Authorized function homomorphic signature

Qingwen GUO

Qiong HUANG

Guomin YANG
Singapore Management University, gmyang@smu.edu.sg

© The British Computer Society 2018. All rights reserved.
For permissions, please e-mail: journals.permissions@oup.com
Advance Access publication on 3 November 2018
doi:10.1093/comjnl/bxy114

Authorized Function Homomorphic Signature

QINGWEN GUO[1], QIONG HUANG[1*] AND GUOMIN YANG[2]

[1]*College of Mathematics and Informatics, South China Agricultural University, Guangzhou 510642, China*
[2]*School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia*
[*]*Corresponding author: qhuang@scau.edu.cn*

Homomorphic signature (HS) is a novel primitive that allows an agency to carry out arbitrary (polynomial time) computation f on the signed data $\overrightarrow{m}$ and accordingly gain a signature σ_h for the computation result $f(\overrightarrow{m})$ with respect to f on behalf of the data owner (DO). However, since DO lacks control of the agency's behavior, receivers would believe that DO did authenticate the computation result even if the agency misbehaves and applies a function that the DO does not want. To address the problem above, in this paper we introduce a new primitive called *authorized function homomorphic signature* (AFHS). In AFHS, the agency has to obtain a confidence key sk_f from DO in order to evaluate a function f on the data $\overrightarrow{m}$ and to obtain a signature with which one can check whether the agency acts in accordance with DO's instructions. A black-box construction of AFHS based on HS is given in this paper, and we show that if the underlying primitives are secure, so is our construction under the given security model. Moreover, we provide a somewhat concrete construction that offers stronger security guarantee.

Keywords: digital signature; homomorphic signature; functional signature; zk-SNARKs; cloud computing

Received 5 April 2018; revised 5 August 2018; editorial decision 7 October 2018
Handling Editor: Joseph Liu

1. INTRODUCTION

Nowadays public and private organizations would like to upload their data to the cloud server $\mathcal{E}$ in order to mitigate the burden of the local computation and storage and to make good use of $\mathcal{E}$'s cloud computing capacity. Cloud computing entails that the computational process must be delegated to $\mathcal{E}$, while a disruptive server may behave dishonestly [1]. In modern cryptography, digital signature (DS) is a significant primitive and unforgeability of DS ensures that only with knowledge of the secret signing key can one generate a valid signature on messages. Homomorphic signature (HS) [2] is a handy tool for cloud computing security, and extends the functionality of DS in a way that one can do *arbitrary* (polynomial) computations f over DO's data $\overrightarrow{m}$ and meanwhile gain a new signature σ_h which authenticates the computation result $\hat{m} = f(\overrightarrow{m})$. The signature σ_h gives the receiver a reason to believe that the resulting pair $(f, f(\overrightarrow{m}))$ was approved by DO. In this paper, we introduce a new primitive called *authorized function homomorphic signature* (AFHS), to support a new emerging application.

Suppose that Alice is a project manager in a company, and she has signed some arguments $\overrightarrow{m} = (m_1, \ldots, m_k)$, which measure the quality of an important project (e.g. in a transnational agricultural product supply chain, m_1 denotes the environmental risks and m_2 denotes the requirements risk). We assume that the project arguments are kept secret to the public (e.g. the competitors). Then Alice transmits her data and the corresponding signatures to an assistant $\mathcal{E}$ (e.g. a leased cloud server) via a secure channel, and asks it to compute an assessment score $f(\overrightarrow{m})$ using an assigned estimation method f (e.g. multiple attribute decision making method based on triangular fuzzy numbers). Later, $\mathcal{E}$ passes the pair $(f, f(\overrightarrow{m}))$ to Bob (e.g. the boss who is on a business trip) to help him make critical decisions. Since $\mathcal{E}$ is semi-trusted and Bob may lack the necessary expertise to scrutinize the estimation method effectively, he requires Alice's signature on $(f, f(\overrightarrow{m}))$ to ensure that the claim value $f(\overrightarrow{m})$ is indeed the result of applying f on Alice's data $\overrightarrow{m}$ and that Alice did choose the estimation method f (e.g. an inappropriate and low-quality estimation method f^* may mislead Bob fatally).

AFHS provides the following capabilities. First, Alice can delegate the computation f to $\mathcal{E}$ and enable it to sign the result $f(\overrightarrow{m})$ on behalf of her. Second, only when the function f appointed by Alice is applied can $\mathcal{E}$ obtain a valid signature w.r.t. f. What's more, $\mathcal{E}$ can generate a valid signature if and only if the computation is carried out on Alice's data $\overrightarrow{m}$ correctly.

Figure 1 shows the system architecture of AFHS. Compared with HS, DO in this new notion can control the signing behavior of the agency, and enables it to provide a signature that proves the legitimacy of the message(s) from it. Our basic motivations are to prevent an agency from acquiring unlimited signing ability and to protect against the threat of disputes simultaneously. Concretely, the main contributions of our work are twofold.

1. In this paper, we introduce a new notion called *authorized function homomorphic signature* (AFHS). Since the agency must ask for a confidential key sk_f from DO in order to evaluate the function f, it is appropriate to restrict the signing ability of the agency.
2. We present the formal definition of AFHS as well as its security models. To demonstrate the new notion, we propose a generic construction taking advantages of HS and *functional signature* (FS) [3], and prove it to satisfy *weak* unforgeability (c.f. Definition 4.2). To enhance the security guarantee, we give another somewhat concrete construction of AFHS based on zk-SNARKs, which is inspired by Boyle et al's work on FS [3] and is proved to satisfy unforgeability (c.f. Definition 4.3).

To achieve the functionality of AFHS, a trivial way is that DO creates a certificate c_f for the prescribed function f and sends it to the agency. The agency calls the evaluation algorithm of HS, $\mathcal{HS}.\mathsf{Eval}$, to obtain a HS σ_h and submits $(f, f(\overrightarrow{m}), \sigma_h, c_f)$ to Bob. Bob checks the validity of f according to c_f and runs the verification algorithm of HS, $\mathcal{HS}.\mathsf{Verify}$, to verify the tuple $(f, f(\overrightarrow{m}))$. However, this solution will raise another security concern. Consider the following example.

The tuples $(\overrightarrow{m}, \overrightarrow{\sigma})$, $(\overrightarrow{m}', \overrightarrow{\sigma}')$ are sent to $\mathcal{E}_1$, $\mathcal{E}_2$, respectively. At some later point, DO issues a certificate c_f of function f for $\mathcal{E}_1$ so that it can generate a signature on $(f, f(\overrightarrow{m}))$ from $(\overrightarrow{m}, \overrightarrow{\sigma})$. Due to the publicity of c_f, $\mathcal{E}_2$ can also maliciously compute a signature on $(f, f(\overrightarrow{m}'))$ which is not desired for DO. Therefore, we need a better way to implement AFHS.

Paper Organization. Section 2 reviews the outline of some related works in the literature. In Section 3, we recall some necessary notions which will be used in our constructions. We present the definition of AFHS as well as its corresponding security models in Section 4. In Section 5, we propose a generic construction of AFHS which is based on FS and HS, and prove its security. Section 6 describes another somewhat concrete AFHS scheme in details, which is followed by a section discussing about the instantiation of the two AFHS schemes. Finally, this work is concluded in Section 8.

