Hindawi
Security and Communication Networks
Volume 2021, Article ID 6699974, 14 pages
https://doi.org/10.1155/2021/6699974

Research Article

Distributed Functional Signature with Function Privacy and Its Application

Muhua Liu, Lin Wang, Qingtao Wu, and Jianqiang Song

Control Science and Engineering Postdoctoral Mobile Station, Henan University of Science and Technology, Luoyang, China

Correspondence should be addressed to Qingtao Wu; wqt8921@haust.edu.cn

Received 27 December 2020; Revised 23 February 2021; Accepted 27 February 2021; Published 12 March 2021

Academic Editor: Liguo Zhang

We introduce a novel notion of distributed functional signature. In such a signature scheme, the signing key for function f will be split into n shares sk_f^i and distributed to different parties. Given a message m and a share sk_f^i, one can compute locally and obtain a pair signature $(f_i(m), \sigma_i)$. When given all of the signature pairs, everyone can recover the actual value $f(m)$ and corresponding signature σ. When the number signature pairs are not enough, nobody can recover the signature $(f(m), \sigma)$. We formalize the notion of function privacy in this new model which is not possible for the standard functional signature and give a construction from standard functional signature and function secret sharing based on one-way function and learning with error assumption. We then consider the problem of hosting services in multiple untrusted clouds, in which the verifiability and program privacy are considered. The verifiability requires that the returned results from the cloud can be checked. The program privacy requires that the evaluation procedure does not reveal the program for the untrusted cloud. We give a verifiable distributed secure cloud service scheme from distributed functional signature and prove the securities which include untrusted cloud security (program privacy and verifiability) and untrusted client security.

1. Introduction

Digital signature schemes were introduced by Diffie and Hellman [1]. In a digital signature scheme, it has a secret signing key and a corresponding verification key. Only the person who has the secret key can sign a message, and any person can verify the signature with the verifiable key. Goldwasser et al. [2] gave a standard secure definition. A signature scheme is unforgeable against chosen message attack if an adversary produces a valid signature of any message with at most negligible probability in probabilistic polynomial time. The adversary is allowed to query signatures for a polynomial number of messages of his choice. Subsequently, it appeared to have many other forms of signatures, such as blind signature [3], group signature [4], ring signature [5], identity-based signature [6], homomorphic signature [7, 8], and so on.

Functional signatures (FSs) were proposed by Boyle et al. [9]. In a functional signature scheme, in addition to a master key which can sign any messages belonging to the message space, it also has a signing key sk_f for any function f, which are derived from the master key. Using the signing key sk_f, one can sign any message in the range of f. The security requires that the probability of producing a valid signature on message m does not exceed a negligible function for any probability polynomial adversary which can query the signing key for functions $f_1, f_2, \ldots, f_\ell$ of his choice and signatures of messages $m_1, m_2, \ldots, m_n$ of his choice. A valid signature on m means that m does not equal to one of the queried messages $m_1, m_2, \ldots, m_n$ or m does not belong to range of one of the queried functions $f_1, f_2, \ldots, f_\ell$. For a functional signature scheme, a desirable property is function privacy which requires a signature should not reveal the function. A typical application is the signed photo-processing software [9]; the owner of photos want to remove red eyes or color scale, but does not allow more significant changes. At the same, the owner wishes the signed photos do not reveal the original image.

Boyle et al. [9] firstly gave a construction based on the existence of one-way functions, which does not satisfy the property of function privacy. Then, they transformed it into a construction that satisfied the function privacy by the noninteractive zero-knowledge arguments of knowledge for NP languages which is based on nonfalsifiable assumptions. In their secure definition of function privacy, it requires that the distribution of signatures on a message m generated via different keys sk_f to be computationally indistinguishable. While this model restricted the application of the scheme, it is not clear how could this weakened notion be applied in the general setting. For example, a software company develops software and delegates the computation ability to a server. The company wishes it does not reveal the core function during the computational procedure and the returned results can be verified. In a functional signature scheme, standard function privacy cannot possibly satisfy this scenario. Since the attacker who has a signing key sk_f can generate the signature $(\sigma, f(m))$ for the message m on the fly, the attacker can obtain the evaluations of $f(m_1), \ldots, f(m_n)$ for the messages $m_1, \ldots, m_n$ of its choices. It will reveal some functions of the software by the evaluation pairs $\{(m_i, f(m_i))\}_{i=1}^{n}$.

Therefore, we try to find a realistic model that allows us to approach function privacy for functional signature based on some standard assumption, such as the existing one-way function and learning with error assumption.

1.1. Our Contributions.

In this paper, we introduce the definition of distributed functional signature scheme. In such a model, the signing key for a function f will be split into n shares sk_f^i and distributed to different parties. Given a secret signing key sk_f^i, anyone can compute locally and obtain a pair signature $(f_i(m), \sigma_i)$. When given n signatures $\{(f_i(m), \sigma_i)\}_{i=1}^{n}$ on the message m, everyone can reconstruct the function evaluation $f(m)$ and the corresponding signature σ. While given less than n signatures $\{(f_i(m), \sigma_i)\}_{i\in S}$ satisfying $|S| < n$, anyone cannot reconstruct the evaluation $f(m)$ and the corresponding signature σ. In our secure definition of function privacy, we require the attacker to not distinguish the distribution of signing key $\{\mathrm{sk}_f^i\}_{i\in S}$ generated via different function f, where $|S| < n$. The attacker is allowed to query key generation oracle and evaluation oracle. This new definition of distributed functional signature generalizes the notion of threshold signature to the setting of functional signature, enables the participants to recover an evaluation of the function f on input m, and verifies the result by the corresponding signature together.

In our new model, it provides a possibility of function privacy in the setting of functional signature. Specifically, when given less than n signing key sk_f^i for function f, the adversary can only know $f_i(m)$ corresponding to the share function f_i which is not enough to determine $f(m)$. We give a construction for distributed functional signature by transforming any functional signature which does not support function privacy into a distributed version which satisfies both unforgeability and function privacy via function secret sharing [10]. Our scheme can be constructed based on the one-way function and learning with error assumption.

We remark here that our notion of function privacy is different from the standard function privacy [9]. The latter mainly considers that single signature does not reveal the function, while we consider that the signature process does not reveal the function by splitting each function into secret key shares. In our scheme, the attacker only gets at most $n-1$ secret key shares and thus obtains at most $n-1$ signatures $\{(f_i(m), \sigma_i)\}_{i\in S}$ satisfying $|S| < n$. Anyone cannot reconstruct the value $f(m)$. Therefore, it satisfies the function privacy.

Hosting services securely in multiple untrusted clouds. In recent years, cloud computing has become a very hot research area in cryptography. Boneh et al. [11] first studied hosting service in the cloud. They considered the security which includes protecting program information and client's inputs against untrusted cloud and protecting program information and authorization procedure against untrusted clients. It has wide applications to protect program information against untrusted client and cloud, such as software protected. Fan and Tang [12] gave a construction from distributed functional encryption. However, it requires that the cloud is semihonest. Sometimes, the cloud may run a fast but incorrect computation due to a financial incentive. Therefore, we also consider another property of verifiability which requires that the returned results from the cloud can be checked in the untrusted cloud model.

In addition, the construction of Fan and Tang [12] is based on the generic functional encryption. As far as we know, most works have approached the problem of constructing generic functional encryption by proposing a candidate under the existing of indistinguishability obfuscation [13]. Meanwhile, the construction of Boneh et al. [11] is also based on the existing indistinguishability obfuscation. However, indistinguishability obfuscation is not a particularly appealing assumption since the security of constructions relies on an exponential number of assumptions [14]. Recently, numerous attacks on multilinear maps [15] have reduced the community's confidence in the security of existing construction of indistinguishability obfuscation. In this work, we provide a construction of secure cloud service scheme based on some more general assumptions, such as the existing one-way function and learning with error assumption.

A secure cloud service scheme satisfying the verifiability can be constructed by a distributed functional signature scheme and a standard signature scheme. The standard signature scheme is used to authenticate the client. In order to achieve the verifiability, the service first generates some signing key $\mathrm{sk}_{\widehat{P}}^i$ for the program $\widehat{P} = x|P(x)$. When the client receives the returned results from clouds, it can get the evaluation (y, σ) by the reconstruction algorithm of distributed signature scheme. To prove that $y = x|P(x)$, the client verifies the signature σ of y, where the signature σ could only be obtained if y is in the range of $\widehat{P}$.

1.2. Related Works. The concept of identity-based signature was proposed by Shamir in [16]. Bellare and Fuchsbauer [17] introduced policy-based signatures, where a signer can only sign messages satisfying some authority-specified policy. Backes et al. [18] introduced delegatable functional signatures which support the delegation of signing capabilities to another party with respect to a functionality. This new notion unifies several signature primitives, such as policy-based signature, functional signature, identity-based signature, and blind signature.

Homomorphic signature was first proposed by Johnson et al. [19], which was initially designed to establish authentication in network coding. In addition, it can also be used to authenticate the stored data. In a homomorphic signature scheme, a holder which has the signature pairs (m_0, σ_0) and (m_1, σ_1) can construct a new signature σ on the value $f(m_0, m_1)$ for some function f without the signing key. Freeman [20] gave a linearly homomorphic signature scheme which allows authentication of linear functions on signed data. Catalano et al. [21] constructed a homomorphic signature scheme for polynomial functions.

The concept of functional encryption comes from the work of Sahai and Waters [22]. The definition of simulation-based security was firstly proposed by Boneh et al. [23]. Garg et al. [13] firstly proposed a construction for functional encryption which supported all polynomial size circuits. Goldwasser et al. [24] constructed a succinct functional encryption scheme based on reusable garbled circuits and identity-based encryption for any polynomial-time function. Subsequently, Goldwasser et al. [25] introduced the notion of multi-input functional encryption, which supported the multi-input functions, and gave a construction based on indistinguishability obfuscation. In a multi-input functional encryption, the decryption key sk_f for a function f takes multiple ciphertexts as input and outputs a function evaluation. Besides, there are many other functional encryptions for inner-product functions [26–28].

Boneh et al. [11] gave a construction of hosting service in the cloud. Their construction relied on indistinguishability obfuscation $i\mathcal{O}$ and restricted the number of colluded clients for the security. Fan and Tang [12] presented a new definition of distributed public key functional encryption. In their scheme, the decryption key for a function f is split into several sharing key sk_i^f. Given a ciphertext that encrypts a message m and a sharing key sk_i^f, it can evaluate a shared value y_i which reveals nothing about the plaintext and the value of $f(m)$. One can recover the value of $f(m)$ by adding all the shared values y_i. With this approach, it can achieve function privacy which is not possible in the setting of regular public key functional encryption. Then, they considered the problem of hosting services in the untrusted cloud. Applying the function private distributed public key functional encryption to the setting of hosting service in multiple clouds, it can remove the restriction that the number of corrupted clients has to be bounded and known in the previous works.

