk and perform the following steps in network coding. A lite version can be found in Fig. 3.

2.1.4 Certificateless Network Coding

As we said in Introduction, the essential problem for identity-based network coding lies in the key escrow. For example, in Fig. 3, KGC knows the two signing keys sk_{ID_1}, $sk_{ID,2}$. If KGC is malicious, then the consequences will be disastrous. Hence, avoiding key escrow seems to be an urgent tasks for identity-based network coding. As a result, we propose a new terminology named certificateless network coding, which is a combination

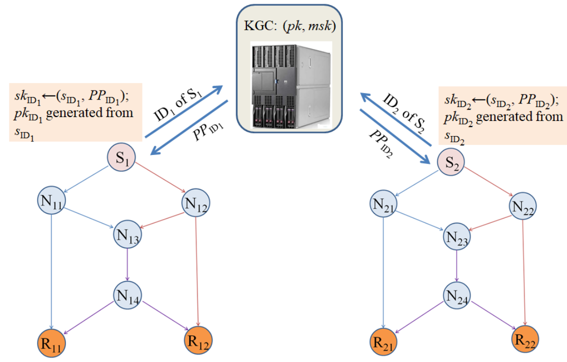

Fig. 4. Certificateless Network Coding

of certificateless homomorphic signature (we will define in later subsection) and network coding.

In particular, in certificateless network coding, KGC only returns a *partial* private key PP_{ID} (for source node S) after receiving S's identity ID. For S, he will choose a secret value s_{ID} and generate the *full* private key sk_{ID} (i.e. signing key) by combining s_{ID} and PP_{ID}. Hence, the signatures for vectors will be computed by using sk_{ID}. Meanwhile, S's public key pk_{ID} is also computed from s_{ID}. Then the intermediate and destination nodes will obtain pk_{ID} and verify the vector-signature pairs based on pk_{ID}. A lite version can be found in Fig. 4. We remark that the values PP_{ID}, s_{ID}, sk_{ID} and pk_{ID} are respectively computed by the corresponding algorithms in certificateless homomorphic signature scheme.

Because KGC only contributes a partial private key, which is not the true signing key sk_{ID}, it can not forge the signatures for S even if KGC is malicious. In addition, in above certificateless network coding, the relationship between public key pk_{ID} and S's identity ID is still not authenticated. In other words, the intermediate and destination nodes may receive an incorrect public key pk'_{ID} instead of pk_{ID}. Here, we call it key replacement attack. Therefore, in the security model of our certificateless homomorphic signature scheme, two types adversaries will be considered: a usual adversary $\mathcal{A}^I$ and a KGC-type adversary $\mathcal{A}^{II}$, which will respectively simulate the key replacement attack and malicious KGC attack.

Since certificateless homomorphic signature scheme is the core of certificateless network coding, we will mainly focus on how to present its definition, security model and construction in the following sections.

2.2 Bilinear Map

Let G_1 G_2 be two cyclic groups with the same (prime) order q and e be a map from $G_1 \times G_1$ to G_2 which can be efficiently computable and satisfies the following: For any two generators $g, h \in G_1$,

1) **Non-Degenerate Property.** $e(g, h) \neq 1_{G_2}$, where 1_{G_2} is the identity element of G_2.
2) **Bilinear Property.** For any $a, b \in \mathbb{Z}_q$, it holds that

$$e(g^a, h^b) = e(g, h)^{ab}.$$

2.3 CDH Assumption

The CDH assumption on a cyclic group G with generator g refers to that it is "hard" to compute g^{ab} for any PPT adversary when given the items g, g^a, and g^b. Formally, let the order of G be q and randomly choose $a, b \xleftarrow{\$} \mathbb{Z}_q^*$. Then give the tuple (q, G, g, g^a, g^b) to a PPT adversary whose goal is to compute and output g^{ab}. However, the adversary can successfully output g^{ab} with at most negligible probability. Here, the probability is taken over the selection of a, b.

2.4 Certificateless Signature Scheme

Denote by CLS a certificateless signature scheme, which consists of the following algorithms.

- `CL-Setup`: For the input of security parameter λ, generate and output the public parameter *params* as well as master key *msk*. This algorithm is usually run by KGC. We always assume that *params* is publicly and authentically available, but *msk* is only known by KGC.
- `CL-Partial-Private-Key-Extract`: For the inputs of *msk* and user's identity ID, this algorithm generates a partial private key PP_{ID} for ID. Usually, this algorithm is also run by KGC and its output PP_{ID} is confidentially given to user ID over an authentic channel.
- `CL-Set-Secret-Value`: For the input of user's identity ID, this algorithm generates a secret value s_{ID}.
- `CL-Set-Private-Key`: For the inputs of user's secret value s_{ID} and the partial private key PP_{ID}, this algorithm generates the *full* private key SK_{ID} for user ID.
- `CL-Set-Public-Key`: For the input s_{ID}, this algorithm generates and outputs the corresponding public key PK_{ID} for user with the identity of ID.
  Normally, the algorithms `CL-Set-Private-Key` and `CL-Set-Public-Key` are run by user for itself after running `CL-Set-Secret-Value`.
- `CL-Sign`: For the inputs of user's identity ID, its public key PK_{ID}, private key SK_{ID} and a message m, this algorithm generates a signature σ for the message m.
- `CL-Verify`: Given the inputs of user's public key PK_{ID}, its identity ID and the message-signature pair (m, σ), this algorithm outputs 1 (accept) or 0 (reject).

Security. Here, we adopt the security model in [27], in which two types of adversaries $\mathcal{A}^I$ and $\mathcal{A}^{II}$ are considered. The former one denotes a usual adversary who can replace any user's public key at its will, while the latter one describes a malicious KGC. Hence, two games Game-I and Game-II, which are respectively played by the two adversaries and their challengers, are considered.

Game-*I*:

- **Initialization-*I*.** A challenger CH^I runs the algorithm

$$(params, msk) \leftarrow \texttt{CL-Setup}(\lambda).$$

  Give *params* to $\mathcal{A}^I$ and keep *msk* secret.
- **Queries-*I*.** The adversary $\mathcal{A}^I$ can adaptively make the following queries. For convenience, the challenger initializes an empty "query-answer" list, which will record $\mathcal{A}^I$'s queries and his own answers.
  - **Extract-Partial-Private-Key:** For the queried identity ID, the challenger runs

$$PP_{ID} \leftarrow \texttt{CL-Partial-Private-Key-Extract}(ID) \tag{1}$$

    and return PP_{ID} to the adversary $\mathcal{A}^I$. Add (ID, PP_{ID}) to the "query-answer" list.
  - **Extract-Private-Key:** For the input ID, CH^I first checks if (ID, PP_{ID}) is in the "query-answer" list. If it is, recover PP_{ID}. Otherwise, run (1). Then compute

$$s_{ID} \leftarrow \texttt{CL-Set-Secret-Value}(ID), \tag{2}$$

    and

$$SK_{ID} \leftarrow \texttt{CL-Set-Private-Key}(PP_{ID}, s_{ID}). \tag{3}$$

    Give SK_{ID} to $\mathcal{A}^I$ and add $(ID, PP_{ID}, s_{ID}, SK_{ID})$ to the "query-answer" list.
  - **Request-Public-Key:** For the queried ID, the challenger checks if (ID, s_{ID}) appears in "query-answer" list. If it is, recover s_{ID}. Otherwise, run (2). Then compute