2. RELATED WORKS

2.1. Homomorphic signature

HS [2] allows any entity to combine the authenticated data and generate a signature for the new data without knowing the signer's private key. More precisely, given DO's data $\overrightarrow{m}$ and the signatures $\overrightarrow{\sigma}$ on $\overrightarrow{m}$, an assistant can invoke the algorithm $(\hat{m}, \sigma_h) \Leftarrow \mathcal{HS}.\mathsf{Eval}(\mathrm{pk}, \cdot f, \overrightarrow{\sigma})$ to show that the claimed value $\hat{m}$ is exactly the result of applying f to $\overrightarrow{m}$. In recently, Schabhüser *et al.* [4] have come up with an unforgeable linearly HS scheme which is secure under the DL and CDH assumption and allows for constant time verification. In

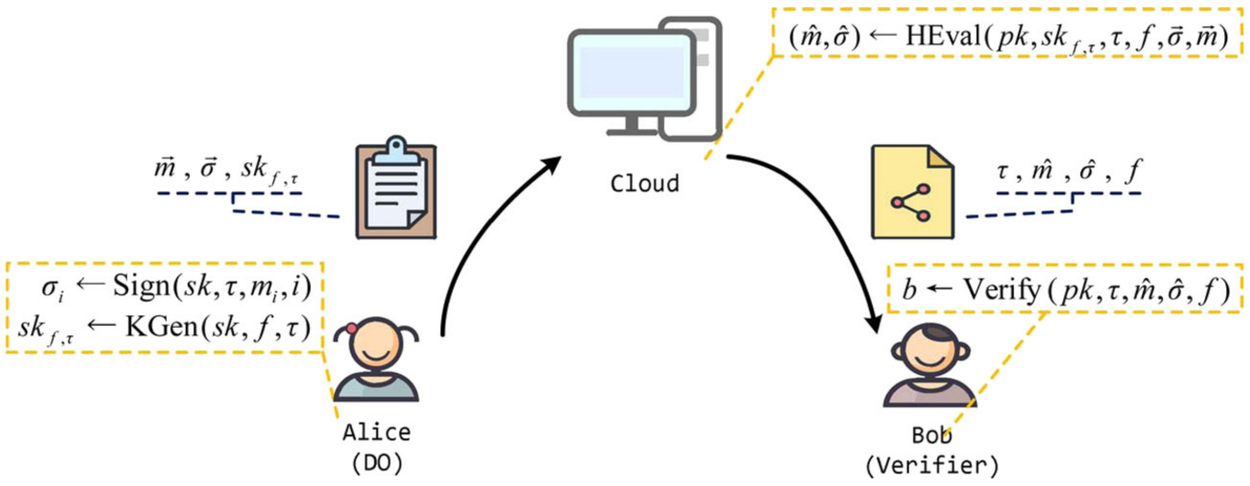
$(\hat{m},\hat{\sigma}) \leftarrow \mathrm{HEval}(pk, sk_{f,\tau}, \tau, f, \vec{\sigma}, \vec{m})$
$\vec{m}, \vec{\sigma}, sk_{f,\tau}$
Cloud
$\tau, \hat{m}, \hat{\sigma}, f$
$\sigma_i \leftarrow \mathrm{Sign}(sk, \tau, m_i, i)$
$sk_{f,\tau} \leftarrow \mathrm{KGen}(sk, f, \tau)$
$b \leftarrow \mathrm{Verify}(pk, \tau, \hat{m}, \hat{\sigma}, f)$
Alice
(DO)
Bob
(Verifier)

FIGURE 1. System architecture of AFHS.

addition, their scheme can be combined with homomorphic encryption to achieve context hiding. Tsabary [5] has proposed a concrete construction of attribute-based signature whose security relies on a worst-case lattice assumption, and shown that there exists an equivalence between context hiding HS and attribute-based signature. An amount of studies have been done on HS, notably from supporting linear functions [6–8, 4], to polynomial functions [9, 10] and fully homomorphic operations [11], and even to multi-key HS [12]. However, in most of the state-of-the-art HS schemes the agency has the ability to freely evaluate *arbitrary* functions. How to limit the signing ability of the agency is the focus of our work in this paper. Compared with HS, AFHS provides DO with more flexibility and reduces the possibility of dispute and loss. Our scheme guarantees that no one can persuade the third party to accept the result pair $(f^*, \hat{m}^*)$ while f^* is not assigned by DO or $\hat{m}^*$ is not the result of apply f^* to DO's data $\overrightarrow{m}$.

2.2. Functional signature

Another related notion is FS [3], in which DO hands out a secondary key sk_f to allow a specified agency to sign messages $\hat{m}$ which satisfy that $\hat{m} \in \mathrm{range}(f)$. The work of Boyle *et al.* [3] enlightens us and it is an important utility used in our first construction. The authors of [12] point out a new way of how to achieve FS through two-key HS (2-key HS). Besides, recently Backes *et al.* [13] introduced a new primitive called *delegatable functional signature* (DFS) which is closely related to FS. DFS considers not only the delegation of signing process but also a controlled form of malleability (i.e. DO delegates allowed functions $\mathcal{F}$ to an agency and the agency can further delegate the computations $\mathcal{F}' \subseteq \mathcal{F}$ to its sub-contractors).

2.3. Computation security and privacy

Due to the resource virtualization of cloud computing, the computation result from $\mathcal{E}$ cannot be fully trusted. Wei *et al.* [1] summarized three kinds of security problems in cloud computing: (i) storage-cheating attack, (ii) computation-cheating attack and (iii) privacy-cheating attack, of which the last two models are taken into consideration in our work.

Lots of researchers studying verifiable computation (VC) devote themselves to secure cloud computation [14–17]. In a VC scheme, DO is able to delegate the computation of a prescribed function to a powerful agency who can later output the computation result as well as a proof to prove that the computation was applied correctly to the data. There exist some differences between VC and AFHS, although they share certain similarity. In general, in VC it just requires that a spiteful agency cannot misguide the verification algorithm to accept $y^* \neq f(x)$ and unforgeability is not mandatory. Furthermore, a VC scheme providing input privacy w.r.t. the verifier (e.g. Bob) can be built from AFHS but the converse is not necessarily right. More details and further works about VC can be found in [15].

In the cloud computing applications, we should pay more attention to privacy protection in view of the potential risk of misuse of sensitive data. Recently, Li *et al.* [18] have put forward a creative protocol called scalable and privacy-preserving friend matching protocol (SPFM), which provides a friend matching solution in mobile cloud (that can be viewed as a special kind of computation) without compromising users' privacy. The principle is that each user in SPFM executes XOR operations to obfuscate every bit of the sensitive data. As for the privacy of DS in cloud computing, there are numerous works in the literature which have considered similar notions, e.g. completely context hiding, strongly context hiding, weakly context hiding for HS [19, 6], and unlinkability, transparency for sanitizable signature [20]. Overall, their goals are to assure that a new signature on the result message does not lead to disclosure of information about the original data. In AFHS, we demand that it is difficult for a greedy (but limited) competitor to infer which space the intercepted tuple $(\hat{m}, \hat{\sigma})$ comes from.

3. PRELIMINARIES

3.1. Notations

In this paper, the security parameter is represented by the notation λ, and we use $\mathrm{negl}(\cdot)$ to denote a negligible function and label a polynomial as $\mathrm{poly}(\cdot)$. The notation $y \Leftarrow x$ merely means that variable y is assigned the value x, and the notation $y \leftarrow \Omega$ denotes assigning to y an element uniformly chosen from a finite set Ω. Let the notation $\mathcal{X}_1 \overset{c}{\equiv} \mathcal{X}_2$ denote that no probabilistic polynomial time (PPT) adversary $\mathcal{A}$ can distinguish between $\mathcal{X}_1$ and $\mathcal{X}_2$.