Boyle et al. [9] firstly proposed functional signatures, which allowed an authority to generate signing keys corresponding to various functions such that a user with a signing key sk_f can sign the image of function f on a message m. Okamoto and Takashima [29] firstly introduced the concept of a decentralised multiauthority functional signatures, which supports nonmonotone access structures combined with inner-product relations. Liang and Mitrokotsa [30] generalised the definition of decentralised multiauthority functional signature for more general policy functions. Datta et al. [31] introduced the concept of functional signcryption, which provided the functionalities of both functional encryption and functional signature. Pan et al. [32] introduced the notion of hierarchical functional signcryption which expanded the scope of functional signcryption in hierarchical access-control application. Li et al. [33] introduced the notion of private functional signatures, in which the signing key sk_f can be used to generate a signature $\sigma_{f(x)}$ on the ciphertext c_x, where c_x is a ciphertext for message x.

Blockchains originate from Bitcoin and are essentially a decentralised database that uses peer-to-peer network. It is a new application mode of computer technology such as distributed data storage, point-to-point transmission, and consensus mechanism. This makes blockchains potentially suitable for recording events, medical records, and other management activities. Although it can realize the function of software protection, it does not possess computing capabilities. In this work, our goal is to achieve software protection while ensuring software availability.

Notation. In what follows, we will denote with $\lambda \in \mathcal{N}$ a security parameter. We say $\mathrm{negl}(\lambda)$ is negligible if $|\mathrm{negl}(\lambda)| < (1/\mathrm{poly}(\lambda))$ holds for all polynomials $\mathrm{poly}(\lambda)$ and all sufficiently large λ. Denote $[n]$ as the set $\{1, 2, \ldots, n\}$. We abbreviate probabilistic polynomial time as PPT. Denote $|S|$ as the number of elements in the set S.

2. Preliminaries

In this section, we present definition for various cryptographic primitives that we will use in our construction of distributed functional signature. We assume familiarity with standard signature satisfying unforgeability against adaptively chosen message attack. Below, we first recall the notions of function secret sharing and functional signature.

2.1. Function Secret Sharing. We give the formal definition following the syntax of [10]. An (n, n)–function secret sharing scheme for a function family $\mathcal{F}$ is a tuple of algorithms (FSS.Setup, FSS.ShareGen, FSS.Recon) described as follows:

(i) $\mathrm{FSS.Setup}(1^\lambda, n, \mathcal{F}) \longrightarrow \mathrm{FSS.pp}$: on inputting the security parameter λ, the number of sharers n, and the description of function family $\mathcal{F}$, this algorithm outputs the public parameters FSS.pp.

(ii) $\mathrm{FSS.ShareGen}(\mathrm{FSS.pp}, f) \longrightarrow \{f_i\}_{i=1}^n$: on inputting the parameters FSS.pp and a function $f \in \mathcal{F}$, this algorithm outputs n shares $\{f_i\}_{i=1}^n$ for function f.

(iii) $\text{FSS.Recon}(\text{FSS.pp}, \{f_i(x)\}_{i=1}^{n}) \longrightarrow f_i(x)$: on inputting n values $\{f_i(x)\}_{i=1}^{n}$ which evaluated each function share f_i on x, this algorithm reconstructs all the share values $\{f_i(x)\}_{i=1}^{n}$ and outputs $f(x)$.

Definition 1. Correctness: an (n,n)–function secret sharing scheme for the function family $\mathcal{F}$ is correct, if for any function $f \in \mathcal{F}, x \in \mathcal{D}_f$, $\text{FSS.pp} \longleftarrow \text{FSS.Setup}(1^{\lambda}, n, \mathcal{F})$, and $\{f_i\}_{i=1}^{n} \longleftarrow \text{FSS.ShareGen}(\text{FSS.pp}, f)$, we have

$$\Pr[f(x) = \text{FSS.Recon}(\text{FSS.pp}, \{f_i(x)\}_{i=1}^{n})] \geq 1 - \text{negl}(\lambda). \tag{1}$$

Definition 2. Security: we say an (n,n)–function secret sharing scheme is secure if for any PPT adversary $\mathcal{A}$, the advantage in the following game is negligible:

(i) The challenger generates $\text{FSS.PP} \longleftarrow \text{FSS.Setup}(1^{\lambda}, n, \mathcal{F})$ and sends FSS.pp to the adversary $\mathcal{A}$.
(ii) The adversary outputs $(S, f^0, f^1) \longleftarrow \mathcal{A}(\text{FSS.pp}, \lambda)$, where $f^0, f^1 \in \mathcal{F}$, $\mathcal{D}_{f^0} = \mathcal{D}_{f^1}$, and $S \in [n], |S| = n-1$.
(iii) The challenger randomly chooses a bit $b \longleftarrow \{0,1\}$, computes $\{f_i^b\}_{i=1}^{n} \longleftarrow \text{FSS.ShareGen}(\text{FSS.pp}, f^b)$, and sends $\{f_i^b\}_{i \in S}$ to the adversary $\mathcal{A}$.
(iv) $\mathcal{A}$ outputs a guess $b' \longleftarrow \mathcal{A}(\text{FSS.pp}, \{f_i^b\}_{i \in S})$. The advantage of $\mathcal{A}$ is defined as $\text{Adv} = (\Pr[b' = b] - 1/2)$.

2.2. Functional Signatures. We describe the definition of a functional signature scheme, which is proposed by Boyle et al. in [9]. A functional signature scheme for a message space $\mathcal{M}$ and a function family $\mathcal{F} = \{f: \mathcal{D}_f \longrightarrow \mathcal{M}\}$ consists of four algorithms (FS.Setup, FS.KeyGen, FS.Sign, FS.Very):

(i) $\text{FS.Setup}(1^{\lambda}) \longrightarrow (\text{FS.mvk}, \text{FS.msk})$: on inputting the security parameter, the setup algorithm outputs the master verification key FS.mvk and the master signing key FS.msk.
(ii) $\text{FS.KeyGen}(\text{FS.msk}, f) \longrightarrow \text{sk}_f$: on inputting the master signing key FS.msk and a function $f \in \mathcal{F}$, the key generation algorithm outputs a constrained signing key sk_f, which just signs the message $f(m)$ for any $m \in \mathcal{D}_f$.
(iii) $\text{FS.Sign}(f, \text{sk}_f, m) \longrightarrow (f(m), \sigma)$: on inputting the function f, constrained signing key sk_f, and a message $m \in \mathcal{D}_f$, the signing algorithm outputs a pair signature $(f(m), \sigma)$.
(iv) $\text{FS.Very}(\text{FS.mvk}, m^*, \sigma) \longrightarrow \{0,1\}$: on inputting the verification algorithm FS.mvk, a message m^*, and a signature σ, the verifiable algorithm outputs "1" or "0," where 1 indicates that the signature is valid.

Definition 3. Correctness: for any $f \in \mathcal{F}$ and $m \in \mathcal{D}_f$, $(\text{FS.msk}, \text{FS.mvk}) \longleftarrow \text{FS.Setup}(1^{\lambda})$, $\text{sk}_f \longleftarrow \text{FS.KeyGen}(\text{FS.msk}, f)$, $(m^*, \sigma) \longleftarrow \text{FS.Sign}(f, \text{sk}_f, m)$, it holds that

$$\text{FS.Very}(\text{FS.mvk}, m^*, \sigma) = 1. \tag{2}$$

Definition 4. Unforgeability: the scheme is unforgeable if the successful probability of any PPT algorithm $\mathcal{A}$ in the following game is negligible:

(i) The challenger runs $(\text{FS.mvk}, \text{FS.msk}) \longleftarrow \text{FS.Setup}(1^{\lambda})$ and sends FS.mvk to the adversary $\mathcal{A}$.
(ii) The adversary $\mathcal{A}$ can query a key generation oracle and a signing oracle. The challenger initializes a dictionary indexed by tuples $(f, j) \in \mathcal{F} \times \mathcal{N}$, which contains the signing keys: $\text{sk}_f \longleftarrow \text{FS. KeyGen}(\text{FS.msk}, f)$.
(iii) Key generation oracle: on inputting (f, j), the challenger runs as follows:
  (1) If there is an entry for (f, j) in the dictionary, then output the corresponding key sk_{fj}.
  (2) Otherwise, sample a fresh key $\text{sk}_{fj} \longleftarrow \text{FS.KeyGen}(\text{FS.msk}, f)$, update the dictionary $(f, j) \longrightarrow \text{sk}_{fj}$, and output sk_{fj}.
(iv) Signing oracle: on inputting (f, j, m), the challenger runs as follows:
  (1) If there is an entry for the key (f, j) in the dictionary, then output the signature $\sigma \longleftarrow \text{FS.Sign}(f, \text{sk}_{fj}, m)$.
  (2) Otherwise, sample a fresh key $\text{sk}_{fj} \longleftarrow \text{FS.KeyGen}(\text{FS.msk}, f)$, update the dictionary $(f, j) \longrightarrow \text{sk}_{fj}$, and output the signature $\sigma \longleftarrow \text{FS.Sign}(f, \text{sk}_{fj}, m)$.
(v) The adversary succeeds if it outputs a signature (m^*, σ) such that
  (1) $\text{FS.Very}(\text{FS.mvk}, m^*, \sigma) = 1$.
  (2) There does not exist m such that $m^* = f(m)$ for any f which was sent as a query to the key generation oracle.
  (3) There does not exist a (f, m) such that (f, m) was queried to the signing oracle and $m^* = f(m)$.

Lemma 1 (see [9]). *A functional signature scheme can be constructed based on any one-way function that supports signing keys for any function f which is computed by a polynomial-sized circuit. This scheme satisfies the property of unforgeability, but not function privacy.*

2.3. General Aggregate Signatures. A general aggregate signature scheme [34] consists of four algorithms (AS.KeyGen, AS.Sign, AS.Agg, AS.Very). The key generation algorithm and signing algorithm are the same as a standard digital signature. In the aggregate algorithm, it produce a new signature by $\sigma \longleftarrow \mathrm{Agg}((\mathrm{pk}_1, m_1, \sigma_1), (\mathrm{pk}_n, m_n, \sigma_n))$. Given a sequence of public key, message, and signature triples, anyone can yield an aggregate signature σ. Through $\mathrm{Very}((\mathrm{pk}_1, m_1), \ldots, (\mathrm{pk}_n, m_n), \sigma)$, anyone can verify the correctness of generation aggregate signature.