$$PK_{ID} \leftarrow \texttt{CL-Set-Public-Key}(s_{ID}). \tag{4}$$

    Give PK_{ID} to $\mathcal{A}^I$ and add (ID, s_{ID}, PK_{ID}) to the "query-answer" list.
  - **Replace-Public-Key:** The adversary $\mathcal{A}^I$ is allowed to replace PK_{ID} with any value PK'_{ID} he would like to set. Add (ID, PK_{ID}, PK'_{ID}) to the "query-answer" list. Note that, in this oracle, the adversary does not need to give the corresponding secret value for the replaced public key.
  - **Signing-Queries:** When $\mathcal{A}^I$ submits (ID, m) to this oracle, the challenger first checks if $(ID, PP_{ID}, s_{ID}, SK_{ID}, PK_{ID})$ appears in the "query-answer" list. If it is, recover SK_{ID}. Otherwise, run (1), (2), and (4) to generate PP_{ID}, s_{ID} and PK_{ID}, respectively. Then compute SK_{ID} by running (3). After those steps, the challenger runs

$$\sigma \leftarrow \texttt{CL-Sign}(ID, PK_{ID}, SK_{ID}, m) \tag{5}$$

    and returns it to the adversary. Finally, add

$$(ID, PP_{ID}, s_{ID}, SK_{ID}, PK_{ID}, m, \sigma)$$

    to the "query-answer" list.
    Note that, if PK_{ID} has been changed into PK'_{ID} by $\mathcal{A}^I$'s Replace-Public-Key query, then CH^I may not have the correct SK_{ID} and hence can not correctly answer this query. In this case, $\mathcal{A}^I$ needs to additionally give s_{ID}, which is corresponding to PK'_{ID}, to the challenger.

Output-*I*. At the end, the adversary outputs a tuple of

$$(ID^*, PK_{ID^*}, m^*, \sigma^*).$$

We remark that, ID^* should not be queried to Extract-Private-Key oracle. Moreover, ID^* should not be simultaneously queried to both Replace-Public-Key oracle and Extract-Partial-Private-Key oracle. m^* should not be queried for signature w.r.t. ID^* and PK_{ID^*}.

If it holds that

$$1 \leftarrow \texttt{CL-Verify}(ID^*, PK_{ID^*}, m^*, \sigma^*), \tag{6}$$

then we call $\mathcal{A}^I$ wins the game. Denote by $\mathrm{Adv}^{\mathrm{EUF}}_{\mathcal{A}^I,\mathrm{CLS}}(\lambda)$ $\mathcal{A}^I$'s advantage, which equals to the probability of $\mathcal{A}^I$ winning the game.

<u>**Game-*II*:**</u>

- **Initialization-*II*.** A challenger CH^{II} runs the algorithm
$$(params, msk) \leftarrow \texttt{CL-Setup}(\lambda).$$
Give *params* and *msk* to $\mathcal{A}^{II}$.
- **Queries-*II*.** The adversary $\mathcal{A}^{II}$ is allowed to adaptively make the following queries. Note that, the adversary has the master key *msk* and it can compute any user's partial private key PP_{ID}. Hence, we require that it also submits PP_{ID} (if necessary), which will be no longer generated by the challenger. For convenience, the challenger still initializes an empty "query-answer" list.
  - **Extract-Private-Key:** $\mathcal{A}^{II}$ submits (ID, PP_{ID}) to the challenger. Then the challenger runs (2) and (3). Return SK_{ID} to $\mathcal{A}^{II}$ and add $(ID, PP_{ID}, s_{ID}, SK_{ID})$ to the "query-answer" list.
  - **Request-Public-Key:** $\mathcal{A}^{II}$ submits (ID, PP_{ID}) to the challenger in this oracle. Then CH^{II} checks if (ID, s_{ID}) appears in the "query-answer" list. If it is, recover s_{ID}. Otherwise, run (2) to obtain s_{ID}. Then run (4) to generate PK_{ID}. Give PK_{ID} to the adversary and add $(ID, PP_{ID}, s_{ID}, PK_{ID})$ to the "answer-query" list.
  - **Signing-Queries:** This oracle is same as Signing-Queries oracle in Game-*I* except that PP_{ID} is provided by the adversary $\mathcal{A}^{II}$.
- **Output-*II*.** Finally, $\mathcal{A}^{II}$ outputs a tuple of $(ID^*, PK_{ID^*}, m^*, \sigma^*)$. Here, ID^* should not be queried to Extract-Private-Key oracle and m^* also should not be queried to Signing-Queries oracle w.r.t. the identity ID^* and the corresponding public key PK^*.

If the output satisfies (6), then $\mathcal{A}^{II}$ is called winning the game. Denote by $\mathrm{Adv}^{\mathrm{EUF}}_{\mathcal{A}^{II},\mathsf{CLS}}(\lambda)$ $\mathcal{A}^{II}$'s advantage, which equals to the probability of $\mathcal{A}^{II}$ winning the game.

If for both PPT adversaries $\mathcal{A}^{I}$ and $\mathcal{A}^{II}$, their advantages $\mathrm{Adv}^{\mathrm{EUF}}_{\mathcal{A}^{I},\mathsf{CLS}}(\lambda)$ and $\mathrm{Adv}^{\mathrm{EUF}}_{\mathcal{A}^{II},\mathsf{CLS}}(\lambda)$ are negligible, then we call the scheme CLS is existentially unforgeable (EUF) under the chosen message attacks. Or shortly, CLS is a secure certificateless signature scheme.

2.5 Certificateless Homomorphic Signature Scheme

In this subsection, we formally introduce the notion of certificateless homomorphic signature (CLHS) scheme and its security model. First, a CLHS scheme CLHS consists of the following eight algorithms.

- `Setup.` Input: the security parameter λ. Output: public parameter *params* and a master key *msk*. Usually, this algorithm is run by KGC. The parameter *params* is publicly and authentically available, and *msk* is only known by KGC.
- `Partial-Private-Key-Extract.` Input: the master key *msk* and a user's identity ID. Output: the partial private key PP_{ID} for ID. Generally, this algorithm is also run by KGC to generate partial private key for any user.
- `Set-Secret-Value.` Input: user's identity ID. Output: a secret value s_{ID}.
- `Set-Private-Key.` Input: user's secret value s_{ID} and its partial private key PP_{ID}. Output: a *full* private key SK_{ID}.
- `Set-Public-Key.` Input: user's secret value s_{ID}. Output: public key PK_{ID}.
Normally, the two algorithms `Set-Private-Key` and `Set-Public-Key` are run by user for itself after running `Set-Secret-Value`.
- `Sign.` Input: a tuple of
$$(ID, PK_{ID}, SK_{ID}, id, \mathbf{v}),$$
where $\mathbf{v}$ is a vector belonging to some file with identifier *id*. Output: a signature σ.
- `Verify.` Input: a tuple of
$$(ID, PK_{ID}, id, \mathbf{v}, \sigma).$$
Output: 1 (accept) or 0 (reject).
- `Combine.` Input: a user's identity ID, a public key PK_{ID}, a file identifier *id* and the tuples
$$(c_1, \mathbf{v}_1, \sigma_1), \cdots, (c_\ell, \mathbf{v}_\ell, \sigma_\ell).$$
Here, the vectors $\mathbf{v}_1, \cdots, \mathbf{v}_\ell$ should belong to the same identifier *id*. Output: a signature σ for the "combined" vector $\mathbf{v} = \sum_{i=1}^{\ell} c_i \mathbf{v}_i$ w.r.t. ID, or a symbol $\perp$, which denotes that the inputs are incorrect and hence incombinable.