3.2. Functional signature

Definition 3.1 (Functional signature, FS [3]). *A functional signature scheme consists of four PPT algorithms* $\mathcal{FS} =$ (Setup, KGen, Sign, Verify), *as follows.*

- Setup$(1^\lambda) \rightarrow (\mathrm{msk}, \mathrm{mvk})$: The setup algorithm creates a pair of keys $(\mathrm{msk}, \mathrm{mvk})$.
- KGen$(\mathrm{msk}, f) \rightarrow \mathrm{sk}_f$: Given msk and a function $f \in \mathcal{F}$, where $\mathcal{F}$ is the space of functions, this algorithm returns a key sk_f for f.
- Sign$(f, \mathrm{sk}_f, \overrightarrow{m}) \rightarrow (\hat{m}, \sigma_f)$: Taking as input a function f, a key sk_f issued for f and a bunch of messages $\overrightarrow{m}$, the signing algorithm outputs the computation result $\hat{m} = f(\overrightarrow{m})$ and a signature σ_f.
- Verify$(\mathrm{mvk}, \hat{m}, \sigma_f) \rightarrow b$: Given a signature σ_f on $\hat{m}$, this algorithm outputs either 1 (accept) or 0 (reject).

A functionality of FS is to guarantee that only appointed computation was performed (Refer to [3] for details). For completeness, below we recall its security modele. Consider the following game within which $\mathcal{A}$ is a PPT adversary and $i \in \mathbb{N}$.

(1) $\mathcal{A}$ is given mvk and has access to oracles $\mathcal{FS}.\mathcal{O}_{\mathsf{Key}}(\cdot)$ and $\mathcal{FS}.\mathcal{O}_{\mathsf{Sign}}(\cdot)$ which work as follows:

- $\mathcal{FS}.\mathcal{O}_{\mathsf{Key}}(f, i)$:
  - If there is an entry for the key (f, i) in the database, output the corresponding value sk^i_f.
  - Otherwise, return a fresh key $\mathrm{sk}^i_f \Leftarrow \mathcal{FS}.\mathsf{KGen}(\mathrm{msk}, f)$, and save (f, i, sk^i_f) to the database.
- $\mathcal{FS}.\mathcal{O}_{\mathsf{Sign}}(f, i, \overrightarrow{m})$:
  - If there is an entry (f, i, sk^i_f) in the database, output a functional signature with this corresponding key, i.e. $\sigma_f \Leftarrow \mathcal{FS}.\mathsf{Sign}(f, \mathrm{sk}^i_f, \overrightarrow{m})$.
  - Otherwise, generate a fresh key $\mathrm{sk}^i_f \Leftarrow \mathcal{FS}.\mathsf{KGen}(\mathrm{msk}, f)$, store (f, i, sk^i_f) to the database, and work as above.

(2) The adversary $\mathcal{A}$ wins the game if it is able to produce a forgery $(\hat{m}^*, \sigma^*_f)$ such that

(a) $\mathcal{FS}.\mathsf{Verify}(\mathrm{mvk}, \hat{m}^*, \sigma^*_f) = 1$; and
(b) $\nexists\ (\overrightarrow{m}, \mathrm{sk}_f)$ s.t. $\hat{m}^* = f(\overrightarrow{m}) \wedge \mathcal{FS}.\mathcal{O}_{\mathsf{Key}}(f, \cdot) \to \mathrm{sk}_f \wedge (\hat{m}^*, \sigma^*_f) \Leftarrow \mathcal{FS}.\mathsf{Sign}(f, \mathrm{sk}_f, \overrightarrow{m})$; and
(c) $\nexists\ (f, \overrightarrow{m})$ s.t. $\hat{m}^* = f(\overrightarrow{m}) \wedge \mathcal{FS}.\mathcal{O}_{\mathsf{Sign}}(f, \cdot, \overrightarrow{m}) \to \sigma^*_f$.

Definition 3.2 (Unforgeability [3]). *A FS scheme is* unforgeable *if for any PPT adversary $\mathcal{A}$, it holds that* $\Pr[\mathcal{A}\ wins] \le \mathrm{negl}(\lambda)$.

On the other side, it is required that a functional signature should reveal neither f nor $\overrightarrow{m}$ in the view of $\mathcal{A}$. More precisely, the privacy of FS is defined via the following game where $\mathcal{C}$ plays the role of challenger and $\mathcal{A}$ plays the part of PPT adversary.

(1) $\mathcal{C}$ obtains the key pair $(\mathrm{msk}, \mathrm{mvk}) \Leftarrow \mathcal{FS}.\mathsf{Setup}(1^\lambda)$ and runs $\mathcal{A}(\mathrm{msk}, \mathrm{mvk})$.
(2) During the training phase, $\mathcal{A}$ can choose two functions f_0, f_1 on its own terms, and ask for the corresponding keys $\mathrm{sk}_{f_0}, \mathrm{sk}_{f_1}$ from $\mathcal{C}$. Then $\mathcal{A}$ adaptively chooses $\overrightarrow{m}_0, \overrightarrow{m}_1$ and refers them to $\mathcal{C}$. If $f_0(\overrightarrow{m}_0) \neq f_1(\overrightarrow{m}_1)$, $\mathcal{C}$ aborts the game.
(3) $\mathcal{C}$ flips a coin $b \leftarrow \{0, 1\}$ to computes a challenge signature $\sigma_f \Leftarrow \mathcal{FS}.\mathsf{Sign}(f_b, \mathrm{sk}_{f_b}, \overrightarrow{m_b})$, and gives it to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs its guess b', and wins a victory if $b' = b$.

Definition 3.3 (Function privacy [3]). A FS scheme satisfies *function privacy* if for any PPT adversary $\mathcal{A}$, it holds that $|\Pr[\mathcal{A}\ wins] - 1/2| \le \mathrm{negl}(\lambda)$.

3.3. Homomorphic signature

Definition 3.4 (Homomorphic signature, HS [7, 9]). *A HS scheme consists of a tuple of four PPT algorithms* $\mathcal{HS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Eval})$, *as follows.*

- $\mathsf{Setup}(1^\lambda, k) \to (\mathrm{hsk}, \mathrm{hpk})$: Taking as input a security parameter and a variable k, the setup algorithm creates a pair of keys $(\mathrm{hsk}, \mathrm{hpk})$.
- $\mathsf{Sign}(\mathrm{hsk}, \tau, m, i) \to \sigma_i$: The signing algorithm gets as input hsk, a tag τ which is the class label of m, and an indicator i for which $1 \le i \le k$, and outputs a signature σ_i.
- $\mathsf{Verify}(\mathrm{hpk}, \tau, \hat{m}, \sigma_h, f) \to b$: This algorithm takes as input hpk, a tag τ, a result message $\hat{m}$, a signature σ_h that verifies the message $\hat{m} = f(\overrightarrow{m})$ for a vector of messages $\overrightarrow{m}$, and a function f, and outputs a bit b indicating whether the verifier accepts the tuple $(\tau, \hat{m}, f)$.
- $\mathsf{Eval}(\mathrm{hpk}, \tau, f, \overrightarrow{\sigma}) \to \sigma_h$: The evaluation algorithm gets as input hpk, a tag τ, a function f, and a tuple of signatures $\overrightarrow{\sigma}$ that verifies $\overrightarrow{m}$, and outputs a HS σ_h.