In addition to satisfying unforgeability under chosen message attack of the standard signature, it needs to satisfy aggregate unforgeability. The security requires that it is computationally infeasible to produce an aggregate forgery for a PPT adversary which can corrupt at most $n-1$ players. The detailed description of security is given as follows.

Definition 5. Aggregate security: a general aggregate signature scheme is aggregate unforgeable, if it holds that

$$\Pr\left[\mathrm{Very}((\mathrm{pk}_1, m_1), \ldots, (\mathrm{pk}_n, m_n), \sigma) = 1\right] < \mathrm{negl}(\lambda), \quad (3)$$

where the probability is over the experiment $(\mathrm{pk}, \mathrm{sk}) \longleftarrow \mathrm{AS.KeyGen}(1^\lambda)$, $((\mathrm{pk}_1, m_1), \ldots, (\mathrm{pk}_n, m_n), \sigma) \longleftarrow \mathscr{A}^{\mathrm{AS.Sign}(\mathrm{sk}, \cdot)}(\mathrm{pk})$.

Lemma 2 (see [34]). *A general aggregate signature scheme can be constructed based on the difficulty of coCDH problem.*

3. Distributed Functional Signature with Function Privacy

In this section, we give a detailed study of distributed functional signature (DFS), n-out-of-n threshold functional signature. In an (n, n) − DFS scheme, during the key generation algorithm, the secret key corresponding to the function is split into n secret key shares $\left\{(\mathrm{sk}_f^i, f_i)\right\}_{i=1}^n$. Then, we can obtain a pair shared signature $(f_i(m), \sigma_i)$ by running partial signature algorithm on the secret key share sk_f^i corresponding to the shared function f_i and a message m. There is also a reconstruction algorithm that outputs a pair signature $(f(m), \sigma)$ on n shared signatures $\left\{(f_i(m), \sigma_i)\right\}_{i=1}^n$.

3.1. Syntax and Security Definition. We present a formal definition of a distributed functional signature, specifying the properties of unforgeability and function privacy. A distributed functional signature scheme for a message space $\mathscr{M}$ and function family $\mathscr{F} = \left\{f \colon \mathscr{D}_f \longrightarrow \mathscr{M}\right\}$ consists of algorithms (DFS.Setup, DFS.KeyGen, DFS.Sign, DFS.Con, DFS.Very).

(i) $\mathrm{DFS.Setup}(1^\lambda, n) \longrightarrow (\mathrm{msk}, \mathrm{mvk})$: on inputting the secure parameter λ and threshold parameter n, the setup algorithm outputs the master verifiable key mvk and the secret key msk.

(ii) $\mathrm{DFS.KeyGen}(\mathrm{msk}, f) \longrightarrow \left\{(\mathrm{sk}_f^i, f_i)\right\}_{i=1}^n$: on inputting the master secret key *msk* and a function $f \in \mathscr{F}$, the key generation algorithm outputs n secret key shares $\left\{(\mathrm{sk}_f^i, f_i)\right\}_{i=1}^n$ for the function f.

(iii) $\mathrm{DFS.Sign}(f_i, \mathrm{sk}_f^i, m) \longrightarrow (f_i(m), \sigma_i)$: on inputting the signing key sk_f^i for a function f_i and an input $m \in \mathscr{D}_f$, the signature algorithm outputs a value $f_i(m)$ and a signature σ_i.

(iv) $\mathrm{DFS.Con}(\mathrm{mvk}, \{(f_i(m), \sigma_i)\}_{i=1}^n) \longrightarrow (f(m), \sigma)$: on inputting the master verifiable key *mvk* and signing pairs $\{(f_i(m), \sigma_i)\}_{i=1}^n$ for the same function f, the reconstruction algorithm outputs a signing pair $(f(m), \sigma)$.

(v) $\mathrm{DFS.Very}(\mathrm{mvk}, m^*, \sigma) \longrightarrow \{0, 1\}$: on inputting the master verifiable key mvk, a message m^*, and a signature σ, the verifiable algorithm outputs "1" or "0," where 1 implies that the signature is valid.

Definition 6. Correctness: an (n, n) − DFS scheme is correct if for any $\mathrm{DFS.Setup}(1^\lambda, n) \longrightarrow (\mathrm{msk}, \mathrm{mvk})$, any $f \in \mathscr{F}$, and any $m \in \mathscr{D}_f$, $\left\{(\mathrm{sk}_f^i, f_i)\right\}_{i=1}^n \longleftarrow \mathrm{DFS.KeyGen}(\mathrm{msk}, f)$, it holds that

$$\Pr\left[f(m) = f'(m)\right] \geq 1 - \mathrm{negl}(\lambda),$$
$$\Pr\left[\mathrm{DFS.Very}(\mathrm{mvk}, f'(m), \sigma) = 1\right] \geq 1 - \mathrm{negl}(\lambda), \quad (4)$$

where $(f_i(m), \sigma_i) \longleftarrow \mathrm{DFS.Sign}(f_i, \mathrm{sk}_f^i, m)$, $\mathrm{DFS.Con}(\mathrm{mvk}, \{(f_i(m), \sigma_i)\}_{i=1}^n) = (f'(m), \sigma)$, and the probability is taken over the coins in algorithms $\mathrm{DFS.Setup}(1^\lambda, n)$ and $\mathrm{DFS.KeyGen}(\mathrm{msk}, f)$.

Definition 7. Unforgeability: the scheme is unforgeable if the successful probability of any PPT adversary $\mathscr{A}$ in the following game is negligible:

(i) The challenger generates $\mathrm{DFS.Setup}(1^\lambda, n) \longrightarrow (\mathrm{msk}, \mathrm{mvk})$ and sends *mvk* to $\mathscr{A}$.

(ii) The adversary is allowed to query a key generation oracle and a signing oracle. The challenger initializes a dictionary indexed by tuples $(f, j) \in \mathscr{F} \times \mathscr{N}$, which contains signing keys $\left\{(\mathrm{sk}_f^i, f_i)\right\}_{i=1}^n \longleftarrow \mathrm{DFS.KeyGen}(\mathrm{msk}, f)$.

(iii) Key generation oracle: on inputting (f, j), the challenger runs as follows:

  (1) If there exists an entry for the key (f, j) in the dictionary, then output the corresponding value $\left\{(\mathrm{sk}_{f^j}^i, f_i^j)\right\}_{i=1}^n$.

  (2) Else, run the key generation algorithm $\left\{(\mathrm{sk}_{f^j}^i, f_i^j)\right\}_{i=1}^n \longleftarrow \mathrm{DFS.KeyGen}(\mathrm{msk}, f^j)$, add an entry $(f, j) \longrightarrow \left\{(f_i^j, \mathrm{sk}_{f^j}^i)\right\}_{i=1}^n$ to the dictionary, and output $\left\{(f_i^j, \mathrm{sk}_{f^j}^i)\right\}_{i=1}^n$.

(iv) Signing oracle: on inputting (f, j, m), the challenger runs as follows:

  (1) If there exists an entry for the key (f, j) in the dictionary, then run $\left\{\mathrm{DFS.Sign}(f_i^j, \mathrm{sk}_{f^j}^i, m) \longrightarrow (f_i^j(m), \sigma_i^j)_{i=1}^n\right\}$ and DFS.Con $(\mathrm{mvk}, \left\{(f_i^j(m), \sigma_i^j)\right\}_{i=1}^n) \longrightarrow (f^j(m), \sigma^j)$ and output $(f^j(m), \sigma^j)$.

  (2) Else, run the key generation algorithm $\left\{(\mathrm{sk}_{f^j}^i, f_i^j)\right\}_{i=1}^n \longleftarrow \mathrm{DFS.KeyGen}(\mathrm{msk}, f^j)$, add

an entry $(f, j) \longrightarrow (\{\mathrm{sk}^i_{f^j}\}^n_{i=1}, f^j_i)$ to the dictionary, and generate a signature σ^j on $f^j(m)$ by the following steps: $\{\mathrm{DFS.Sign}(f^j_i, \mathrm{sk}^i_{f^j}, m) \longrightarrow (f^j_i(m), \sigma^j_i)\}^n_{i=1}$, $\mathrm{DFS.Con}(\mathrm{mvk}, \{(f^j_i(m), \sigma^j_i)\}^n_{i=1}) \longrightarrow (f^j(m), \sigma^j)$.

(v) The adversary $\mathscr{A}$ succeeds if it can produce (m^*, σ^*) such that

(1) $\mathrm{DFS.Very}(\mathrm{mvk}, m^*, \sigma^*) = 1$.
(2) There does not exist m such that $f^j(m) = m^*$ and $f^j_i(m) = m^*$ for any f^j_i which was contained in the dictionary.
(3) There does not exist a (f, m) pair such that (f, m) was queried to the signing oracle and $m^* = f(m)$.

Definition 8. Ind-based function privacy: intuitively, we require that the successful advantage of any PPT adversary $\mathscr{A}$ in the following game is negligible:

(i) Setup: the challenger generates a key pair $(\mathrm{msk}, \mathrm{mvk}) \longleftarrow \mathrm{DFS.Setup}(1^\lambda, n)$ and sends the verifiable key mvk to the adversary $\mathscr{A}$.
(ii) Key query: proceeding adaptively, the adversary $\mathscr{A}$ submits $f^j \in \mathscr{F}$ to the challenger. The challenger computes $\{(\mathrm{sk}^i_{f^j}, f^j_i)\}^n_{i=1} \longleftarrow \mathrm{DFS.KeyGen}(\mathrm{msk}, f^j)$ and sends $\{(\mathrm{sk}^i_{f^j}, f^j_i)\}^n_{i=1}$ to the adversary $\mathscr{A}$.
(iii) Challenge phase: $\mathscr{A}$ sends the challenge function pair (f^0, f^1, S) which is not queried in the key query phase. The challenger computes $\{(\mathrm{sk}^i_{f^b}, f^b_i)\}^n_{i=1} \longleftarrow \mathrm{DFS.KeyGen}(\mathrm{msk}, f^b)$ and sends $\{(\mathrm{sk}^i_{f^b}, f^b_i)\}^n_{i=1}$ to the adversary $\mathscr{A}$, where $b \longleftarrow \{0, 1\}$ and S is chosen randomly from $[n]$ such that $|S| = n - 1$.
(iv) Proceeding adaptively, the adversary can query the key generation oracle and evaluation oracle.

(1) Key query: $\mathscr{A}$ sends a function $f^j \in \mathscr{F}$ with a restriction that $f^j \neq f^0$ and $f^j \neq f^1$. The challenger answers the same as previous key query.
(2) Evaluation query: $\mathscr{A}$ sends an input x_k with a restriction that $f^0(x_k) = f^1(x_k)$. For $i \in ([n]/S)$, challenger returns $(f_i(x_k), \sigma_i) \longleftarrow \mathrm{DFS.Sign}(f^b_i, \mathrm{sk}^i_{f^b}, x_k)$.