Correctness. The correctness requires that, for any
$$msk, PP_{ID}, s_{ID}, SK_{ID}, PK_{ID}$$
generated by respectively running above algorithms, all $id \in \{0,1\}^*$ and any message vector $\mathbf{v}$, the following two conditions hold.

1) $1 \leftarrow \texttt{Verify}(ID, PK_{ID}, id, \mathbf{v}, \texttt{Sign}(ID, PK_{ID}, SK_{ID}, id, \mathbf{v}))$,
2) If for $1 \le i \le \ell$,
$$1 \leftarrow \texttt{Verify}(ID, PK_{ID}, id, \mathbf{v}_i, \sigma_i),$$
then it holds that
$$1 \leftarrow \texttt{Verify}\Big(ID, PK_{ID}, id, \sum c_i \mathbf{v}_i, \texttt{Combine}\big(ID, PK_{ID}, id, (c_j, \mathbf{v}_j, \sigma_j)_{j=1}^{\ell}\big)\Big).$$

Security. Here, we still consider two kinds of adversaries $\mathcal{A}^{I}$ and $\mathcal{A}^{II}$, who respectively denote a usual adversary and a malicious KGC. The security of CLHS is modeled by two games Game-*I* and Game-*II*, which are respectively played by $\mathcal{A}^{I}$ with its challenger CH^{I} and $\mathcal{A}^{II}$ with its challenger CH^{II}.

<u>**Game-*I***</u>: This is a game designed for the adversary $\mathcal{A}^{I}$ and its challenger CH^{I}.

- **Setup-*I*:** The challenger runs $(params, msk) \leftarrow \texttt{Setup}(\lambda)$. Keep *msk* secret and return *params* to $\mathcal{A}^{I}$. In addition, the challenger also initializes an empty "query-answer" list, which will be used to store all the following queries and their answers (from the challenger).
- **Queries-*I*:** $\mathcal{A}^{I}$ is allowed to adaptively make the following queries.
  - **Extract-Partial-Private-Key.** When $\mathcal{A}^{I}$ submits the identity ID and wants to obtain its partial private key, the challenger runs
$$PP_{ID} \leftarrow \texttt{Partial-Private-Key-Extract}(msk, ID), \tag{7}$$
and give it to the adversary. Then store (ID, PP_{ID}) to the "query-answer" list.

- **Extract-Private-Key.** When $\mathcal{A}^I$ submits the identity ID and wants to get the corresponding private key, the challenger first checks if (ID, PP_{ID}, s_{ID}) appears in the "query-answer" list. If it is, recover PP_{ID} and s_{ID}. Otherwise, run (7) to obtain PP_{ID} and generate
$$s_{ID} \leftarrow \texttt{Set-Secret-Value}(ID). \tag{8}$$
Then compute
$$SK_{ID} \leftarrow \texttt{Set-Private-Key}(PP_{ID}, s_{ID}). \tag{9}$$
Return SK_{ID} to $\mathcal{A}^I$, and add $(ID, PP_{ID}, s_{ID}, SK_{ID})$ to the "query-answer" list.
- **Request-Public-Key:** When the adversary queries the public key of the identity ID, CH^I first checks if (ID, s_{ID}) appears in the "query-answer" list. If it is, recover s_{ID}. Else, run (8) to get s_{ID}. Then compute
$$PK_{ID} \leftarrow \texttt{Set-Public-Key}(s_{ID}), \tag{10}$$
and return it to $\mathcal{A}^I$. Add (ID, s_{ID}, PK_{ID}) to the "query-answer" list.
- **Replace-Public-Key.** The adversary is allowed to replace some public key PK_{ID} of ID with another value PK'_{ID} chosen by himself. The challenger adds (ID, PK_{ID}, PK'_{ID}) to the "query-answer" list.
Note that, it is not necessary to provide the secret value of PK'_{ID} when $\mathcal{A}^I$ making this kind of queries.
- **Signing-Queries.** When the adversary queries the signature of a vector $\mathbf{v}$, which belongs to the identifier id, with respect to the identity ID and PK_{ID}, the challenger first finds SK_{ID} from the list of query-answer. If it does not exist, generate it by respectively running (7), (8), and (9) to get PP_{ID}, s_{ID}, and SK_{ID}. Then run
$$\sigma \leftarrow \texttt{Sign}(ID, PK_{ID}, SK_{ID}, id, \mathbf{v}),$$
and return it to $\mathcal{A}^I$.
If PK_{ID} has been replaced by the adversary, then CH^I may not be able to find SK_{ID} and hence the answers for signature queries may not be correct. In this case, we stipulate that the adversary needs to additionally submit s_{ID}, which is corresponding to the replaced PK_{ID}.

- **Challenge-I** : Finally, the adversary submits the tuple of
$$(ID^*, PK_{ID^*}, id^*, \mathbf{v}^*, \sigma^*).$$
Here, ID^*'s private key should not be extracted. Moreover, ID^* can also not be an identity for which the public key has been replaced but its partial private key has been extracted. In addition, $(id^*, \mathbf{v}^*)$ should not be queried for signature with respect to ID^*, PK_{ID^*}.

In this case, we call $\mathcal{A}^I$ wins the above Game-I if
$$1 \leftarrow \texttt{Verify}(ID^*, PK_{ID^*}, id^*, \mathbf{v}^*, \sigma^*), \tag{11}$$
and one of the following holds:

- **Case 1.** The identifier id^* does not equal to any id appeared in the "query-answer" list and $\mathbf{v}^* \neq 0$. (Type 1 Forgery)
- **Case 2.** $id^* = id_0$, where id_0 is some identifier queried by $\mathcal{A}^I$ to the signature oracle, but $\mathbf{v}^*$ does not belong to the subspace V_0 spanned by the vectors queried to signature oracle with the same id_0. (Type 2 Forgery)

Define the advantage $\mathrm{Adv}^{\mathrm{EUF}}_{\mathrm{CLHS},\mathcal{A}^I}(\lambda)$ as the probability of $\mathcal{A}^I$ winning the above game.

Game-II : This is a game played by another adversary $\mathcal{A}^{II}$ and its challenger CH^{II}.