An immediate application of HS is to certify that the alleged value is indeed the computation result of applying f over the client's data (please refer to [7, 9] for details). The *unforgeability* of HS is defined via the following game in which $\mathcal{A}$ is a PPT adversary and $\mathcal{M}$ denotes the message space.

(1) $\mathcal{A}$ is given hpk and access to oracle $\mathcal{HS}.\mathcal{O}_{\mathsf{Sign}}(\tau_j, m)$ which works as follows:

- If $m \notin \mathcal{M}$, output $\perp$.
- If m is the first query for tag τ_j, set $i_j = 1$, output $\sigma_{i_j} \Leftarrow \mathcal{HS}.\mathsf{Sign}(\mathrm{hsk}, \tau_j, m, i_j)$ and initialize $\overrightarrow{m_j} = (m)$.
- If $i_j = k$, output $\perp$.
- Otherwise, set $i_j \Leftarrow i_j + 1$, output σ_{i_j} as above and set $\overrightarrow{m_j} = (m_1, \ldots, m_{i_j - 1}, m)$.

(2) Finally, $\mathcal{A}$ returns its forgery $(\tau^*, \hat{m}^*, \sigma_h^*, f^*)$, and wins a victory if $\mathcal{HS}.\mathsf{Verify}(\text{hpk}, \tau^*, \hat{m}^*, \sigma_h^*, f^*) = 1$ and either (a) $\tau^* \notin \{\tau_j\}$, or (b) $\tau^* = \tau_j$ for some j but $\hat{m}^* \neq f^*(\overrightarrow{m_j})$.

DEFINITION 3.5 (Unforgeability [7, 9]). *An HS scheme is* unforgeable *if for any PPT adversary $\mathcal{A}$, it holds that* $\Pr[\mathcal{A}\ wins] \leq \text{negl}(\lambda)$.

The *privacy* notion of HS captures the idea that even though hsk is leaked, $\mathcal{A}$ cannot deduce valuable information about client's data. Consider the following game played between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$.

(1) $\mathcal{C}$ gets the key pair $(\text{hsk}, \text{hpk}) \Leftarrow \mathcal{HS}.\mathsf{Setup}(1^\lambda, k)$, and later on it gives both keys to $\mathcal{A}$.
(2) Then $\mathcal{A}$ adaptively submits a tuple $(f, \overrightarrow{m_0}, \overrightarrow{m_1})$ which satisfies that $f(\overrightarrow{m_0}) = f(\overrightarrow{m_1})$. In response, $\mathcal{C}$ chooses a random bit $b \leftarrow \{0, 1\}$ and a random tag $\tau \leftarrow \{0, 1\}^\lambda$ to get a challenge signature. In details, $\mathcal{C}$ generates a bunch of signatures $\overrightarrow{\sigma_b}$ on $\overrightarrow{m_b}$ using tag τ, i.e. $\sigma_{bi} \Leftarrow \mathcal{HS}.\mathsf{Sign}(\text{hsk}, \tau, m_{bi}, i)$ and $\overrightarrow{\sigma_b} = (\sigma_{b1}, \ldots, \sigma_{bk})$, and outputs $\sigma_h \Leftarrow \mathcal{HS}.\mathsf{Eval}(\text{hpk}, \tau, f, \overrightarrow{\sigma_b})$.
(3) Finally, $\mathcal{A}$ outputs its guess b', and wins the game if $b' = b$.

DEFINITION 3.6 (Weakly context hiding [7, 9]). *An HS scheme satisfies* weakly context hiding *if for any PPT adversary $\mathcal{A}$, it holds that* $|\Pr[\mathcal{A}\ wins] - 1/2| \leq \text{negl}(\lambda)$.

3.4. Zero-knowledge SNARK

DEFINITION 3.7 (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge, zk-SNARK [3]). *A zk-SNARK system for a language $L \in \mathcal{NP}$ with witness relation $\mathcal{R}$ consists of a tuple of PPT algorithms* $\Pi_{\text{zk-SNARK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\text{crs}}, \mathcal{S}^{\mathsf{Prove}}), E_{P^*} = (E_{P^*}^1, E_{P^*}^2))$ *and satisfies the following properties*: completeness, adaptive soundness, adaptive zero-knowledge, succinctness and argument of knowledge.

- $\mathsf{Setup}(1^\lambda) \to \text{crs}$: Given a security parameter, the setup algorithm generates a common reference string crs.
- $\mathsf{Prove}(\text{crs}, \iota, \omega) \to \pi$: Holding a witness ω, the prover algorithm will output a proof π in order to prove that the given statement ι is true, i.e. $\iota \in L$ where $L \overset{\text{def}}{=} \{\iota: \exists \omega \text{ s.t. } \mathcal{R}(\iota, \omega) = 1\}$.
- $\mathsf{Verify}(\text{crs}, \iota, \pi) \to b$: This algorithm returns a bit $b \in \{0, 1\}$ which indicates whether the receiver accepts the statement.

Completeness: If $\text{crs} \Leftarrow \mathsf{Setup}(1^\lambda) \wedge \mathcal{R}(\iota, \omega) = 1$, it holds that

$$\Pr[\mathsf{Verify}(\text{crs}, \iota, \mathsf{Prove}(\text{crs}, \iota, \omega)) \to 0] \leq \text{negl}(\lambda).$$

Adaptive Soundness: It demands that a PPT adversary is unable to convince the verifier about a wrong statement, i.e.

$$\Pr[\mathsf{Verify}(\text{crs}, \iota^*, \pi) \to 1: \text{crs} \Leftarrow \mathsf{Setup}(1^\lambda) \wedge \mathcal{A}(\text{crs}) \to (\iota^*, \pi) \wedge \iota^* \notin L] \leq \text{negl}(\lambda).$$

Adaptive Zero-Knowledge: It means that $\mathcal{A}$ cannot learn any more other than the fact that $\iota \in L$ since whatever can be observed after the interaction with the prover can also be efficiently simulated by the adversary itself, i.e.

$$\Big|\Pr\Big[\text{crs} \Leftarrow \mathsf{Setup}(1^\lambda): \mathcal{A}^{\mathsf{Prove}(\text{crs},\iota,\omega)}(\text{crs}) \to 1\Big] - \Pr\Big[(\text{crs}, \text{td}) \Leftarrow \mathcal{S}^{\text{crs}}(1^\lambda): \mathcal{A}^{\mathcal{S}^{\mathsf{Prove}}(\text{crs},\text{td},\iota)}(\text{crs}) \to 1\Big]\Big| \leq \text{negl}(1^\lambda).$$

Succinctness: It requires that the size of the proof is much smaller than the length of the actual computation, i.e.

$$\forall(\iota, \omega) \text{ s.t. } \mathcal{R}(\iota, \omega) = 1, \text{crs} \Leftarrow \mathsf{Setup}(1^\lambda), \pi \Leftarrow \mathsf{Prove}(\text{crs}, \iota, \omega), |\pi| \leq \text{poly}(\lambda + \log \mathcal{R}).$$

Argument of Knowledge: It requests that a malicious prover is unable to output a valid proof without knowing a certain witness, i.e.

$$\Big|\Pr\Big[\text{crs} \Leftarrow \mathsf{Setup}(1^\lambda): \mathcal{A}(\text{crs}) \to 1\Big] - \Pr[(\text{crs}, \text{td}) \Leftarrow E_{P^*}^1(1^\lambda): \mathcal{A}(\text{crs}) \to 1]\Big| \leq \text{negl}(1^\lambda)$$

and

$$\Pr\Big[(\text{crs}, \text{td}) \Leftarrow E_{P^*}^1(1^\lambda), \mathcal{A}(\text{crs}) \to (\iota, \pi), E_{P^*}^2(\text{crs}, \text{td}, \iota, \pi) \to \omega^*: \mathsf{Verify}(\text{crs}, \iota, \pi) \to 1 \wedge \mathcal{R}(\iota, \omega^*) = 0\Big] \leq \text{negl}(1^\lambda).$$

A zk-SNARK system enables the receiver to verify the correctness of a specific computation efficiently while it cannot learn any more. More details about the construction of zk-SNARK can be found in [21].