(v) Guess: finally, the adversary $\mathscr{A}$ outputs a bit b'. The successful advantage of $\mathscr{A}$ is defined as $\mathrm{Adv} = (\Pr[b' = b] - 1/2)$.

4. The Construction of DFS

We are now ready to describe a construction of distributed functional signature scheme. Informally, our scheme works as follows. In the setup algorithm, the master verifiable key contains a public parameter of function secret sharing scheme and a verifiable key of functional signature. It sets the master signing key of functional signature scheme to the master secret key. The key generation algorithm consists of sharing function algorithm and key generation algorithm of functional signature, which generates n constrained signature keys sk^i_f corresponding to the shadow function f_i. In the signing algorithm, it computes signing pairs $(f_i(m), \sigma_i)$ on inputting a message m. When enough signatures had been collected, anyone can reconstruct the function value and verify its correctness by the verifiable algorithm. We give a detailed description below. Let FSS = (FSS.Setup, FSS.ShareGen, FSS.Recon) be a function secret sharing scheme for function ensemble $\mathscr{F}$ and (FS.Setup, FS.KeyGen, FS.Sign, FS.Very) be a functional signature scheme. The construction of DFS = (DFS.Setup, DFS.KeyGen, DFS.Sign, DFS.Con, DFS.Very) is given as follows.

(i) $\mathrm{DFS.Setup}(1^\lambda, n) \longrightarrow (\mathrm{msk}, \mathrm{mvk})$:

(1) Run the setup algorithm of FS and generate a key pair $(\mathrm{FS.mvk}, \mathrm{FS.msk}) \longleftarrow \mathrm{FS.Setup}(1^\lambda)$.
(2) Run the setup algorithm of FSS and generate a public parameter $\mathrm{FSS.pp} \longleftarrow \mathrm{FS.Setup}(1^\lambda, n, \mathscr{F})$.
(3) Set the master verifiable key to be mvk = (FS.mvk,FSS.pp) and the master secret key to be msk = FS.msk.

(ii) $\mathrm{DFS.KeyGen}(\mathrm{msk}, f) \longrightarrow \{(\mathrm{sk}^i_f, f_i)\}^n_{i=1}$:

(1) Run the function sharing algorithm of FSS and get n sharing functions $\{f_i\}^n_{i=1} \longleftarrow \mathrm{FSS.ShareGen}(\mathrm{FSS.pp}, f)$.
(2) Run the key generation algorithm of FS and generate n constrained signature keys $\mathrm{sk}^i_f \longleftarrow \mathrm{FS.KeyGen}(\mathrm{FS.msk}, f_i)$ for $i = 1, \ldots, n$.
(3) Output the signature pairs $\{(\mathrm{sk}^i_f, f_i)\}^n_{i=1}$.

(iii) $\mathrm{DFS.Sign}(f_i, \mathrm{sk}^i_f, m) \longrightarrow (f_i(m), \sigma_i)$: given the i–th secret key (f_i, sk^i_f), it computes and outputs $(f_i(m), \sigma_i) = \mathrm{FS.Sign}(f_i, \mathrm{sk}^i_f, m)$.
(iv) $\mathrm{DFS.Con}(\mathrm{mvk}, \{(f_i(m), \sigma_i)\}^n_{i=1}) \longrightarrow (f(m), \sigma)$: it computes $f(m) = \mathrm{FSS.Recon}(\mathrm{FSS.pp}, \{f_i(m)\}^n_{i=1})$ and $\sigma = \{(f_i(m), \sigma_i)\}^n_{i=1}$.
(v) $\mathrm{DFS.Very}(\mathrm{mvk}, m^*, \sigma) \longrightarrow \{0, 1\}$:

(1) If $\sigma = \{(f_i(m), \sigma_i)\}^n_{i=1}$, then it runs $\mathrm{FS.Very}(\mathrm{FS.mvk}, f_i(m), \sigma_i)$ for $i = 1, \ldots, n$. If all of the verifiable algorithms output "1" and $m^* = \mathrm{FSS.Recon}(\mathrm{FSS.pp}\{f_i(m)\}^n_{i=1})$, it outputs "1"; else, it outputs "0."
(2) If not, it runs $\mathrm{FS.Very}(\mathrm{FS.mvk}, m^*, \sigma)$ and outputs the corresponding result.

Remark 1. In our verifiable algorithm, it can verify two forms of signature. One is generated by the reconstruction algorithm. The another is generated by the signing algorithm.

Correctness. The correctness of our DFS construction follows from the correctness of FS and FSS. Firstly, the outputs $f_i(m)$ of signature algorithm satisfy $f(m) = \mathrm{FSS.Recon}(\mathrm{FSS.pp}, \{f_i(m)\}^n_{i=1})$ by the reconstruction algorithm of FSS. Secondly, the verifiable algorithm will output “1” if the signature is generated correctly by the FS scheme.

Theorem 1. *Let FS be an unforgeability functional signature scheme; then, our construction of DFS described above is secure (c.f. Definition 7).*

Proof. We now prove that if there exists a PPT adversary $\mathscr{A}$ that can break the unforgeable security of DFS with non-negligible probability, then another adversary $\mathscr{B}$ can be constructed to break the unforgeable security of FS with the same probability.

Let $\mathscr{C}$ be the challenger for the FS scheme. We construct the adversary $\mathscr{B}$ by $\mathscr{A}$ as follows:

(i) $\mathscr{B}$ first honestly runs the secure experiment of FS and gets FS.mvk of functional signature. Then, it generates FSS.pp$\longleftarrow$FSS.Setup$(1^\lambda, n)$. The adversary $\mathscr{B}$ returns mvk = (FS.mvk,FSS.pp) to $\mathscr{A}$.

(ii) $\mathscr{B}$ initializes a dictionary indexed by tuples $(f, j) \in \mathscr{F} \times \mathscr{N}$, which contains signing keys $\{(\mathrm{sk}^i_{f^j}, f^j_i)\}^n_{i=1}$.

(iii) Key generation oracle: on inputting (f, j), $\mathscr{A}$ operates as follows:

(1) If there exists an entry for the key (f, j) in the dictionary, then output the corresponding pair $\{(\mathrm{sk}^i_{f^j}, f^j_i)\}^n_{i=1}$.

(2) Else, compute $\{f^j_i\}^n_{i=1}\longleftarrow$FSS.ShareGen (FSS.pp, f^j). For $i = 1, \dots, n$, $\mathscr{B}$ queries the key generation oracle of FS and gets n secret keys $\{\mathrm{sk}^i_{f^j}\}^n_{i=1}$. Add an entry $(f, j) \longrightarrow \{(\mathrm{sk}^i_{f^j}, f^j_i)\}^n_{i=1}$ to the dictionary and output $\{(\mathrm{sk}^i_{f^j}, f^j_i)\}^n_{i=1}$.

(iv) Signing oracle: on inputting (f, j, m), $\mathscr{A}$ operates as follows:

(1) If there exists an entry for the key (f, j) in the dictionary, then run $\{$DFS.Sign $(f^j_i, \mathrm{sk}^i_{f^j}, m) \longrightarrow (f^j_i(m), \sigma^j_i)^n_{i=1}\}$ and DFS.Con(mvk, $\{(f^j_i(m), \sigma^j_i)\}^n_{i=1}) \longrightarrow (f^j(m), \sigma^j)$ and output $(f^j(m), \sigma^j)$.

(2) Else, compute $\{f^j_i\}^n_{i=1}\longleftarrow$FSS.ShareGen (pp, f^j). For $i = 1, \dots, n$, $\mathscr{B}$ queries the signing oracle $\{(f^j_i, m)\}^n_{i=1}$ of *FS* and gets n signature pairs $\{(f^j_i(m), \sigma_i)\}^n_{i=1}$. Then, it computes DFS.Con(mvk, $\{(f^j_i(m), \sigma^j_i)\}^n_{i=1}) \longrightarrow (f^j(m), \sigma^j)$ and returns $(f^j(m), \sigma^j)$ to $\mathscr{A}$.

(v) The adversary $\mathscr{A}$ produces a forgeable signature (m^*, σ^*). Then, B computes as follows:

(1) If σ^* is the form of $\{(f^j_i(m), \sigma_i)\}^n_{i=1}$, then $\mathscr{B}$ returns $(f^j_1(m), \sigma^j_1)$ to the challenger.

(2) Else, $\mathscr{B}$ returns (m^*, σ^*) to the challenger.

We observe that $\mathscr{B}$ perfectly simulates the unforgeable experiment of DFS. If $\mathscr{A}$ successfully outputs a forgeable signature (m^*, σ^*), it must satisfy the following:

(1) DFS.Very(mvk, $m^*, \sigma^*) = 1$.

(2) There does not exist m such that $f^j(m) = m^*$ and $f^j_i(m) = m^*$ for any f^j_i which was contained in the dictionary.

(3) There does not exist a (f, m) such that (f, m) was a query to the signing oracle and $m^* = f(m)$.

If the forgeable signature (m^*, σ^*) is the first form, then FS.Very(FS.mvk, $f_i(m), \sigma_i) = 1$ for any $i = 1, \dots, n$. Because f_i is not contained in the dictionary and (m^*, σ^*) is not queried to the signature oracle, $(f^j_1(m), \sigma^j_1)$ is a legally forgeable signature for the functional signature scheme. If the forgeable signature (m^*, σ^*) is the second form, then (m^*, σ^*) is a distributive signature and FS.Very(FS.mvk, $m^*, \sigma^*) = 1$. There does not exist m such that $f_i(m) = m^*$ which is contained in the dictionary; it is also a legally forgeable signature. Therefore, if $\mathscr{A}$ can break the unforgeability security of DFS with non-negligible probability, then we can construct another adversary $\mathscr{B}$ which breaks the unforgeability security of FS with the same probability. ☐

Theorem 2. *Assume FSS.Con is an invertible algorithm. FSS.Con is an invertible algorithm if for any $n-1$ inputs $\{f_i(x)\}^{n-1}_{i=1}$ and an output $f(x)$, it can compute the n–th input $f_n(x)$, where the functions $\{f_i(x)\}^{n-1}_{i=1}$ share function of $f(x)$. For example, Fan and Tang [12] gave a function secret sharing in which the reconstruction algorithm is $f(x) = \sum^n_{i=1} f_i(x)$, and $f_n(x) = f(x) - \sum^{n-1}_{i=1} f_i(x)$. Let FSS be a secure function secret sharing scheme (c.f. Definition 2); then, our construction of DFS described above satisfies function privacy (c.f. Definition 8).*

Proof. We now prove that if there exists a PPT adversary $\mathscr{A}$ that can break the function privacy of DFS with non-negligible probability, then another adversary $\mathscr{B}$ can be constructed to break the security of FSS with the same probability.