- **Setup-II**: First, CH^{II} runs $(params, msk) \leftarrow \texttt{Setup}(\lambda)$ and gives $(params, msk)$ to $\mathcal{A}$. In addition, the challenger also initializes an empty "query-answer"list.
- **Queries-II**: The adversary $\mathcal{A}^{II}$ is allowed to adaptively make the following queries:
  - **Extract-Partial-Private-Key:** The adversary runs
  $$PP_{ID} \leftarrow \texttt{Partial-Private-Key-Extract}(msk, ID)$$
  to obtain the partial private key for any identity ID. This can be done by the adversary itself since it owns the master key. For consistency, $\mathcal{A}^{II}$ needs to submit PP_{ID} (if necessary) when it making the following queries about ID.
  - **Extract-Private-Key:** When the adversary submits the identity ID and the corresponding partial private key PP_{ID}, the challenger first checks if (ID, s_{ID}) appears in the "query-answer" list. If it is, recover s_{ID}. Else, runs
  $$s_{ID} \leftarrow \texttt{Set-Secret-Value}(ID), \tag{12}$$
  to obtain s_{ID}. Then compute
  $$SK_{ID} \leftarrow \texttt{Set-Private-Key}(PP_{ID}, s_{ID}), \tag{13}$$
  return the value SK_{ID} to the adversary and add $(ID, PP_{ID}, s_{ID}, SK_{ID})$ to the "query-answer" list.
  - **Request-Public-Key:** When $\mathcal{A}^{II}$ submits ID, PP_{ID} and intends to obtain the corresponding public key, the challenger first checks if (ID, s_{ID}) appears in the "query-answer" list. If it is, recover s_{ID}. Otherwise, run (12) to get s_{ID}. Then compute
  $$PK_{ID} \leftarrow \texttt{Set-Public-Key}(s_{ID}). \tag{14}$$
  Give PK_{ID} to $\mathcal{A}^{II}$ and add (ID, s_{ID}, PK_{ID}) to the "query-answer" list.
  - **Signing-Queries:** When the adversary queries the signature of a vector $\mathbf{v}$, which belongs to the identifier id, with respect to the identity ID and PP_{ID}, the challenger first finds SK_{ID} and PK_{ID} from the list of "query-answer". If they do not exist, generate them by running (12), (13) and (14). Then compute
  $$\sigma \leftarrow \texttt{Sign}(ID, PK_{ID}, SK_{ID}, id, \mathbf{v}),$$
  and return it to $\mathcal{A}^{II}$. Finally, store
  $$(ID, PP_{ID}, s_{ID}, SK_{ID}, PK_{ID})$$
  to the "query-answer" list.
- **Challenge-II** : Finally, the adversary outputs a tuple of
$$(ID^*, PK_{ID^*}, id^*, \mathbf{v}^*, \sigma^*).$$
We remark that ID^* and $(id^*, \mathbf{v}^*)$ should not be issued as a Extract-Private-Key query and a signing query (with respect to ID^* and PK_{ID^*}), respectively.

The adversary $\mathcal{A}^{II}$ is called winning Game-II if (6) holds and either Case 1 or Case 2 (in Game-I) holds. Define the advantage $\mathrm{Adv}^{\mathrm{EUF}}_{\mathrm{CLHS},\mathcal{A}^{II}}(\lambda)$ as the probability of $\mathcal{A}^{II}$ winning the above game.

The scheme CLHS is called existentially unforgeable (EUF) under the chosen message attacks if both of the advantages $\mathrm{Adv}^{\mathrm{EUF}}_{\mathrm{CLHS},\mathcal{A}^{I}}(\lambda)$ and $\mathrm{Adv}^{\mathrm{EUF}}_{\mathrm{CLHS},\mathcal{A}^{II}}(\lambda)$ are negligible. Or shortly, CLHS is a secure certificateless homomorphic signature scheme.

3 A Concrete Scheme and Its Security Proof

Next, we give a concrete construction of CLHS scheme CLHS based on a standard certificateless signature scheme CLS, which is described in Section 2.4.

- `Setup`: For the security parameter λ, choose two groups G_1, G_2 with the same order $q \geq 2^\lambda$, and a bilinear map $e: G_1 \times G_1 \to G_2$. Let g be a generator of G_1, randomly choose $x \xleftarrow{\$} \mathbb{Z}_q^*$ and set $h = g^x$. Denote by H_1 and H_2 two hash functions from $\{0,1\}^*$ to G_1. Then run

$$(params', msk') \leftarrow \texttt{CL-Setup}(\lambda).$$

  Finally, output the public parameter

$$params = (q, g, h, e, G_1, G_2, H_1, H_2, params')$$

  and the master key $msk = (msk', x)$.
- `Partial-Private-Key-Extract`: When given the master key $msk = (msk', x)$ and an identity ID for some user, this algorithm computes

$$PP_{ID,1} \leftarrow \texttt{CL-Partial-Private-Key-Extract}(msk', ID)$$

  and

$$PP_{ID,2} \leftarrow H_1(ID)^x.$$

  Output

$$PP_{ID} = (PP_{ID,1}, PP_{ID,2}).$$

- `Set-Secret-Value`: For the input ID, run

$$s_{ID,1} \leftarrow \texttt{CL-Set-Secret-Value}(ID),$$

  and randomly choose

$$s_{ID,2} := y \xleftarrow{\$} \mathbb{Z}_q^*.$$

  Set $s_{ID} = (s_{ID,1}, s_{ID,2})$ and output it.
- `Set-Private-Key`: For the input $s_{ID} = (s_{ID,1}, y)$ and the corresponding $PP_{ID} = (PP_{ID,1}, PP_{ID,2})$, run

$$SK_{ID,1} \leftarrow \texttt{CL-Set-Private-Key}(s_{ID,1}, PP_{ID,1}),$$

  set

$$SK_{ID,2} = (PP_{ID,2})^y = H_1(ID)^{xy},$$

  and output $SK_{ID} = (SK_{ID,1}, SK_{ID,2})$.
- `Set-Public-Key`: For the input $s_{ID} = (s_{ID,1}, y)$, run

$$PK_{ID,1} \leftarrow \texttt{CL-Set-Public-Key}(s_{ID,1}),$$

  and compute $PK_{ID,2} = h^y$. Set $PK_{ID} = (PK_{ID,1}, PK_{ID,2})$ and output it.
- `Sign`: For the input of the tuple $(ID, PK_{ID}, SK_{ID}, id, \mathbf{v})$, where $\mathbf{v} = (v_1, \cdots, v_N) \in \mathbb{Z}_q^N$, this algorithm runs as follows. The signer maintains a list L to record the identifier id and its related information. First, check if id appears in L.
  - If it is not, randomly choose $r \xleftarrow{\$} \mathbb{Z}_q$, let $w = g^r$ and run

$$\sigma_1 \leftarrow \texttt{CL-Sign}(ID, PK_{ID,1}, SK_{ID,1}, (id, w)).$$

    Add $(id, (r, w, \sigma_1))$ into L.
  - Else, retrieve (r, w, σ_1) from the list L.

  Then randomly choose $s \xleftarrow{\$} \mathbb{Z}_q^*$ and compute

$$\sigma_2 = (SK_{ID,2})^{\sum_{j=1}^N v_j} \left(H_1(ID)^s \cdot \prod_{j=1}^N H_2(id, ID, PK_{ID}, j)^{v_j} \right)^r.$$

  Output $Q = (w, \sigma_1, \sigma_2, s)$ as the signature.
- `Verify`: For the input of the tuple $(ID, PK_{ID}, id, \mathbf{v}, Q)$, first parse Q as $(w, \sigma_1, \sigma_2, s)$, PK_{ID} as $(PK_{ID,1}, PK_{ID,2})$. Then run

$$b \leftarrow \texttt{CL-Verify}(ID, PK_{ID,1}, (id, w), \sigma_1).$$

  If $b = 0$, stop and output 0. Else, check if

$$e(\sigma_2, g) = e\left(H_1(ID), PK_{ID,2}\right)^{\sum_{j=1}^N v_j} \cdot e\left(H_1(ID)^s \cdot \prod_{j=1}^N H_2(id, ID, PK_{ID}, j)^{v_j}, w \right).$$

  If it holds, then output 1. Otherwise, output 0.
- `Combine` : For the inputs of ID, PK_{ID}, id and the tuples

$$(c_1, \mathbf{v}_1, Q_1), \cdots, (c_\ell, \mathbf{v}_\ell, Q_\ell),$$

  this algorithm first parses

$$Q_i = (w^{(i)}, \sigma_1^{(i)}, \sigma_2^{(i)}, s^{(i)}),$$

  for $1 \leq i \leq \ell$, and check if $w^{(1)} = \cdots = w^{(\ell)}$.
  - If it isn't, output $\perp$.
  - Else, continue to check if

$$1 \leftarrow \texttt{Verify}(ID, PK_{ID}, id, \mathbf{v}_i, Q_i),$$

    for $1 \leq i \leq \ell$. If one of them does not hold, output $\perp$. Else,

$$\sigma_2 = \prod_{i=1}^{\ell} \left(\sigma_2^{(i)}\right)^{c_i}, \; s = \sum_{i=1}^{\ell} c_i s^{(i)}.$$

  Finally, output $Q = (w^{(1)}, \sigma_1^{(1)}, \sigma_2, s)$ as the signature of "combined" vector $\mathbf{v} = \sum_{i=1}^{\ell} c_i \mathbf{v}_i$.