4. AUTHORIZED FUNCTION HS

4.1. Formal definition

Definition 4.1 (Authorized Function Homomorphic Signature, AFHS). *An AFHS scheme consists of five PPT algorithms* (Setup, Sign, Verify, KGen, HEval), *as follows.*

- $\mathsf{Setup}(1^\lambda, k) \to (\mathrm{sk}, \mathrm{pk})$: Taking as input a variable k (i.e. the maximal data size), this algorithm creates a pair of keys (sk, pk). The secret key sk is kept by DO and the public key pk is published.
- $\mathsf{Sign}(\mathrm{sk}, \tau, m_i, i) \to \sigma_i$: DO runs this algorithm to sign the tuple (τ, m_i, i), where τ is the class tag of m_i and $1 \le i \le k$. Note that $\overrightarrow{m} = (m_1, \ldots, m_k)$.
- $\mathsf{Verify}(\mathrm{pk}, \tau, \hat{m}, \hat{\sigma}, f) \to b$: Given the computation result $(\tau, \hat{m}, f)$ with the corresponding signature $\hat{\sigma}$, the verifier calls this algorithm to determine whether to accept the computation result.
- $\mathsf{KGen}(\mathrm{sk}, f, \tau) \to \mathrm{sk}_{f,\tau}$: The algorithm is executed by DO to obtain a function key $\mathrm{sk}_{f,\tau}$ for f associated with τ.
- $\mathsf{HEval}(\mathrm{pk}, \mathrm{sk}_{f,\tau}, \tau, f, \overrightarrow{\sigma}, \overrightarrow{m}) \to (\hat{\sigma}, f(\overrightarrow{m}))$: The cloud server runs this algorithm to gain an evaluated signature $\hat{\sigma}$.

Correctness. Let $p_i \colon \mathcal{M}^k \to \mathcal{M}$ be a function which projects the input onto its i th component, i.e. $p_i(m_1, \ldots, m_k) = m_i$. We require that $\forall i \in [k]$, $p_i \in \mathcal{F}$. An AFHS scheme satisfies *correctness* if for any $f \in \mathcal{F}$, $m \in \mathcal{M}$ and $(\mathrm{sk}, \mathrm{pk}) \in \mathsf{Setup}(1^\lambda, k)$, it holds that

Signing. $\forall i \in [k]$, if $\sigma_i \Leftarrow \mathsf{Sign}(\mathrm{sk}, \tau, m_i, i)$, $\mathsf{Verify}(\mathrm{pk}, \tau, m_i, \sigma_i, p_i) = 1$;
Evaluation. If $\mathrm{sk}_{f,\tau} \Leftarrow \mathsf{KGen}(\mathrm{sk}, f, \tau)$, $(\hat{\sigma}, \hat{m}) \Leftarrow \mathsf{HEval}(\mathrm{pk}, \mathrm{sk}_{f,\tau}, \tau, f, \overrightarrow{\sigma}, \overrightarrow{m})$, $\mathsf{Verify}(\mathrm{pk}, \tau, \hat{m}, \hat{\sigma}, f) = 1$.

4.2. Security models

In our work, we ask that the proposed scheme must satisfy both *unforgeability* and *privacy*. Informally, unforgeability requires that (i) a malicious (but computationally bounded) adversary cannot do any illegal processing on the data on behalf of DO and (ii) if being allowed to evaluate a function f on the data $\overrightarrow{m}$, the adversary is incapable of forging a valid signature on $\hat{m}^* \ne f(\overrightarrow{m})$. Basically, the privacy demands that no one can deduce valuable information about DO's data from an honestly created pair $(\hat{m}, \hat{\sigma})$ in polynomial time. Below we describe the two security properties in details.

Unforgeability. Consider the following game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ (i.e. the malicious agency).

(1) The challenger $\mathcal{C}$ initializes $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{Setup}(1^\lambda, k)$ and calls $\mathcal{A}(\mathrm{pk})$.
(2) Within this game $\mathcal{A}$ is allowed to adaptively query a signing oracle $\mathcal{O}_{\mathsf{Sign}}(\cdot)$, a key generation oracle $\mathcal{O}_{\mathsf{KGen}}(\cdot)$, and an evaluation oracle $\mathcal{O}_{\mathsf{HEval}}(\cdot)$ for polynomially many times, which are defined as follows. Here we assume that if a query was issued before, the oracle will respond directly to $\mathcal{A}$ using the previous answer.

- $\mathcal{O}_{\mathsf{Sign}}(\tau, m)$: The signing oracle gets as input a message m and a tag τ, and returns a signature σ_i. This oracle is subject to the condition that at most k messages can be queried for a tag τ.
- $\mathcal{O}_{\mathsf{KGen}}(f, \tau)$: The key generation oracle will return a key $\mathrm{sk}_{f,\tau}$ for the function f associated with τ.
- $\mathcal{O}_{\mathsf{HEval}}(\tau, f, \overrightarrow{\sigma}, \overrightarrow{m})$: The evaluation oracle uses the public key pk and a vector of signatures $\overrightarrow{\sigma}$ to generate an evaluated signature σ_h for the tuple $(\tau, f, f(\overrightarrow{m}))$.

(3) Finally, $\mathcal{A}$ returns its forgery $(\tau^*, \hat{m}^*, \hat{\sigma}^*, f^*)$ such that it did not query oracle $\mathcal{O}_{\mathsf{HEval}}(\cdot)$ on input $(\tau^*, f^*, \cdot, \cdot)$ for which the oracle outputs a signature on $(\tau^*, f^*, \hat{m}^*)$.

Definition 4.2 (Weak unforgeability). *We say that an AFHS scheme is* weakly unforgeable *if $\mathcal{A}$ cannot output a forgery $(\tau^*, \hat{m}^*, \hat{\sigma}^*, f^*)$ such that,* $\mathsf{Verify}(\mathrm{pk}, \tau^*, \hat{m}^*, \hat{\sigma}^*, f^*) = 1$ *and* Type_1

Type_1 $\mathcal{A}$ did not query oracle $\mathcal{O}_{\mathsf{KGen}}(\cdot)$ about (f', τ^*) such that $\hat{m}^* \in \mathrm{range}(f')$.
Type_2 $\tau^* \ne \tau_j$ for all τ_j queried by $\mathcal{A}$.
Type_3 $\tau^* = \tau_j$ for some j but $\hat{m}^* \ne f^*(\overrightarrow{m_j})$. Namely, the adversary authenticates an incorrect value of a function on the previously seen messages.