Let $\mathscr{C}$ be the challenger for the FSS scheme. We construct the adversary $\mathscr{B}$ by $\mathscr{A}$ as follows:

(i) $\mathscr{B}$ first honestly runs the secure experiment of FSS and gets FSS.pp of function secret sharing scheme. Then, it generates (FS.mvk, FS.msk)$\longleftarrow$FS.Setup(1^λ). The adversary $\mathscr{B}$ returns mvk = (FSS.pp, FS.mvk) to $\mathscr{A}$.

(ii) $\mathscr{B}$ initializes a dictionary indexed by tuples $(f, j) \in \mathscr{F} \times \mathscr{N}$, which contains signing keys $\{(\mathrm{sk}^i_{f^j}, f^j_i)\}^n_{i=1}$.

(iii) Key query: on inputting (f, j), $\mathscr{B}$ operates as follows:

(1) If there exists an entry for the key (f, j) in the dictionary, then output $\{(\mathrm{sk}^i_{f^j}, f^j_i)\}^n_{i=1}$.

(2) Else, compute $\{f_i^j\}_{i=1}^n \longleftarrow$ FSS.ShareGen (FSS.pp, f^j). For $i = 1, \ldots, n$, $\mathscr{B}$ computes $\{(\mathrm{sk}_{f^j}^i, f_i^j)\} \longleftarrow$ FS.KeyGen (FS.msk, f_i^j). Add an entry $(f, j) \longrightarrow \{(\mathrm{sk}_{f^j}^i, f_i^j)\}_{i=1}^n$ to the dictionary and output $\{(\mathrm{sk}_{f^j}^i, f_i^j)\}_{i=1}^n$.

(iv) $\mathscr{A}$ sends the challenge function pair (f^0, f^1, S) to the adversary $\mathscr{B}$, where both f^0 and f^1 were not queried in the previous phase, and $D_{f^0} = D_{f^1}, |S| = n - 1$. Then, $\mathscr{B}$ queries the challenger of FS on (f^0, f^1, S) and gets $\{f_i^b\}_{i \in S}$. For $i \in S$, $\mathscr{B}$ computes $\{(\mathrm{sk}_{f^b}^i, f_i^b)\} \longleftarrow$ FS.KeyGen (FS.msk, f_i^b) and returns $\{(\mathrm{sk}_{f^b}^i, f_i^b)\}_{i \in S}$ to the adversary $\mathscr{A}$.

(v) Proceeding adaptively, $\mathscr{A}$ can query the key generation oracle and evaluation oracle.

(1) Key query: $\mathscr{A}$ sends a function $f^j \in \mathscr{F}$ with a restriction that $f^j \neq f^0$ and $f^j \neq f^1$. $\mathscr{B}$ answers the same as previous key query.

(2) Evaluation query: $\mathscr{A}$ sends an input x_k with a restriction that $f^0(x_k) = f^1(x_k)$. For $i \in ([n]/S)$, $\mathscr{B}$ first computes $\{(f_i^b(x_k), \sigma_i)\}_{i \in S}$ and $f^0(x_k)$. Then, $\mathscr{B}$ computes $f_{i \in ([n]/S)}(x_k)$ by the invertible property of FSS.Con and a corresponding signature $\sigma_{i \in ([n]/S)}$ by the master secret key FS.msk of functional signature. For a functional signature scheme, it has a master signing key FS.msk, which can be used to sign any message. At the same time, it also has a constrained key sk_f for the function f, which only signs the message in the range of f. $\mathscr{B}$ returns $(f_{i \in ([n]/S)}(x_k), \sigma_{i \in ([n]/S)})$ to $\mathscr{A}$.

(vi) At last, the adversary $\mathscr{B}$ outputs the guess b' of the adversary $\mathscr{A}$.

Although the generation way of the signature $\sigma_{i \in ([n]/S)}$ is changed, it does not influence the view of $\mathscr{A}$ because this signature can be verified by the verifiable algorithm of FS. Therefore, $\mathscr{B}$ perfectly simulates function private experiment. If $\mathscr{A}$ can distinguish the function f^0 and f^1 with non-negligible advantage, then $\mathscr{B}$ can break the security of FSS with the same advantage.

In the above construction, the size of combined signature depends on the threshold, which will reduce the verification efficiency. In order to improve the efficiency, it can use aggregate signature to compress the final signature size based on the construction of functional signature proposed by Boyle et al. [9]. In their key generation algorithm, it generates the secret key sk_f for the function f by a standard signature scheme. Therefore, we can change the standard signature into an aggregate signature. The generated signatures $\{\sigma_1, \ldots, \sigma_n\}$ of functional signature scheme can be aggregated into a single signature σ. This change does not affect the security of the construction because the security of Boyle's construction is reduced to the unforgeability of a standard signature. On the one hand, it does not use the aggregate property. On inputting an aggregate signature, the verifiable algorithm of functional signature will output one bit "0." On the other hand, if the adversary can produce a forgeable signature for functional signature scheme by the aggregate signature, then it can be used to break the aggregate security. Based on the first construction, we modify the reconstruction algorithm and verifiable algorithm. The specific description is as follows.

(i) DFS.Setup$(1^\lambda, n) \longrightarrow$ (msk, mvk): it runs the same as before.

(ii) DFS.KeyGen(msk, f) $\longrightarrow \{(\mathrm{sk}_f^i, f_i)\}_{i=1}^n$: it runs the same as before.

(iii) DFS.Sign$(f_i, \mathrm{sk}_f^i, m) \longrightarrow (f_i(m), \sigma_i)$: it runs the same as before.

(iv) DFS.Con(mvk, $\{(f_i(m), \sigma_i)\}_{i=1}^n$) $\longrightarrow (f(m), \sigma)$: it computes $f(m) =$ FSS.Recon(FSS.pp$\{f_i(m)\}_{i=1}^n$) and $\sigma' =$ AS.Agg($\{(f_i(m), \sigma_i)\}_{i=1}^n$) and returns $(f(m), \sigma = (\sigma', \{f_i(m)\})_{i=1}^n)$.

(v) DFS.Very(mvk, m^*, σ) $\longrightarrow \{0, 1\}$:

(1) If $\sigma = (\sigma', \{f_i(m)\}_{i=1}^n)$, then it runs AS.Very($\sigma', \{f_i(m)\}_{i=1}^n$). If the verifiable algorithms output '1' and $m^* =$ FSS.Recon (FSS.pp, $\{f_i(m)\}_{i=1}^n$), it outputs "1"; otherwise, it outputs "0."

(2) If not, it runs FS.Very(FS.mvk, m^*, σ) and outputs the corresponding result.

The secure proofs of the modified construction are similar to the Theorems 1 and 2. In the proof of unforgeability, there are two forms of signature. The first one is $(m^*, \sigma = (\sigma', \{f_i(m^*)\}_{i=1}^n))$ which satisfies AS.Very(σ', $\{f_i(m^*)\}_{i=1}^n) = 1$. Because the reconstruction algorithm is run honestly, there exists an index $i \in \{1, \ldots, n\}$ such that FS.Very(FS.mvk, $f_i(m^*), \sigma_i^*) = 1$. Meanwhile, it satisfies that f_i is not contained in the dictionary, and (m^*, σ^*) is not queried to the signature oracle. So, $(f_i(m^*), \sigma_i^*)$ is a legally forgeable signature for the functional signature scheme. The second form of signature is the same as Theorem 1, which can also be reduced to the security of functional signature. The proof of function privacy is exactly the same as Theorem 2. □

5. Hosting Services Securely in Multiple Untrusted Clouds

We consider the setting of hosting service in untrusted clouds which was first presented by Boneh [11]. In this model, it has three parties: service provider, who owns a program P, cloud server, who provides the computation capability, and arbitrary many clients. The service provider wants to host the program P on a cloud server and authenticate the clients who pay for the service provided by program P. Only the authenticated clients can access the program hosted on the cloud server and compute output on inputs of their choice. Meanwhile, the program P may contain proprietary information, and it needs to protect its privacy. Moreover, the scheme should satisfy some properties [11]:

(i) Weak client: the total cost of the client should only depend on the size of input and security parameter and should be independent of the running time of program P.

(ii) Delegation: the service provider only needs to run one-time setup of the whole system and authentication clients. The total cost should be bounded by a fixed polynomial in the program size in the setup phase and should only depend on the security parameter in the authentication phase.

(iii) Polynomial slowdown: the running time of the cloud is bounded by a fixed polynomial in the running time of the program P.

In our verifiable distributed secure cloud service scheme, the service provider generates a set of encoded program shares for program P and then hosts each encoded program share on one cloud server. Any authenticated client can access the encoded program shares hosted in multiple cloud servers and compute output on inputs of his choice. Meanwhile, we require the client to verify the correctness of returned result from the cloud servers.

5.1. Syntax and Security Definitions. We recall the notion of secure cloud service scheme proposed by Boneh et al. [11] and make some changes based on [12]. The verifiable distributed secure cloud service scheme consists of five algorithms VDS = (VDS.Prog, VDS.Auth, VDS.Inp, VDS. Eval, VDS.Very) which is described as follows:

(i) $\text{VDS.Prog}(1^\lambda, n, P) \longrightarrow \{(\{P_i\}_{i=1}^n, \text{sk})\}$: on inputting the security parameter λ, the threshold parameter n, and a program P, the program generation algorithm outputs encoded programs $\{P_i\}_{i=1}^n$ and a secret key sk which is used in the authentication algorithm.

(ii) $\text{VDS.Auth}(\text{sk}, \text{id}) \longrightarrow \text{token}_{\text{id}}$: on inputting an identity id of a client and the secret key sk, the authentication algorithm outputs a token token_{id} for the client.

(iii) $\text{VDS.Inp}(\text{token}_{\text{id}}, x) \longrightarrow (\hat{x}, \alpha)$: on inputting the token token_{id} and an input x, this algorithm outputs an encoded input $\hat{x}$ and α which is used by client to verify the evaluated value.

(iv) $\text{VDS.Eval}(P_i, \hat{x}) \longrightarrow \hat{y}_i$: on inputting the encoded program P_i and input $\hat{x}$, the evaluation algorithm outputs a result $\hat{y}_i = P_i(\hat{x})$.

(v) $\text{VDS.Very}(\alpha, \{\hat{y}\}_{i=1}^n) \longrightarrow P(x) \text{ or } \perp$: on inputting the verifiable information α and the evaluated value $\hat{y}_i$, the verifiable algorithm outputs the value $P(m)$ or $\perp$, which implies that the client rejects the evaluated result.