The correctness of this scheme can be easily verified. About its security, we have the following:

Theorem 1. If the underlying CLS scheme is secure and the CDH assumption in G_1 holds, then the above scheme CLHS is also secure (in the random oracle model).

Proof. Since there are two kinds of adversaries in the security model, we present the proof in the following two parts: Part 1 and Part 2.

Part 1. Let $\mathcal{A}^I$ be an adversary in Game-I against the scheme CLHS. We will construct another adversary $\mathcal{B}^I$ breaking the security of the underlying CLS scheme or the CDH assumption. In particular, $\mathcal{B}^I$ first randomly choose $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, then he guesses $\mathcal{A}^I$ will output a Type 1 forgery and hence attacks on the security of CLS scheme. Otherwise, he guesses $\mathcal{A}^I$ will output a Type 2 forgery and chooses to attack on the CDH assumption.

Now, we introduce the construction of $\mathcal{B}^I$ when $b = 0$. Concretely, given the public parameter $params'$, $\mathcal{B}^I$ generates a bilinear map e on two cyclic groups G_1, G_2 with the same order q. Let g be a generator of G_1. Then randomly choose $x \xleftarrow{\$} \mathbb{Z}_q^*$, set

$$h = g^x, \text{ and } params = (q, g, h, e, G_1, G_2, params'),$$

and return $params$ to $\mathcal{A}^I$. The two hash functions H_1 and H_2 are modeled as random oracles and simulated by $\mathcal{B}^I$. For convenience to simulate, he initializes two empty lists L_{H_1} and L_{H_2}. In addition, he also maintains a "query-answer" list, which will store all the following queries (from $\mathcal{A}^I$) and the corresponding answers. Then answer $\mathcal{A}$'s queries as follows.

- **Hash-Queries.** For the query ID to H_1, $\mathcal{B}^I$ first checks if there exists $(ID, H_1(ID))$ in L_{H_1}. If it is, return $H_1(ID)$ to $\mathcal{A}^I$. Otherwise, randomly choose $H_1(ID) \xleftarrow{\$} G_1$ and return it to $\mathcal{A}^I$. Then add $(ID, H_1(ID))$ to L_{H_1}. The simulation of H_2 oracle is also similar.
- **Extract-Partial-Private-Key.** When $\mathcal{A}^I$ submits ID, $\mathcal{B}^I$ also gives it to his own Extract-Partial-Private-Key oracle and gets the response $PP_{ID,1}$. Then compute $PP_{ID,2} = H_1(ID)^x$ and return
$$PP_{ID} = (PP_{ID,1}, PP_{ID,2}).$$
Add (ID, PP_{ID}) to the "query-answer" list.
- **Extract-Private-Key.** For the query ID to this oracle, $\mathcal{B}^I$ submits it to his own Extract-Private-Key oracle and gets the response $SK_{ID,1}$. Then choose $y \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $SK_{ID,2} = H_1(ID)^{xy}$. Return $SK_{ID} = (SK_{ID,1}, SK_{ID,2})$ to $\mathcal{A}_I$. Finally, add
$$(ID, s_{ID}, SK_{ID}) = (ID, (\perp, y), (SK_{ID,1}, SK_{ID,2}))$$
to the "query-answer" list.
- **Request-Public-Key.** When $\mathcal{A}^I$ requests ID's public key, $\mathcal{B}^I$ also submits it to its own Request-Public-Key oracle and obtains the response $PK_{ID,1}$. Then check if $(ID, s_{ID}) = (ID, (\perp, y))$ appears in the "query-answer" list. If it is, obtain this y. Otherwise, randomly choose $y \xleftarrow{\$} \mathbb{Z}_q^*$. Compute $PK_{ID,2} = h^y$ and return $PK = (PK_{ID,1}, PK_{ID,2})$ to $\mathcal{A}^I$. Finally, store
$$(ID, s_{ID}, PK_{ID}) = (ID, (\perp, y), (PK_{ID,1}, PK_{ID,2}))$$
into the "query-answer" list.
- **Replace-Public-Key.** If $\mathcal{A}^I$ replaces some
$$PK_{ID} = (PK_{ID,1}, PK_{ID,2})$$
with another value
$$PK'_{ID} = (PK'_{ID,1}, PK'_{ID,2}),$$
then $\mathcal{B}^I$ submits $(ID, PK'_{ID,1})$ as his public-key replacement query on $PK_{ID,1}$. Then record (ID, PK_{ID}, PK'_{ID}) into the "query-answer" list.
- **Signing-Queries.** For the signing query $(ID, PK_{ID}, id, \mathbf{v})$, $\mathcal{B}^I$ first checks if (ID, SK_{ID}) appears in the "query-answer" list.
  - If it is not, submit ID to his own Extract-Private-Key oracle and obtain $SK_{ID,1}$. Then choose $y \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $SK_{ID,2} = H_1(ID)^{xy}$. Add
  $$(ID, SK_{ID}) = (ID, (SK_{ID,1}, SK_{ID,2}))$$
  to the "query-answer" list.
  - Else, recover SK_{ID} from the "query-answer" list.

  Then perform the remaining steps in `Sign` using this SK_{ID}. Note that, if the public key PK_{ID} is replaced by another one PK'_{ID}, then $\mathcal{A}^I$ has to submit the corresponding secret value s'_{ID} according to the stipulation of our security model. Now, $\mathcal{B}^I$ can still compute the full secret key SK_{ID} by querying the partial private key $PP_{ID,1}$ and combining it with s'_{ID}. Hence, the simulations for signatures in this case are still correct.

Finally, $\mathcal{A}^I$ submits a tuple of

$$(ID^*, PK_{ID^*}, id^*, \mathbf{v}^*, Q^*),$$

where $PK_{ID^*} = (PK_{ID^*,1}, PK_{ID^*,2})$, $Q^* = (w^*, \sigma_1^*, \sigma_2^*, s^*)$. Then $\mathcal{B}^I$ outputs

$$(ID^*, PK_{ID^*,1}, (id^*, w^*), \sigma_1^*)$$

as his own forgery.

If $\mathcal{A}^I$'s output is a successful Type 1 forgery, then it holds that

$$1 \leftarrow \texttt{CL-Verify}(ID^*, PK_{ID^*,1}, (id^*, w^*), \sigma_1^*),$$

and this id^* does not appear in the "query-answer" list as a signature query. Therefore, $\mathcal{B}^I$'s output is a successful forgery for the CLS scheme.

Next, we introduce the construction of $\mathcal{B}^I$ when $b = 1$. Concretely, $\mathcal{B}^I$ will attack on the CDH assumption on group G_1 by using $\mathcal{A}^I$ as a subroutine. Given the tuple (q, G_1, g, g^a, g^b), $\mathcal{B}^I$ wants to compute and output g^{ab}.