Definition 4.3 (Unforgeability). *We say that an AFHS scheme is* unforgeable *if $\mathcal{A}$ cannot output a forgery $(\tau^*, \hat{m}^*, \hat{\sigma}^*, f^*)$ such that,* $\mathsf{Verify}(\mathrm{pk}, \tau^*, \hat{m}^*, \hat{\sigma}^*, f^*) = 1$ *and either of the following holds*:

Type_1 $\mathcal{A}$ did not query oracle $\mathcal{O}_{\mathsf{KGen}}(\cdot)$ about (f^*, τ^*).
Type_2 $\tau^* \ne \tau_j$ for all τ_j queried by $\mathcal{A}$.
Type_3 $\tau^* = \tau_j$ for some j but $\hat{m}^* \ne f^*(\overrightarrow{m_j})$.

The latter security model (w.r.t. Definition 4.2) is defined as the former one (w.r.t. Definition 4.3) except for the first condition. More concretely, in the former model a signature σ^* on $(\tau^*, \hat{m}^*, f^*)$ is not considered as a Type_1 forgery if $\mathcal{A}$ possesses a function key with regard to (f', τ^*) and $\hat{m}^* \in \mathrm{range}(f') \cap \mathrm{range}(f^*)$. In contrast, $\mathcal{A}$ in the latter model is granted more power since it has no such limitations, i.e. we only ask that $\mathcal{A}$ cannot hold a function key with regard to (f^*, τ^*) when it chooses to commit a Type_1 forgery.

Privacy. The privacy notion we consider in AFHS is called *weakly context hiding*, defined via the following game.

(1) $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda, k)$ to get a key pair (sk, pk), and invokes the adversary $\mathcal{A}$ on input (pk, sk).
(2) $\mathcal{A}$ outputs two vectors of messages $\overrightarrow{m_0}, \overrightarrow{m_1} \in \mathcal{M}^k$ and a function f such that $f(\overrightarrow{m_0}) = f(\overrightarrow{m_1})$. $\mathcal{C}$ selects a random bit $b \leftarrow \{0, 1\}$, signs $(\overrightarrow{m_b}, \tau)$ where $\tau \leftarrow \{0, 1\}^\lambda$ and outputs $\overrightarrow{\sigma}$. Then $\mathcal{C}$ generates a key $\mathrm{sk}_{f,\tau} \Leftarrow \mathsf{KGen}(\mathrm{sk}, f, \tau)$ and computes $(\hat{\sigma}, f(\overrightarrow{m})) \Leftarrow \mathsf{HEval}(\mathrm{pk}, \mathrm{sk}_{f,\tau}, \tau, f, \overrightarrow{\sigma}, \overrightarrow{m})$. Finally, $\mathcal{C}$ sends $(\tau, \hat{m}, \hat{\sigma})$ to $\mathcal{A}$.
(3) Finally, $\mathcal{A}$ outputs a bit b' and wins the game if $b' = b$.

The *advantage* of $\mathcal{A}$ in the game above is defined to be the difference between the probability that $\mathcal{A}$ wins and $1/2$.

DEFINITION 4.4 (Weak context hiding). *An AFHS scheme is* weakly context hiding *if for any PPT adversary $\mathcal{A}$, its advantage in the game above is negligible.*

Notice that the word 'weak' refers to the fact that the original signatures on messages $\overrightarrow{m_b}$ are kept secret to $\mathcal{A}$.

5. A GENERIC CONSTRUCTION

5.1. The construction

Below we show a generic construction that transforms a traditional HS scheme into an AFHS scheme, by employing a functional signature scheme as a building block. Let $\mathcal{HS}$ be a HS scheme and $\mathcal{FS}$ be a functional signature scheme. Our generic construction works as below.

- $\mathsf{Setup}(1^\lambda, k)$:
  - $(\mathrm{hsk}, \mathrm{hpk}) \Leftarrow \mathcal{HS}.\mathsf{Setup}(1^\lambda, k)$;
  - $(\mathrm{msk}, \mathrm{mvk}) \Leftarrow \mathcal{FS}.\mathsf{Setup}(1^\lambda)$;
  - $\mathrm{sk} \Leftarrow (\mathrm{hsk}, \mathrm{msk})$;
  - $\mathrm{pk} \Leftarrow (\mathrm{hpk}, \mathrm{mvk})$;
  - Output (sk, pk).
- $\mathsf{Sign}(\mathrm{sk}, \tau, m_i, i)$:
  - Parse sk as (hsk, msk);
  - Output $\sigma_i \Leftarrow \mathcal{HS}.\mathsf{Sign}(\mathrm{hsk}, \tau, m_i, i)$.
- $\mathsf{Verify}(\mathrm{pk}, \tau, \hat{m}, \hat{\sigma}, f)$:
  - Parse $\hat{\sigma}$ as (σ_h, σ_g);
  - Parse pk as (hpk, mvk);
  - If $\mathcal{HS}.\mathsf{Verify}(\mathrm{hpk}, \tau, \hat{m}, \sigma_h, f) = 1$ and $\mathcal{FS}.\mathsf{Verify}(\mathrm{mvk}, \hat{m}\|\tau, \sigma_g) = 1$, output 1;
  - Otherwise, output 0.
- $\mathsf{KGen}(\mathrm{sk}, f, \tau)$:
  - Parse sk as (hsk, msk);
  - Encode $g(\cdot) = f(\cdot)\|\tau$;
  - Output $\mathrm{sk}_{f,\tau} \Leftarrow \mathcal{FS}.\mathsf{KGen}(\mathrm{msk}, g)$.
- $\mathsf{HEval}(\mathrm{pk}, \mathrm{sk}_{f,\tau}, \tau, f, \overrightarrow{\sigma}, \overrightarrow{m})$:
  - Parse pk as (hpk, mvk);
  - Encode $g(\cdot) = f(\cdot)\|\tau$;
  - $\sigma_h \Leftarrow \mathcal{HS}.\mathsf{Eval}(\mathrm{hpk}, \tau, f, \overrightarrow{\sigma})$;
  - $(f(\overrightarrow{m})\|\tau, \sigma_g) \Leftarrow \mathcal{FS}.\mathsf{Sign}(g, \mathrm{sk}_{f,\tau}, \overrightarrow{m})$;
  - $\hat{\sigma} \Leftarrow (\sigma_h, \sigma_g)$;
  - Output $(\hat{\sigma}, f(\overrightarrow{m}))$.

Correctness of the scheme is straightforward, and we omit it here. Below we analyze its security.

5.2. Security analysis

THEOREM 5.1. *If $\mathcal{HS}$ is an unforgeable* HS *scheme and $\mathcal{FS}$ is an unforgeable functional signature scheme, our AFHS scheme above is weakly unforgeable.*

The intuition behind is that to win the unforgeability game, the PPT algorithm $\mathcal{A}$ has to forge either a signature with respect to $\mathcal{FS}$ or a signature with respect to $\mathcal{HS}$: (i) A Type_1 forgery implies that the challenger didn't issue $\mathrm{sk}_g = \mathrm{sk}_{f,\tau}$ which can be used to sign $\hat{m}^*$ (i.e. $\hat{m}^* \notin \{\mathrm{range}(f)\}_{Query}$), and $\mathcal{A}$ must break the unforgeability of $\mathcal{FS}$; (ii) A Type_2 forgery or a Type_3 forgery leads to an attack against $\mathcal{HS}$ since $\mathcal{HS}$ requires that $\mathcal{A}$ is unable to sign $(f^*, \hat{m}^*, \tau^*)$ such that $\tau^* \notin \{\tau\}_{Query}$ or $\hat{m}^* \notin \{f^*(\overrightarrow{m})\}_{Query}$.