The procedure runs as follows: the server provider first runs the algorithm $\text{VDS.Prog}(1^\lambda, n, P)$ and obtains the distributed encoded program $\{P_i\}_{i=1}^n$ and a secret key sk. Then, it sends P_i to the ith cloud server. If a client wants to access the program hosted on the cloud server, the service provider would authenticate the client by the secret key sk. An authenticated client with identity id can encode its inputs by the algorithm $\text{VDS.Inp}(\text{token}_{\text{id}}, x)$ and send $\hat{x}$ to each cloud servers, respectively. Every cloud server will evaluate the sharing program P_i on encoded input $\hat{x}$ and return the evaluation $P_i(\hat{x})$. Finally, the client can reconstruct the value $P(x)$ by the algorithm $\text{VDS.Very}(\alpha, \{\hat{y}\}_{i=1}^n)$ and verify its correctness. In Figure 1, we give the algorithm flowchart for the verifiable distributed secure cloud service scheme.

Two cases for security definition are considered in [11], untrusted cloud security and untrusted client security. We consider the case of untrusted cloud security into two subcases, program privacy and verifiability.

Definition 9 (untrusted cloud security: program privacy) (see [12], Definition 4.1). A VDS scheme is program privacy in untrusted cloud security if the successful advantage of any PPT adversary $\mathscr{A}$ in the following experiment is negligible:

(i) The adversary $\mathscr{A}$ sends challenge program (P_0, P_1, S) to challenger such that $S \subset [n]$, and $|S| = n - 1$. The challenger samples a random bit $b \longleftarrow \{0, 1\}$, computes the encoded program $(\{P_i^b\}_{i=1}^n, \text{sk}) \longleftarrow \text{VDS.Prog}(1^\lambda, n, P_b)$, and sends $\{P_i^b\}_{i \in S}$ to adversary $\mathscr{A}$.

(ii) Proceeding adaptively, the adversary $\mathscr{A}$ can make the authentication query and input query:

(1) Authentication query: the challenger initializes a dictionary indexed by $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$. When $\mathscr{A}$ queries an identity id_j, the challenger returns $\text{token}_{\text{id}_j}$ to adversary $\mathscr{A}$ if there exists a (id, j) in the dictionary. If not, the challenger returns $\text{token}_{\text{id}_j} \longleftarrow \text{VDS.Auth}(\text{sk}, \text{id})$ and updates the dictionary $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$.

(2) Input query: A sends (id, j, x) to the challenger. If there exists $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$ in the dictionary, challenger returns $(\hat{x}, \alpha) \longleftarrow \text{VDS.Inp}(\text{token}_{\text{id}_j}, x)$. If not, challenger computes $\text{token}_{\text{id}_j} \longleftarrow \text{VDS.Auth}(\text{sk}, \text{id}_j)$, returns $(\hat{x}, \alpha) \longleftarrow \text{VDS.Inp}(\text{token}_{\text{id}_j}, x)$, and updates the dictionary $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$.

(iii) Finally, the adversary $\mathscr{A}$ outputs his guess b' for the bit b. The adversary $\mathscr{A}$ succeeds if $b' = b$. The advantage of $\mathscr{A}$'s success is defined as $(\Pr[b' = b] - 1/2)$.

Definition 10 (untrusted cloud security: verifiability). A VDS scheme is verifiable in untrusted cloud security if the successful probability of any PPT adversary $\mathscr{A}$ in the following experiment is negligible:

(i) The adversary $\mathscr{A}$ sends challenge program P to challenger. The challenger samples the encoded

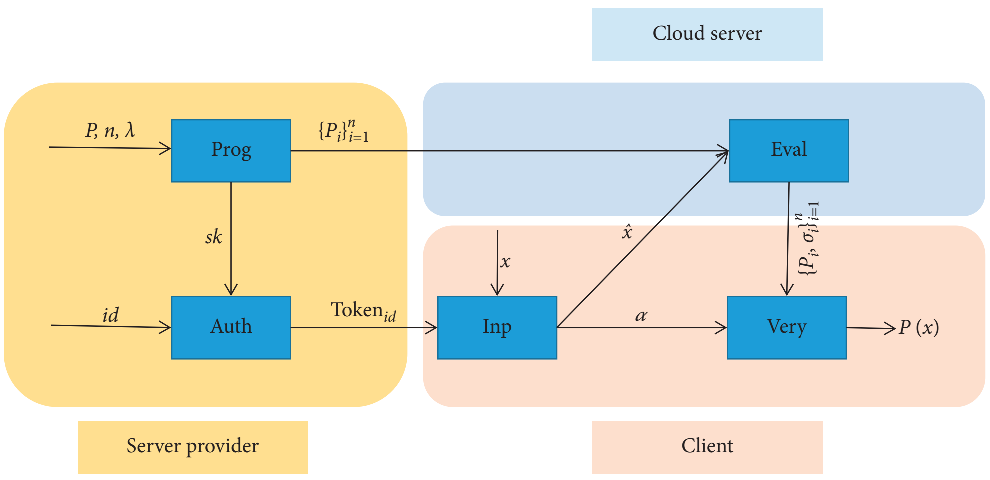

Figure 1: Algorithm flowchart for VDS.

program $(\{P_i\}_{i=1}^n, \mathrm{sk}) \longleftarrow \mathrm{VDS.Prog}(1^\lambda, n, P)$ and sends $\{P_i\}_{i=1}^n$ to adversary $\mathscr{A}$.

(ii) Proceeding adaptively, the adversary $\mathscr{A}$ can make the authentication query. The challenger initializes a dictionary indexed by $(\mathrm{id}, j) \longrightarrow \mathrm{token}_{\mathrm{id}_j}$. When $\mathscr{A}$ queries an identity id_j, the challenger returns $\mathrm{token}_{\mathrm{id}_j}$ to adversary $\mathscr{A}$ if there exists a (id, j) in the dictionary. If not, the challenger returns $\mathrm{token}_{\mathrm{id}_j} \longleftarrow \mathrm{VDS.Auth}(\mathrm{sk}, \mathrm{id})$ and updates the dictionary $(\mathrm{id}, j) \longrightarrow \mathrm{token}_{\mathrm{id}_j}$.

(iii) Finally, $\mathscr{A}$ outputs $(x, \alpha, \{\widehat{y}\}_{i=1}^n)$. $\mathscr{A}$ succeeds if $P(x) \neq \mathrm{VDS.Very}(\alpha, \{\widehat{y}\}_{i=1}^n)$.

For untrusted client security, we require that a collection of corrupt clients with the help of a subset of corrupt servers does not distinguish which program is encoded.

Definition 11 (untrusted client security) (see [12], Definition 4.4). The VDS scheme is untrusted client security if the successfully advantage of any PPT adversary $\mathscr{A}$ in the following experiment is negligible:

(i) The adversary $\mathscr{A}$ sends challenge program (P_0, P_1, S) to challenger such that $S \subset [n]$, $|S| = n - 1$. The challenger samples a random bit $b \longleftarrow \{0,1\}$, computes the encoded program $(\{P_i^b\}_{i=1}^n, \mathrm{sk}) \longleftarrow \mathrm{VDS.Prog}(1^\lambda, n, P_b)$, and sends $\{P_i^b\}_{i\in S}$ to adversary $\mathscr{A}$.

(ii) Proceeding adaptively, the adversary $\mathscr{A}$ can make the authentication query and evaluation query:

(1) Authentication query: the challenger initializes a dictionary indexed by $(\mathrm{id}, j) \longrightarrow \mathrm{token}_{\mathrm{id}_j}$. When $\mathscr{A}$ queries an identity id_j, the challenger returns $\mathrm{token}_{\mathrm{id}_j}$ to adversary $\mathscr{A}$ if there exists a (id, j) in the dictionary. If not, the challenger returns $\mathrm{token}_{\mathrm{id}_j} \longleftarrow \mathrm{VDS.Auth}(\mathrm{sk}, \mathrm{id})$ and updates the dictionary $(\mathrm{id}, j) \longrightarrow \mathrm{token}_{\mathrm{id}_j}$.

(2) Evaluation query: A sends an encoding $\widehat{x}_k$ for the input x_k to the challenger such that $P_0(x_k) = P_1(x_k)$. For $i \in ([n]/S)$, challenger returns $\widehat{y}_{i,k} = P_i^b(\widehat{x}_k)$.

(iii) Finally, the adversary $\mathscr{A}$ outputs his guess b' for the bit b. The adversary $\mathscr{A}$ succeeds if $b' = b$. The successful advantage of $\mathscr{A}$ is defined as $(\Pr[b' = b] - 1/2)$.

5.2. *Our VDS Construction.*

Let $\mathrm{VDS} = (\mathrm{VDS.Prog}, \mathrm{VDS.Auth}, \mathrm{VDS.Inp}, \mathrm{VDS.Eval}, \mathrm{VDS.Very})$ be a distributed functional signature scheme and $\mathrm{Sig} = (\mathrm{Sig.Setup}, \mathrm{Sig.Sign}, \mathrm{Sig.Very})$ be an existential unforgeable signature scheme. Now, we describe our construction as follows:

(i) $\mathrm{VDS.Prog}(1^\lambda, n, P) \longrightarrow (\{P_i\}_{i=1}^n, \mathrm{sk})$:

(1) Run $(\mathrm{msk}, \mathrm{mvk}) \longleftarrow \mathrm{DFS.Setup}(1^\lambda, n)$ and $(\mathrm{Sig.sk}, \mathrm{Sig.vk}) \longleftarrow \mathrm{Sig.Setup}(1^\lambda)$.

(2) Construct a new program $\widehat{P}$, which is given in Figure 2.

(3) Compute $\left\{(\mathrm{sk}_{\widehat{P}}^i, \widehat{P}_i)\right\}_{i=1}^n \longleftarrow \mathrm{DFS.KeyGen}(\mathrm{msk}, \widehat{P})$.

(4) Set $P_i = (\mathrm{sk}_{\widehat{P}}^i, \widehat{P}_i)$ and $\mathrm{sk} = (\mathrm{Sig.sk}, \mathrm{mvk})$.

(ii) $\mathrm{VDS.Auth}(\mathrm{sk}, \mathrm{id}) \longrightarrow \mathrm{token}_{\mathrm{id}}$: parse $\mathrm{sk} = (\mathrm{Sig.sk}, \mathrm{mvk})$, compute the signature $\sigma_{\mathrm{id}} = \mathrm{Sig.Sign}(\mathrm{Sig.sk}, \mathrm{id})$, and output $\mathrm{token}_{\mathrm{id}} = (\mathrm{mvk}, \sigma_{\mathrm{id}}, \mathrm{id})$.