First, choose a bilinear map $e : G_1 \times G_1 \to G_2$, $x \xleftarrow{\$} \mathbb{Z}_q^*$, and set $h = g^x$. Then run

$$(params', msk') \leftarrow \texttt{CL-Setup}(\lambda),$$

and give

$$params = (q, g, h, e, G_1, G_2, params')$$

to $\mathcal{A}^I$. Moreover, he also initializes four lists L_{H_1}, L_{H_2}, L, and "query-answer" lists, which will be used to respectively store $\mathcal{A}^I$'s H_1-query, H_2-query, identifiers in signing-queries, and other query-answer pairs.

- H_1**-Hash Queries.** Without loss of generality, we assume that the identity ID^* (in the Type 2 forgery) has been queried to H_1-oracle when the adversary $\mathcal{A}^I$ outputting the final forgery, and $\mathcal{A}^I$ will (in)directly make q_{H_1} times H_1-oracle queries. Then $\mathcal{B}^I$ randomly choose $\eta \xleftarrow{\$} [q_{H_1}]$, which will be a guess that ID^* was queried to H_1-oracle in the η-th query. For the k-th query ID_k, $\mathcal{B}^I$ answers as follows.
  - If $k \neq \eta$, randomly choose $t_k \xleftarrow{\$} \mathbb{Z}_q^*$, set
  $$H_1(ID_k) = g^{t_k},$$
  and store $(ID_k, H_1(ID_k), t_k)$ to the list L_{H_1}.
  - If $k = \eta$, set $H_1(ID_\eta) = g^b$ and store
  $$\left(ID_\eta, H_1(ID_\eta), \perp\right)$$
  to L_{H_1}.

- **H_2-Hash Queries.** For the input (id, ID, PK_{ID}, j) to H_2-oracle, $\mathcal{B}^I$ randomly chooses $\alpha_j, \beta_j \xleftarrow{\$} \mathbb{Z}_q^*$ and computes
$$H_2(id, ID, PK_{ID}, j) = (g^b)^{\alpha_j} g^{\beta_j} = g^{b\alpha_j + \beta_j}.$$
Return it to $\mathcal{A}^I$ and add
$$\left((id, ID, PK_{ID}, j), g^{b\alpha_j+\beta_j}, \alpha_j, \beta_j\right)$$
to L_{H_2}.
- **Extract-Partial-Private-Key.** If $\mathcal{A}^I$ submits ID to this oracle, $\mathcal{B}^I$ checks if (ID, PP_{ID}) appears in the "query-answer" list. If it is, recover PP_{ID}. Otherwise, run
$$PP_{ID,1} \leftarrow \texttt{CL-Partial-Private-Key-Extract}(msk', ID),$$
and query ID to H_1-oracle. Then compute
$$PP_{ID,2} = H_1(ID)^x,$$
and return $(ID, PP_{ID}) = (ID, (PP_{ID,1}, PP_{ID,2}))$ to the adversary. Finally, store (ID, PP_{ID}) to the "query-answer" list.
- **Extract-Private-Key.** For the queried ID, $\mathcal{B}^I$ obtains PP_{ID} by querying it to the above Extract-Partial-Private-Key oracle. Assume ID is the k-th query to H_1-oracle. If $k = \eta$, stop the simulation. Otherwise, he runs
$$s_{ID,1} \leftarrow \texttt{CL-Set-Secret-Value}(ID)$$
and randomly chooses $s_{ID,2} := y_k \xleftarrow{\$} \mathbb{Z}_q^*$. Compute
$$SK_{ID,1} \leftarrow \texttt{CL-Set-Private-Key}(s_{ID,1}, PP_{ID,1}),$$
and
$$SK_{ID,2} = (PP_{ID,2})^{y_k} = g^{t_k x y_k}.$$
Finally, return $SK = (SK_{ID,1}, SK_{ID,2})$ to $\mathcal{A}^I$ and add
$$(ID, PP_{ID}, s_{ID}, SK_{ID}) = (ID, PP_{ID}, (s_{ID,1}, s_{ID,2}), SK_{ID})$$
to the "query-answer" list.
- **Request-Public-Key.** When $\mathcal{A}^I$ requests the public key of ID, $\mathcal{B}^I$ queries ID to H_1-oracle. Assume ID is the k-th query to H_1-oracle. Then check if (ID, s_{ID}) is in the "query-answer" list. If it is, recover s_{ID}. Otherwise, run
$$s_{ID,1} \leftarrow \texttt{CL-Set-Secret-Value}(ID)$$
and randomly choose $y_k \xleftarrow{\$} \mathbb{Z}_q^*$ for $k \neq \eta$. Compute
$$PK_{ID,1} \leftarrow \texttt{CL-Set-Public-Key}(s_{ID,1}),$$
and
$$PK_{ID,2} = \begin{cases} g^{y_k x}, & k \neq \eta; \\ (g^a)^x, & k = \eta. \end{cases}$$
Finally, return $PK_{ID} = (PK_{ID,1}, PK_{ID,2})$ to $\mathcal{A}^I$ and add
$$(ID, s_{ID}, PK_{ID}) = (ID, (s_{ID,1}, y_k/\perp), PK_{ID})$$
to the "query-answer" list.
- **Replace-Public-Key.** When $\mathcal{A}^I$ replaces PK_{ID} with PK'_{ID}, $\mathcal{B}^I$ updates it in the "query-answer" list.
- **Signing-Queries.** For the signing query $(ID, PK_{ID}, id, \mathbf{v})$, $\mathcal{B}^I$ first obtains the private key SK_{ID} by querying ID to the Extract-Private-Key oracle, which can be done only when $k \neq \eta$. If $k = \eta$, only $SK_{ID,1}$ can be correctly generated but the second part of private key SK_{ID} can not be calculated. Hence, the simulation is divided into the following two cases.
  - $k \neq \eta$. If (ID, PK_{ID}, id) does not appear in the list L, $\mathcal{B}^I$ randomly chooses $r \xleftarrow{\$} \mathbb{Z}_q^*$, set $w = g^r$ and compute
  $$\sigma_1 \leftarrow \texttt{CL-Sign}(ID, PK_{ID,1}, SK_{ID,1}, (id, w)).$$
  Add $(ID, PK_{ID}, id, (r, w, \sigma_1))$ to L. Otherwise, retrieve (r, w, σ_1) from L.
  Next, he chooses $s \xleftarrow{\$} \mathbb{Z}_q^*$, and computes
  $$\sigma_2 = (SK_{ID,2})^{\sum_{j=1}^N v_j} \left(H_1(ID)^s \cdot \prod_{j=1}^N H_2\,(id, ID, PK_{ID}, j)^{v_j}\right)^r$$
  according to the foregoing hash answers for $k \neq \eta$. Then return $Q = (w, \sigma_1, \sigma_2, s)$ to the adversary.
  - $k = \eta$. If (ID, PK_{ID}, id) does not appear in the list L, $\mathcal{B}^I$ randomly chooses $r \xleftarrow{\$} \mathbb{Z}_q^*$, set $w = (g^a)^r = g^{ra}$ and compute
  $$\sigma_1 \leftarrow \texttt{CL-Sign}(ID, PK_{ID,1}, SK_{ID,1}, (id, w)).$$
  Add $(ID, PK_{ID}, id, (r, w, \sigma_1))$ to L. Otherwise, retrieve (r, w, σ_1) from L.
  Then compute
  $$s = -\frac{1}{r} x \sum v_j - \sum \alpha_j v_j, \text{ and } \sigma_2 = w^{\sum \beta_j v_j}.$$
  Return the signature $Q = (w, \sigma_1, \sigma_2, s)$ to $\mathcal{A}^I$.
  It can be verified that Q is a correct signature for $\mathbf{v}$. In fact, we know
  $$1 \leftarrow \texttt{CL-Verify}(ID, PK_{ID,1}, (id, w), \sigma_1),$$
  and
  $$\begin{aligned} &(SK_{ID,2})^{\sum_{j=1}^N v_j} \left(H_1(ID)^s \cdot \prod_{j=1}^N H_2\,(id, ID, PK_{ID}, j)^{v_j}\right)^{ra} \\ &= \left(g^{abx}\right)^{\sum v_j} \left(g^{bs} \cdot \prod g^{(\alpha_j b + \beta_j) v_j}\right)^{ra} \\ &= \left(g^{abx}\right)^{\sum v_j} \left(g^{bs} \cdot g^{b \sum \alpha_j v_j} \cdot g^{\sum \beta_j v_j}\right)^{ra} \\ &= g^{ab(x \sum v_j + rs + r \sum \alpha_j v_j)} \cdot (g^{ra})^{\sum \beta_j v_j} \\ &= w^{\sum \beta_j v_j} \\ &= \sigma_2. \end{aligned}$$