Proof. Denote by Forge_α the event that $\mathcal{A}$ outputs a Type_α forgery. We have

$$\begin{aligned}\Pr[\mathcal{A}\text{ wins}] &= \Pr[\mathsf{Forge}_1 \vee \mathsf{Forge}_2 \vee \mathsf{Forge}_3] \\ &\leq \Pr[\mathsf{Forge}_1] + \Pr[\mathsf{Forge}_2] + \Pr[\mathsf{Forge}_3].\end{aligned}$$

Below we show that each item in the right-hand side is negligible.

If $\mathcal{A}$ outputs a valid Type_1 forgery efficiently, we are able to build another algorithm $\mathcal{A}_{\mathcal{FS}}$ to break the unforgeability of $\mathcal{FS}$. $\mathcal{A}_{\mathcal{FS}}$ is given a key mvk from its own challenger, and has access to oracles $\mathcal{FS}.\mathcal{O}_{\mathsf{Key}}(\cdot)$ and $\mathcal{FS}.\mathcal{O}_{\mathsf{Sign}}(\cdot)$. It works as below.

(1) $\mathcal{A}_{\mathcal{FS}}$ invokes $\mathcal{HS}.\mathsf{Setup}(1^\lambda, k)$ to obtain (hsk, hpk), sets pk $\Leftarrow$ (hpk, mvk), and runs $\mathcal{A}$ on input pk.
(2) It answers $\mathcal{A}$'s queries to oracles $\mathcal{O}_{\mathsf{Sign}}$, $\mathcal{O}_{\mathsf{KGen}}$ and $\mathcal{O}_{\mathsf{HEval}}$ as follows.

- $\mathcal{O}_{\mathsf{Sign}}(\tau, m)$:
    - If m is the first query for τ, set $i_\tau = 1$, output the signature $\sigma_i \Leftarrow \mathcal{HS}.\mathsf{Sign}(\text{hsk}, \tau, m, i_\tau)$.
    - If $i_\tau = k$, abort.
    - Otherwise, increase the counter i_τ by 1, output the corresponding signature σ_i.
- $\mathcal{O}_{\mathsf{KGen}}(f, \tau)$: Obtain $\text{sk}_{f,\tau}$ from $\mathcal{FS}.\mathcal{O}_{\mathsf{Key}}(g, \tau)$ where $g(\cdot) = f(\cdot)\|\tau$, and output $\text{sk}_{f,\tau}$.
- $\mathcal{O}_{\mathsf{HEval}}(\tau, f, \overrightarrow{\sigma}, \overrightarrow{m})$:
    - Compute $\sigma_h \Leftarrow \mathcal{HS}.\mathsf{Eval}(\text{hpk}, \tau, f, \overrightarrow{\sigma})$.
    - If the entry $(f, \tau, \text{sk}_{f,\tau})$ can be found in the database, sign $f(\overrightarrow{m})$ using this key, i.e. $(g(\overrightarrow{m}), \sigma_g) \Leftarrow \mathcal{FS}.\mathsf{Sign}(g, \text{sk}_{f,\tau}, \overrightarrow{m})$.
    - Otherwise, submit the tuple (f, τ) to $\mathcal{FS}.\mathcal{O}_{\mathsf{Key}}(\cdot)$ to get a key $\text{sk}_{f,\tau}$, add a new entry $(f, \tau, \text{sk}_{f,\tau})$ to the database, and sign $f(\overrightarrow{m})$ as above.
    - Output $\hat{\sigma} \Leftarrow (\sigma_h, \sigma_g)$.

(3) Finally, $\mathcal{A}$ outputs a forgery $(\tau^*, \hat{m}^*, \hat{\sigma}^*, f^*)$ where $\hat{\sigma}^* = (\sigma_h^*, \sigma_g^*)$. $\mathcal{A}_{\mathcal{FS}}$ outputs $(\hat{m}^*\|\tau^*, \sigma_g^*)$ as its own forgery.

We can see that the view of $\mathcal{A}$ in the game above is identically distributed to that in a real attack against our AFHS scheme. If the output of $\mathcal{A}$ is a valid Type_1 forgery, $(\hat{m}^*\|\tau^*, \sigma_g^*)$ is also a valid forgery to the underlying FS scheme. Guaranteed by the unforgeability of $\mathcal{FS}$, we conclude that $\Pr[\mathsf{Forge}_1]$ is negligible.

If $\mathcal{A}$ outputs a valid Type_2 forgery or a valid Type_3 forgery, we can use it to build another algorithm $\mathcal{A}_{\mathcal{HS}}$ to break the unforgeability of $\mathcal{HS}$. In details, $\mathcal{A}_{\mathcal{HS}}$ is given as input a key hpk, and has access to oracle $\mathcal{HS}.\mathcal{O}_{\mathsf{Sign}}(\cdot)$. It works as below.

- $\mathcal{A}_{\mathcal{HS}}$ invokes $\mathcal{FS}.\mathsf{Setup}(1^\lambda)$ to obtain (msk, mvk), sets pk $\Leftarrow$ (hpk, mvk), and runs $\mathcal{A}$ on input pk.
- It answers $\mathcal{A}$'s queries to oracles $\mathcal{O}_{\mathsf{Sign}}$, $\mathcal{O}_{\mathsf{KGen}}$, and $\mathcal{O}_{\mathsf{HEval}}$ as follows.
    - $\mathcal{O}_{\mathsf{Sign}}(\tau, m)$:
        - If m is the first query for τ, set $i_\tau = 1$, output $\sigma_i \Leftarrow \mathcal{HS}.\mathcal{O}_{\mathsf{Sign}}(\tau, m, i_\tau)$.
        - If $i_\tau = k$, abort.
        - Otherwise, increase the counter i_τ by 1, and output whatever returned by $\mathcal{HS}.\mathcal{O}_{\mathsf{Sign}}(\tau, m, i_\tau)$.
    - $\mathcal{O}_{\mathsf{KGen}}(f, \tau)$:
        - If there is no entry $(f, \tau, \text{sk}_{f,\tau})$ in the database, output the key $\text{sk}_{f,\tau} \Leftarrow \mathcal{FS}.\mathsf{KGen}(\text{msk}, g)$ where $g(\cdot) = f(\cdot)\|\tau$, and add an entry $(f, \tau, \text{sk}_{f,\tau})$ to the database.
        - Otherwise, directly output the corresponding key $\text{sk}_{f,\tau}$.
    - $\mathcal{O}_{\mathsf{HEval}}(\tau, f, \overrightarrow{\sigma}, \overrightarrow{m})$:
        - Derive $\sigma_h \Leftarrow \mathcal{HS}.\mathsf{Eval}(\text{hpk}, \tau, f, \overrightarrow{\sigma})$.
        - If the entry $(f, \tau, \text{sk}_{f,\tau})$ can be found in the database, sign $f(\overrightarrow{m})$ using this key.
        - Otherwise, execute $\text{sk}_{f,\tau} \Leftarrow \mathcal{FS}.\mathsf{KGen}(\text{msk}, g)$, add $(f, \tau, \text{sk}_{f,\tau})$ to the database, and sign $f(\overrightarrow{m})$.
        - Output $\hat{\sigma} \Leftarrow (\sigma_h, \sigma_g)$.
- Finally, $\mathcal{A}$ outputs $(\tau^*, \hat{m}^*, \hat{\sigma}^*, f^*)$ where $\hat{\sigma}^* = (\sigma_h^*, \pi_f^*)$. $\mathcal{A}_{\mathcal{HS}}$ outputs $(\tau^*, \hat{m}^*, \sigma_h^*, f^*)$ as its own forgery.