(iii) $\mathrm{VDS.Inp}(\mathrm{token}_{\mathrm{id}}, x) \longrightarrow (\widehat{x}, \alpha)$: parse $\mathrm{token}_{\mathrm{id}} = (\mathrm{mvk}, \sigma_{\mathrm{id}}, \mathrm{id})$ and set $\widehat{x} = (x, \sigma_{\mathrm{id}}, \mathrm{id})$ and $\alpha = (x, \mathrm{mvk})$.

(iv) $\mathrm{VDS.Eval}(P_i, \widehat{x}) \longrightarrow \widehat{y}_i$: parse $P_i = (\mathrm{sk}_{\widehat{P}}^i, \widehat{P}_i)$ and compute $\widehat{y}_i = (\widehat{P}_i(x), \sigma_i) \longleftarrow \mathrm{DFS.Sign}(\widehat{P}_i, \mathrm{sk}_{\widehat{P}}^i, \widehat{x})$.

(v) $\mathrm{VDS.Very}(\alpha, \{\widehat{y}_i\}_{i=1}^n) \longrightarrow P(x)$ or $\perp$: parse $\alpha = (x, \mathrm{mvk})$. Compute $\mathrm{DFS.Con}(\mathrm{mvk}, \{(\widehat{P}_i(x), \sigma_i)\}_{i=1}^n) = (\widehat{P}(x), \sigma)$. Parse $\widehat{P}(x) = x' | P'(x)$. If

```
Input: signature $\sigma$, input $x$ and identity $id$.
Constants: sig.$vk$ and program $P$.

(a) If sig.very (sig.$vk$, $\sigma$, $id$) = 0, output $\perp$.
(b) Else, compute and output $x|P(x)$.
```

Figure 2: Program $\widehat{P}$.

$\text{DFS.Very}(\text{mvk}, \widehat{P}(x), \sigma) = 1$ and $x' = x$, output $P'(x)$. Else, it rejects the returned results.

Correctness. The correctness of our VDS construction follows from the correctness of underlying distributed functional signature scheme DFS and the signature scheme Sig. We can observe that it outputs $\perp$ for an invalid signature in the encoded program $\widehat{P}_i$. Otherwise, it outputs $\widehat{y}_i = (\widehat{P}_i(x), \sigma_i)$. By the correctness of DFS, the output of $\text{VDS.Very}(\alpha, \{\widehat{y}_{i=1}^n\})$ is $P(x)$.

Efficiency. We do not consider the time cost of algorithm VDS.Prog because the cost of encoding the program can be amortized over many input computations. For each invocation of VDS.Auth, the service provider generates a signature for the client id. Therefore, it is efficient for the service provider. The work of the client in handling his inputs only depends polynomially on the size of inputs and a security parameter. For the verifiable algorithm, the client needs to reconstruct the function evaluation and verify a signature, which is independent of the complexity of the program. In the phase of reconstructing the function evaluation, the client just needs to compute a summation operation by using the function secret sharing proposed by Fan and Tang [12].

Security. We show that our DFS construction satisfies untrusted cloud security (program privacy and verifiability) and untrusted client security defined as above.

Theorem 3. *Let DFS be a distributed functional signature scheme against the security of function privacy (c.f. Definition 8); then, our construction of VDS described above has program privacy in untrusted cloud security (c.f. Definition 9).*

Proof. We prove that if there exists a PPT adversary $\mathscr{A}$ that can break the program privacy of VDS with non-negligible probability, then another adversary $\mathscr{B}$ can be constructed to break the function privacy of DFS with the same probability.

Let $\mathscr{C}$ be the challenger for the DFS scheme. We construct the adversary $\mathscr{B}$ by $\mathscr{A}$ as follows:

(i) The adversary $\mathscr{A}$ sends challenge program (P_0, P_1, S) to $\mathscr{B}$ such that $S \subset [n], |S| = n - 1$. $\mathscr{A}$ samples a signing key $(\text{Sig.vk}, \text{Sig.sk}) \longleftarrow \text{Sig.Setup}(1^\lambda)$, constructs two new programs $(\widehat{P}_0, \widehat{P}_1)$ as defined in Figure 2, and sends $(\widehat{P}_0, \widehat{P}_1, S)$ to the challenger of DFS. The challenger returns $(\left\{(\text{sk}^i_{\widehat{P}^b}, \widehat{P}^b_i)\right\}_{i\in S}, \text{mvk})$. $\mathscr{B}$ returns $\left\{(\text{sk}^i_{\widehat{P}^b}, \widehat{P}^b_i)\right\}_{i\in S}$ to $\mathscr{A}$ and sets $\text{sk} = (\text{Sig.sk}, \text{mvk})$.

(ii) Proceeding adaptively, the adversary $\mathscr{A}$ can make the authentication query and input query:

(1) Authentication query: $\mathscr{B}$ initializes a dictionary indexed by $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$. When $\mathscr{A}$ queries an identity id_j, $\mathscr{B}$ returns $\text{token}_{\text{id}_j}$ to $\mathscr{A}$ if there exists a (id, j) in the dictionary. If not, $\mathscr{B}$ computes $\sigma_{\text{id}_j} \longleftarrow \text{Sig.Sign}(\text{Sig.sk}, \text{id}_j)$, then returns $\text{token}_{\text{id}_j} = (\text{mvk}, \sigma_{\text{id}_j}, \text{id})$, and updates the dictionary $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$.

(2) Input query: $\mathscr{B}$ sends (id, j, x) to the challenger. If there exists $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$ in the dictionary, $\mathscr{A}$ returns $\widehat{x} = (x, \sigma_{\text{id}}, \text{id}), \alpha = (x, \text{mvk})$ to adversary. Else, $\mathscr{A}$ computes $\text{token}_{\text{id}_j} \longleftarrow \text{VDS.Auth}(\text{sk}, \text{id})$, returns $(\widehat{x}, \alpha) \longleftarrow \text{VDS.Inp}(\text{token}_{\text{id}_j}, x)$, and updates the dictionary $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$.

(iii) Finally, the adversary $\mathscr{A}$ outputs the guess b' of $\mathscr{B}$.

We observe that $\mathscr{B}$ perfectly simulates program private experiment of VDS. Therefore, if $\mathscr{B}$ could distinguish the program P_0 and P_1 with non-negligible advantage in untrusted cloud security, then $\mathscr{A}$ can break the security of DFS with the same advantage. □

Theorem 4. *Let DFS be a distributed functional signature scheme against the security of unforgeability (c.f. Definition 7); then, our construction of VDS described above is verifiable in untrusted cloud security (c.f. Definition 10).*

Proof. We prove that if there exists a PPT adversary $\mathscr{A}$ that can break verifiability of VDS with non-negligible probability, then another adversary $\mathscr{B}$ can be constructed to break unforgeability of DFS with the same probability.

Let $\mathscr{C}$ be the challenger for the DFS scheme. We construct the adversary $\mathscr{B}$ by $\mathscr{A}$ as follows:

(i) $\mathscr{B}$ honestly runs the unforgeable experiment of DFS and gets mvk of distributed functional signature. Then, $\mathscr{B}$ samples a signing pair $(\text{Sig.sk}, \text{Sig.vk}) \longleftarrow \text{Sig.Sign}(1^\lambda)$.

(ii) $\mathscr{A}$ sends challenge program P to $\mathscr{B}$. $\mathscr{B}$ defines a new function $\widehat{P}$ which is described in Figure 2 and sends $\widehat{P}$ to the challenger $\mathscr{C}$ of DFS. The challenger returns $\left\{(\widehat{P}_i, \text{sk}^i_{\widehat{P}})\right\}_{i=1}^n$, sets $P_i = (\widehat{P}_i, \text{sk}^i_{\widehat{P}})$, and sends $\{P_i\}_{i=1}^n$ to adversary $\mathscr{A}$.

(iii) Proceeding adaptively, $\mathscr{A}$ can make the authentication query. $\mathscr{B}$ initializes a dictionary indexed by $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$. When $\mathscr{A}$ queries an identity id_j, the challenger returns $\text{token}_{\text{id}_j}$ to adversary $\mathscr{A}$ if there exists a (id, j) in the dictionary. If not, the challenger returns $\text{token}_{\text{id}_j} = (\sigma_{\text{id}_j}, \text{id}_j, \text{mvk})$ and updates the dictionary $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$, where $\sigma_{\text{id}} = \text{Sig.Sign}(\text{Sig.sk}, \text{id}_j)$.

(iv) Finally, $\mathscr{A}$ outputs $(x, \alpha, \left\{\widehat{y}_i = (\widehat{P}_i(x), \sigma_i)\right\}_{i=1}^n)$. Then, $\mathscr{B}$ computes $(\widehat{P}(x), \sigma) \longleftarrow \text{VDS.Very}(\alpha, \{\widehat{y}\}_{i=1}^n)$ and outputs the forgeable signature $(x'|P'(x), \sigma)$, where $\widehat{P}(x) = x'|P'(x)$.

Assume that $\mathscr{A}$ succeeded in the experiment of verifiability; then, it must satisfy that $\text{DFS.Very}(\text{mvk}, \widehat{P}(x), \sigma) = 1$, $x' = x$, and $P'(x) \neq P(x)$. For $i = 1, \ldots, n$, $\text{DFS.Very}(\text{mvk}, \widehat{P}_i(x), \sigma_i) = 1$; then, $x|P_i(x)$ must be contained in the range of program $\widehat{P}_i$. Because the adversary $\mathscr{A}$ can just sign the message belonging to the range of encoded programs $\widehat{P}_i$, the evaluation $x|P_i(x)$ must be computed correctly. In addition, $x|P'(x)$ is not contained in the range of $\widehat{P}$ because of $P'(x) \neq P(x)$. $(x|P'(x), \sigma)$ is validly forgeable signature for the DFS scheme. So, if there exists an adversary that breaks the verifiability of VDS with non-negligible probability, then there exists another adversary that breaks the unforgeability of DFS with the same probability. □

Theorem 5. *Let DFS be a distributed functional signature scheme against the security of function privacy (c.f. Definition 8) and Sig be an unforgeable signature scheme; then, our construction of VDS described above is secure in untrusted client security (c.f. Definition 11).*

Proof. We prove that if there exists a PPT adversary $\mathscr{A}$ that can break security in untrusted client security with non-negligible probability, then another adversary $\mathscr{B}$ can be constructed to break function privacy of DFS with the same probability.

Let $\mathscr{C}$ be the challenger for the DFS scheme. We construct the adversary $\mathscr{B}$ by $\mathscr{A}$ as follows:

(i) $\mathscr{B}$ runs the function private experiment of DFS and gets a master verifiable key mvk of DFS. Then, $\mathscr{B}$ samples a signing pair $(\text{Sig.sk}, \text{Sig.vk}) \longleftarrow \text{Sig.Sign}(1^\lambda)$.