Finally, when $\mathcal{A}^I$ outputs its forgery
$$(ID^*, PK_{ID^*}, id^*, \mathbf{v}^*, Q^*),$$
where $Q^* = (w^*, \sigma_1^*, \sigma_2^*, s^*)$. If $ID^* \neq ID_\eta$, then $\mathcal{B}^I$ outputs $\perp$. Else, he can solve the CDH problem in G_1 from $\mathcal{A}^I$'s Type 2 forgery.[1] In particular, if $\mathcal{A}^I$'s output is a successful Type 2 forgery, then it holds that
$$1 \leftarrow \texttt{CL-Verify}(ID^*, PK_{ID^*,1}, (id^*, w^*), \sigma_1^*),$$
$$e(\sigma_2^*, g) = e\left(H_1(ID^*), PK_{ID^*,2}\right)^{\sum_{j=1}^N v_j^*} \cdot e\left(H_1(ID^*)^s \cdot \prod_{j=1}^N H_2\,(id^*, ID^*, PK_{ID^*}, j)^{v_j^*}, w^*\right), \tag{15}$$

1. The probability of $ID^* = ID_\eta$ equals to $1/q_{H_1}$ since the choosing of η is hidden in $\mathcal{A}^I$'s view.

and id^* equals to some id_0, which is the identifier queried by $\mathcal{A}^I$ to the signature oracle. Let (r, w) be the item stored in L for id_0. Then we can know that $w^* = w = g^{ra}$. (If $w^* \neq w$, then $((id^*, w^*), \sigma_1^*)$ will be a forgery for the underlying CLS scheme `CLS` w.r.t. ID^* and PK_{ID^*}.)

From (15), we know that

$$\begin{aligned} e(\sigma_2^*, g) &= e\left(H_1(ID^*), PK_{ID^*,2}\right)^{\sum_{j=1}^N v_j^*} \cdot \\ & \quad e\left(H_1(ID^*)^s \cdot \prod_{j=1}^N H_2\left(id^*, ID^*, PK_{ID^*}, j\right)^{v_j^*}, w^*\right) \\ &= e\left(g^b, g^{ax}\right)^{\sum v_j^*} \cdot e\left(g^{bs} \cdot \prod (g^{\alpha_j b + \beta_j})^{v_j^*}, g^{ra}\right) \\ &= e\left(g^{abx\sum v_j^*}, g\right) \cdot e\left(g^{abrs + abr\sum \alpha_j v_j^* + ra\sum \beta_j v_j^*}, g\right) \\ &= e\left(g^{ab\left(x\sum v_j^* + rs + r\sum \alpha_j v_j^*\right)} w^{\sum \beta_j v_j^*}, g\right). \end{aligned}$$

According to the non-degenerate property of bilinear map, it holds that

$$\sigma_2^* = g^{ab\left(x\sum v_j^* + rs + r\sum \alpha_j v_j^*\right)} w^{\sum \beta_j v_j^*}.$$

If

$$s \neq -\sum \alpha_j v_j^* - \frac{1}{r} x \sum v_j^*, \qquad (16)$$

or equally,

$$x\sum v_j^* + rs + r\sum \alpha_j v_j^* \neq 0,$$

then we can obtain

$$g^{ab} = \left(\frac{\sigma_2^*}{w^{\sum \beta_j v_j^*}}\right)^{\frac{1}{x\sum v_j^* + rs + r\sum \alpha_j v_j^*}},$$

which solves the CDH problem in G_1. The event

$$s = -\sum \alpha_j v_j^* - \frac{1}{r} x \sum v_j^*$$

occurs with probability $1/q$, which can be bounded using a similar technique as in [7], [20], and hence is omitted here.

Part 2. Let $\mathcal{A}^{II}$ be an adversary who attacks on the scheme `CLHS` in Game-II and "stands for" a malicious KGC. Since the bounding of $\mathcal{A}^{II}$'s advantage is extremely similar to that of $\mathcal{A}^I$, we only present the basic ideas and the details are omitted.

Similarly, we need to construct another adversary $\mathcal{B}^{II}$ to attack on the security of `CLS` or the CDH assumption by using $\mathcal{A}^{II}$ as a subroutine, which is decided by $\mathcal{A}^{II}$'s forgery type. If $\mathcal{A}^{II}$'s output is a Type 1 forgery, then $\mathcal{B}^{II}$ will attack on `CLS` and hence simulate the environments for $\mathcal{A}^{II}$ by using the master key msk' of `CLS` and the oracles he should obtain in Game-II of Section 2.4. Otherwise, he will attack on the CDH assumption in G_1. In this case, we remark that the whole master key $msk = (msk', x)$ is generated by himself and hence he can correctly return msk to $\mathcal{A}^{II}$. After that, he respectively "embeds" g^a, g^b in the Request-Public-Key and the H_1-oracle queries in a same way as that of Part 1. The remaining parts are also routine.

Combining the above Part 1 and Part 2 as well as the conditions in Theorem 1, we know that our proposed scheme `CLHS` is secure. This ends the proof of Theorem 1.

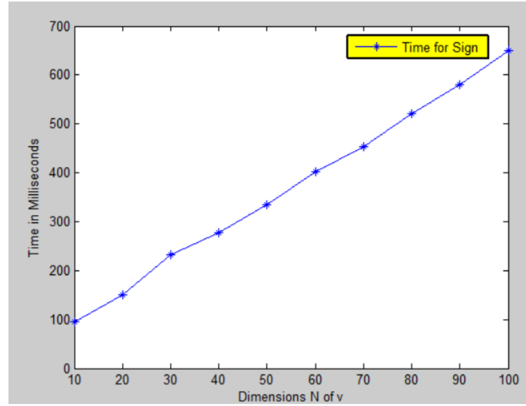

Fig. 5. Time for `Sign` on vector **v** with different dimensions.

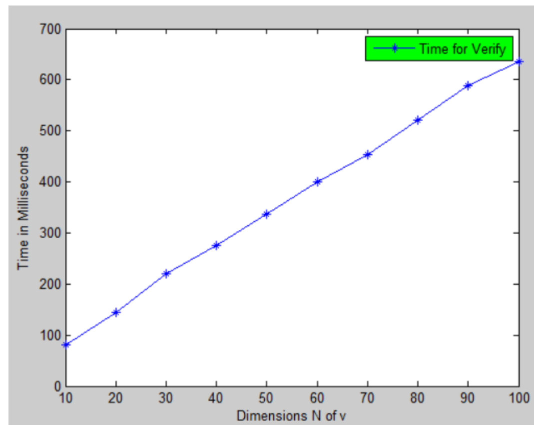

Fig. 6. Time for `Verify` on vector **v** with different dimensions.

4 Performance Analysis

In this section, we consider the efficiency of our proposed scheme in Section 3. In particular, we try to implement it within the framework of "Charm" [2]. Note that a normal certificateless signature scheme is needed, and hence we choose the one suggested by Zhang et al. in [27]. Moreover, we chooses the 512-bit SS elliptic curve from pairing-based cryptography (PBC) library [21] as the basis of whole scheme. All the experiments are run on Intel Core i5-6200U CPU @2.3GHz and 2GB RAM running Ubuntu 14.04 LTS 64-bit and Python 3.4.

Here, we only give the implementations of the algorithms `Partial-Private-Key-Extract`, `Set-Private-Key`, `Sign` and `Verify`, in which the prior two algorithms are irrelevant to the dimension of the signed vector. After choosing 100 different identities IDs, we obtain the average times for extracting partial private key and generating private key respectively equal to 11.38 ms and 9.24 ms.

For the algorithms `Sign` and `Verify`, the dimension of the vector **v** increases from 10 to 100. Each instance is repeated 100 times and an average time is calculated. Then the time consuming for them can be found in Fig. 5 and Fig. 6, respectively.

From the running time of those algorithms, we know that our proposed scheme is practical and hence is suitable for network coding.

5 Conclusions

In this paper, we first give the description of certificateless homomorphic signature scheme and introduce its security model. By revising identity-based homomorphic signature, we naturally obtain a construction of certificateless homomorphic signature. Then we present the detailed security proof for it and its performance analysis.

Acknowledgment

This work is supported in part by National Natural Science Foundation of China (No. 61602061; No. 61672059; No. 61802392; No. 61772520; No. 61772514), and in part by National Key R&D Program of China (No. 2017YFB1400700).

References

[1] R. Ahlswede, N. Cai, S. Li, et al., "Network Information Flow," in "*IEEE Transactions on Information Theory*", vol. 46(4), pp. 1204-1216, 2000.
[2] J. A. Akinyele, C. Garman, I. Miers et al., "Charm: A Framework for Rapidly Prototyping Cryptosystems," in *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111-128, 2013.
[3] S. Al-Riyami, K. Paterson, "Certificateless Public Key Cryptography," in *ASIACRYPT'2003*, pp. 452-473, 2003.
[4] N. Attrapadun, B. Libert, "Homomorphic Network Coding Signatures in the Standard Model," in *PKC*, vol. 6571, pp. 17-34, 2011.
[5] D. Boneh, D. Freeman, "Homomorphic Signatures for Polynomial Functions", in *EUROCRYPT'2011*, pp. 149-168, 2011.
[6] Dan Boneh, D. Freeman, "Linearly Homomorphic Signatures over Binary Fields and New Tools for Lattice-Based Signatures," in *PKC*. Berlin, Germany: Springer, vol 6571, pp. 1-16, 2011.
[7] Dan Boneh, D. Freeman, J. Katz, et al., "Signing a Linear Subspace: Signature: Signature Schemes for Network Coding," in *PKC*. Berlin, Germany: Springer, vol. 5443, pp. 68-87, 2009.
[8] D. Catalano, "Homomorphic Signatures and Message Authentication Codes," in *SCN'2014*, LNCS 8642, pp. 514-519, 2014.
[9] J. Chang, H. Dai, M. Xu, et al.,"Security Analysis of a TELSA-Based Homomorphic MAC Scheme for Authentication in P2P Live Streaming System," in *Security and Communication Networks*, vol. 9(16), pp. 3309-3313, 2016.
[10] J. Chang, Y. Ji, M. Xu, et al., " General Transformations from Single-Genearation to Multi-Generation for Homomorphic Message Authentication Schemes in Network Coding," in *Future Generation Computer Systems*, vol. 91, pp. 416-425, 2019.
[11] J. Chang, H. Ma, A. Zhang, M. Xu, and R. Xue, "RKA Security of Identity-Based Homomorphic Signature Scheme," in *IEEE Access*, vol. 7, pp. 50858-50868, 2019.
[12] W. Chen, H. Lei, K. Qi, "Lattice-Based Linearly Homomorphic Signatures in the Standard Model," in *Theoretical Computer Science*, vol. 634, pp. 47-54, 2016.
[13] M. Choudary Gorantla, A. Saxena, "An Efficient Certificateless Signature Scheme," in *Computational Intelligence and Security,* pp. 110-116, 2015.
[14] X. Huang, Y. Mu, W. Susilo, D. Wong, and W. Wu, "Certificateless Signature Revisited", in *ISC'2007*, pp. 308-322, 2007.
[15] C. Gkantsidis, P. Rodriguez, "Network Coding for Large Scale Content Distriubtion," in *IEEE INFOCOM'2005,* pp. 2235-2245, 2005.
[16] R. Johnson, D. Molnar, D. Song, "Homomorphic Signature Schemes," in *CT-RSA*, vol. 2271, pp. 244-262, 2002.
[17] A. Karati, H. Islam, M. Karuppiah, "Provably Secure and Lightweight Certificateless Signature Scheme for IIoT Environments," in *IEEE Transactions on Industrial Informatics*, to appear, DOI: 10.1109//TII.2018.2794991.
[18] E. Kehdi and B. Li, "Null Keys: Limiting Malicious Attacks via Null Space Properties of Network Coding," in *Proc. IEEE INFOCOM,* 2009, pp. 1224-1232.
[19] Q. Lin, J. Li, Z. Huang, et al., "A Short Linearly Homomorphic Proxy Signature Scheme", in *IEEE ACCESS*, vol. 6, pp. 12966-12972, 2018.
[20] Q. Lin, H. Yan, Z. Huang, et al., "An ID-Based Linearly Homomorphic Signature Scheme and Its Application in Blockchain," in *IEEE ACCESS*, vol. 6, pp. 20632-20639, 2018.
[21] B. Lynn, "The Standard Pairing Based Crypto Library," in `http://crypto.standford.edu/pbc`.
[22] D. Petrovic, K. Ramchandran, J. Rabaey, "Overcoming Untuned Radios in Wireless Networks with Network Coding," in *IEEE Transactions on Information Theory*, vol. 52(6), pp. 2649-2657, 2006.
[23] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," In *Crypto'1984*, vol. 84, pp. 47-53, 1984.
[24] Y. Sun, F. Zhang, "Secure Certificateless Encryption with Short Ciphertext," in *Chinese Journal of Electronics*, vol. 19(2), pp. 313-318, 2010.
[25] F. Wang, Y. Hu, B. Wang, "Lattice-Based Linearly Homomorphic Signature Scheme over Binary Field," in *Science China Information Sciences*, vol. 56(11), pp. 1-9, 2013.
[26] X. Wu, Y. Xu, C. Yuen, et al., "A Tag Encoding Scheme against Pollution Attack to Linear Network Coding," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 25(1), pp. 33-42, 2014.
[27] Z. Zhang, D. Wong, J. Xu and D. Feng, "Certificateless Public-Key Signature: Security Model and Efficient Construction," in *ACNS'2006*, LNCS 3989, pp. 293-308, 2006.
[28] Y. Zhu, B. Li, and J. Guo, "Multicast with Network Coding in Application Layer Overlay Networks," in *IEEE Journal of Selected Areas in Communications on Recent Advances in Service Overlay Networks*, vol. 22(1), pp. 107-120, 2004.