(ii) The adversary $\mathscr{A}$ sends challenge program (P_0, P_1, S) to $\mathscr{B}$ such that $S \subset [n], |S| = n - 1$. $\mathscr{B}$ defines two new functions $\widehat{P}_0$ and $\widehat{P}_1$ which are described in Figure 2 and sends $(\widehat{P}_0, \widehat{P}_1, S)$ to the challenger $\mathscr{C}$ of DFS. The challenger returns $\left\{(\widehat{P}_i^b, \text{sk}_{\widehat{P}_b}^i)\right\}_{i \in S}$, sets $P_i = \left\{(\widehat{P}_i^b, \text{sk}_{\widehat{P}_b}^i)\right\}$, and sends $\{P_i\}_{i \in S}$ to adversary $\mathscr{A}$.

(iii) Proceeding adaptively, the adversary $\mathscr{A}$ can make the authentication query and evaluation query:

  (1) Authentication query: $\mathscr{B}$ initializes a dictionary indexed by $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$. When $\mathscr{A}$ queries an identity id_j, the challenger returns $\text{token}_{\text{id}_j}$ to adversary $\mathscr{A}$ if there exists a (id, j) in the dictionary. If not, the challenger returns $\text{token}_{\text{id}_j} = (\sigma_{\text{id}_j}, \text{id}_j, \text{mvk})$ and updates the dictionary $(\text{id}, j) \longrightarrow \text{token}_{\text{id}_j}$, where $\sigma_{\text{id}} = \text{Sig.Sign}(\text{Sig.sk}, \text{id}_j)$.

  (2) Evaluation query: $\mathscr{A}$ sends an encoding $\widehat{x}_k$ for the input x_k to $\mathscr{B}$. $\mathscr{B}$ first checks whether $P_0(x_k)$ is equal to $P_1(x_k)$. If not, $\mathscr{B}$ stops the experiment. Otherwise, $\mathscr{B}$ queries the evaluation oracle of DFS and returns the output to $\mathscr{A}$.

(iv) Finally, the adversary $\mathscr{B}$ outputs the result b' of the adversary $\mathscr{A}$.

We observe that $\mathscr{B}$ perfectly simulates the secure experiment in untrusted client model. Therefore, if $\mathscr{A}$ can distinguish the program P^0 and P^1 with non-negligible advantage, then $\mathscr{B}$ can break the function private security of DFS with the same advantage. □

6. Conclusion

In this work, we introduce the notion of distributed functional signature. In such model, it provides a possibility of function privacy in the setting of functional signature. We give a construction from function secret sharing and functional signature. The function secret sharing can be constructed based on learning with error assumption, and the functional signature can be constructed based on the existence of one-way function. Therefore, our construction can be obtained from one-way function and learning with error assumption. In addition, we apply it to the host service in multiple untrusted clouds in which the clouds will be dishonest. We require that the returned result from the clouds can be verified.

Data Availability

No underlying data were used in the study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the National Natural Science Foundation of China (nos. 61871430 and 61971458) and the Key Technologies R&D Program of Henan Province (nos. 212102210088 and 202102210169).

References

[1] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[2] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM*, vol. 17, no. 2, pp. 281–308, 1988.

[3] D. Chaum, "Blind signature system," in *Advances in Cryptology, Proc. CRYPTO'83*, D. Chaum, Ed., p. 153, Plenum Press, New York, NY, USA, 1984.

[4] D. Chaum and E. van Heyst, "Group signatures," in *Proceedings of the Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques*, pp. 257–265, Brighton, UK, April 1991.

[5] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Proceedings of the Advances in Cryptology-ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security*, pp. 552–565, Gold Coast, Australia, December 2001.

[6] J. Chang, H. Wang, F. Wang, A. Zhang, and Y. Ji, "RKA security for identity-based signature scheme," *IEEE Access*, vol. 8, pp. 17833–17841, 2020.

[7] J. Chang, Y. Ji, B. Shao, M. Xu, and R. Xue, "Certificateless homomorphic signature scheme for network coding," *IEEE/*

ACM Transactions on Networking, vol. 28, no. 6, pp. 2615–2628, 2020.

[8] J. Chang, B. Shao, Y. Ji, M. Xu, and R. Xue, "Secure network coding from secure proof of retrievability," *Science China Information Sciences [Ol], Early Access*, vol. 24, 2021.

[9] E. Boyle, S. Goldwasser, and I. Ivan, "Functional signatures and pseudorandom functions," in *Proceedings of the Public-Key Cryptography-PKC 2014-17th International Conference on Practice and Theory in Public-Key Cryptography*, pp. 501–519, Buenos Aires, Argentina, March 2014.

[10] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: improvements and extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1292–1303, Vienna, Austria, October 2016.

[11] D. Boneh, D. Gupta, I. Mironov, and A. Sahai, "Hosting services on an untrusted cloud," in *Proceedings of the Advances in Cryptology-EUROCRYPT 2015-34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 404–436, Sofia, Bulgaria, April 2015.

[12] X. Fan and Q. Tang, "Making public key functional encryption function private, distributively," in *Proceedings of the Public-Key Cryptography-PKC 2018-21st IACR International Conference on Practice and Theory of Public-Key Cryptography*, pp. 218–244, Rio de Janeiro, Brazil, March 2018.

[13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," in *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pp. 40–49, Berkeley, CA, USA, October, 2013.

[14] Z. Brakerski and G. N. Rothblum, "Virtual black-box obfuscation for all circuits via generic graded encoding," in *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014 Lecture Notes in Computer Science*, Y. Lindell, Ed., Springer, San Diego, CA, USA, 2014.

[15] Y. Hu and H. Jia, "Cryptanalysis of GGH map," in *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques Lecture Notes in Computer Science*, M. Fischlin and J. Coron, Eds., Springer, Vienna, Austria, 2016.

[16] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology, Proc. CRYPTO' 84, LNCS 196*, G. R. Blakley and D. C. Chaum, Eds., pp. 47–53, Springer, Berlin, Germany, 1985.

[17] M. Bellare and G. Fuchsbauer, "Policy-based signatures," in *Proceedings of the Public-Key Cryptography-PKC 2014-17th International Conference on Practice and Theory in Public-Key Cryptography*, pp. 520–537, Buenos Aires, Argentina, March 2014.

[18] M. Backes, S. Meiser, and D. Schröder, "Delegatable functional signatures," in *Proceedings of the Public-Key Cryptography-PKC 2016-19th IACR International Conference on Practice and Theory in Public-Key Cryptography*, pp. 357–386, Taipei, Taiwan, March 2016.

[19] R. Johnson, D. Molnar, D. X. Song, and D. A. Wagner, "Homomorphic signature schemes," in *Proceedings of the Topics in Cryptology-CT-RSA 2002, the Cryptographer's Track at the RSA Conference, 2002*, pp. 244–262, San Jose, CA, USA, February 2002.

[20] D. M. Freeman, "Improved security for linearly homomorphic signatures: a generic framework," in *Proceedings of the Public Key Cryptography-PKC 2012-15th International Conference on Practice and Theory in Public Key Cryptography*, pp. 697–714, Darmstadt, Germany, May 2012.

[21] D. Catalano, D. Fiore, and B. Warinschi, "Homomorphic signatures with efficient verification for polynomial functions," in *Proceedings of the Advances in Cryptology-CRYPTO 2014-34th Annual Cryptology Conference*, pp. 371–389, Santa Barbara, CA, USA, August 2014.

[22] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proceedings of the Advances in Cryptology-EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 457–473, Aarhus, Denmark, May 2005.

[23] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: definitions and challenges," in *Proceedings of the Theory of Cryptography-8th Theory of Cryptography Conference, TCC 2011*, pp. 253–273, Providence, RI, USA, March 2011.

[24] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "Reusable garbled circuits and succinct functional encryption," in *Proceedings of the Symposium on Theory of Computing Conference, STOC'13*, pp. 555–564, Palo Alto, CA, USA, June 2013.

[25] S. Goldwasser, S. D. Gordon, V. Goyal et al., "Multi-input functional encryption," in *Proceedings of the Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 578–602, Copenhagen, Denmark, May 2014.

[26] M. Abdalla, F. Bourse, A. D. Caro, and D. Pointcheval, "Simple functional encryption schemes for inner products," in *Proceedings of the Public-Key Cryptography-PKC 2015-18th IACR International Conference on Practice and Theory in Public-Key Cryptography*, pp. 733–751, Gaithersburg, MD, USA, April 2015.

[27] S. Agrawal, B. Libert, and D. Stehlé, "Fully secure functional encryption for inner products, from standard assumptions," in *Proceedings of the Advances in Cryptology-CRYPTO 2016-36th Annual International Cryptology Conference*, pp. 333–362, Santa Barbara, CA, USA, August 2016.

[28] F. Benhamouda, F. Bourse, and H. Lipmaa, "Cca-secure inner-product functional encryption from projective hash functions," in *Proceedings of the Public-Key Cryptography-PKC 2017-20th IACR International Conference on Practice and Theory in Public-Key Cryptography*, pp. 36–66, Amsterdam, The Netherlands, March 2017.

[29] T. Okamoto and K. Takashima, "Decentralized attribute-based signatures," in *Public-Key Cryptography-PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography Lecture Notes in Computer Science*, K. Kurosawa and G. Hanaoka, Eds., pp. 125–142, Springer, Nara, Japan, 2013.

[30] B. Liang and A. Mitrokotsa, "Decentralised functional signatures," *Mobile Networks and Applications*, vol. 24, no. 3, pp. 934–946, 2019.

[31] P. Datta, R. Dutta, and S. Mukhopadhyay, "Functional signcryption: notion, construction, and applications," in *Provable Security-9th International Conference, ProvSec 2015, Proceedings. Lecture Notes in Computer Science*, M. H. Au and A. Miyaji, Eds., Springer, Kanazawa, Japan, 2015.

[32] D. Pan, B. Liang, H. Li, and P. Ni, "Hierarchical functional signcryption: notion and construction," in *Provable Security-13th International Conference, ProvSec 2019 Lecture Notes in Computer Science*, R. Steinfeld and T. H. Yuen, Eds., pp. 167–185, Springer, Cairns, QLD, Australia, 2019.

[33] S. Li, B. Liang, and R. Xue, "Private functional signatures: definition and construction," in *Information Security and*

Privacy-23rd Australasian Conference, ACISP 2018 Lecture Notes in Computer Science, W. Susilo and G. Yang, Eds., Springer, Wollongong, NSW, Australia, 2018.

[34] M. Bellare, C. Namprempre, and G. Neven, “Unrestricted aggregate signatures,” in *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007 Lecture Notes in Computer Science*, L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, Eds., Springer, Wroclaw, Poland, 2007.