The view of $\mathcal{A}$ in the game above is identically distributed as that in a real attack. If the output of $\mathcal{A}$ is a valid Type_2 forgery or Type_3 forgery, $(\tau^*, \hat{m}^*, \sigma_h^*, f^*)$ is also a valid forgery against the underlying HS scheme. Guaranteed by the unforgeability of $\mathcal{HS}$, it is not difficult to conclude that $\Pr[\mathsf{Forge}_2]$ and $\Pr[\mathsf{Forge}_3]$ are both negligible.

This completes the proof. □

Theorem 5.2. *If $\mathcal{HS}$ is weakly context hiding and $\mathcal{FS}$ satisfies the privacy requirement for functional signatures, our construction above is weakly context hiding.*

To prove the privacy of AFHS, we actually need to show that

$$\left(\tau, \pi_f^{f(\overrightarrow{m_0})}, \sigma_h^{f(\overrightarrow{m_0})}\right) \overset{c}{\equiv} \left(\tau, \pi_f^{f(\overrightarrow{m_1})}, \sigma_h^{f(\overrightarrow{m_1})}\right). \quad (1)$$

The proof proceeds in two steps. First we prove that

$$\left(\tau, \pi_f^{f(\overrightarrow{m_0})}, \sigma_h^{f(\overrightarrow{m_0})}\right) \overset{c}{\equiv} \left(\tau, \pi_f^{f(\overrightarrow{m_0})}, \sigma_h^{f(\overrightarrow{m_1})}\right) \quad (2)$$

based on the fact that $\mathcal{HS}$ is *weakly context hiding*. Then we prove that

$$\left(\tau, \pi_f^{f(\vec{m}_0)}, \sigma_h^{f(\vec{m}_1)}\right) \overset{c}{\equiv} \left(\tau, \pi_f^{f(\vec{m}_1)}, \sigma_h^{f(\vec{m}_1)}\right) \quad (3)$$

by relying on the *function privacy* of $\mathcal{FS}$. Then we can conclude that equation (1) holds.

Proof. To prove equation (2), we reduce it to the weakly context hiding property of $\mathcal{HS}$. Assume that $\mathcal{A}_1$ is an efficient algorithm which can break the claim. We build an efficient algorithm $\mathcal{A}_{\mathcal{HS}}$ that uses $\mathcal{A}_1$ as a subroutine to break the weak privacy of $\mathcal{HS}$ (c.f. Definition 3.6). The algorithm $\mathcal{A}_{\mathcal{HS}}$ is given as input a key pair (hsk, hpk), and works as below.

- $\mathcal{A}_{\mathcal{HS}}$ runs $\mathcal{FS}.\mathsf{Setup}(1^\lambda)$ to obtain (msk, mvk), sets sk $\Leftarrow$ (hsk, msk) and pk $\Leftarrow$ (hpk, mvk), and invokes $\mathcal{A}_1$ on input (sk, pk).
- $\mathcal{A}_1$ submits $(\vec{m}_0, \vec{m}_1, f)$. If $f(\vec{m}_0) \neq f(\vec{m}_1)$, $\mathcal{A}_{\mathcal{HS}}$ aborts and outputs a random bit; otherwise, it submits $(\vec{m}_0, \vec{m}_1, f)$ to its own challenger, and is returned a signature σ_h and a random tag τ.
- $\mathcal{A}_{\mathcal{HS}}$ computes $\mathrm{sk}_{f,\tau} \Leftarrow \mathcal{FS}.\mathsf{KGen}(\mathrm{msk}, g)$ where $g(\cdot) = f(\cdot)\|\tau$, and generates a signature $\sigma_g \Leftarrow \mathcal{FS}.\mathsf{Sign}(g, \mathrm{sk}_{f,\tau}, \vec{m}_0)$. It returns $(\tau, \sigma_g, \sigma_h)$ to the adversary.
- $\mathcal{A}_{\mathcal{HS}}$ outputs the bit b' that is output by $\mathcal{A}_1$.

If $b = 0$, what $\mathcal{A}_1$ obtains is the left hand side of equation (2); otherwise, $\mathcal{A}_1$ is given the right-hand side of equation (2). By the assumption that $\mathcal{HS}$ is weakly context hiding, the advantage of $\mathcal{A}_1$ in distinguishing the two sides of equation (2) is thus negligible.

To prove equation (3), we reduce it to the function privacy property of $\mathcal{FS}$. Assume that $\mathcal{A}_2$ is an efficient algorithm which can break the claim. We construct an efficient algorithm $\mathcal{A}_{\mathcal{FS}}$ that uses $\mathcal{A}_2$ as a building block to break the function privacy of $\mathcal{FS}$ (c.f. Definition 3.3). The algorithm $\mathcal{A}_{\mathcal{FS}}$ is given as input a key pair (msk, mvk), and works as below.

- $\mathcal{A}_{\mathcal{FS}}$ runs $\mathcal{HS}.\mathsf{Setup}(1^\lambda, k)$ to obtain (hsk, hpk), sets sk $\Leftarrow$ (hsk, msk) and pk $\Leftarrow$ (hpk, mvk), and gives (sk, pk) to the adversary $\mathcal{A}_2$.
- $\mathcal{A}_{\mathcal{FS}}$ receives $(\vec{m}_0, \vec{m}_1, f)$ from $\mathcal{A}_2$. If $f(\vec{m}_0) \neq f(\vec{m}_1)$, it aborts and outputs a random bit; otherwise, it samples a tag τ uniformly, sets $g_0(\cdot) = g_1(\cdot) = f(\cdot)\|\tau$, and receives $\mathrm{sk}_{g_0} = \mathrm{sk}_{g_1}$ from its own challenger.
- $\mathcal{A}_{\mathcal{FS}}$ submits $(\vec{m}_0, \vec{m}_1, g_0, g_1)$ to its own challenger and is returned a signature σ_g. It signs the messages in $\vec{m}_1$ using the tag τ and obtains a vector $\vec{\sigma}$ of k signatures, and computes a signature $\sigma_h \Leftarrow \mathcal{HS}.\mathsf{Eval}(\mathrm{pk}, \tau, f, \vec{\sigma})$. It returns $(\tau, \sigma_g, \sigma_h)$ to the adversary.
- $\mathcal{A}_{\mathcal{FS}}$ outputs the bit b' that is output by $\mathcal{A}_2$.

By the assumption that $\mathcal{FS}$ is function-private, the advantage of $\mathcal{A}_2$ is negligible.

This completes the proof. □

6. ANOTHER CONSTRUCTION

Here we present a trivial example to better illustrate why the first construction cannot achieve *unforgeability* (c.f. Definition 4.3). Suppose that DO has sent $(\vec{m}, \vec{\sigma}, \mathrm{sk}_{f',\tau^*})$ to $\mathcal{A}$. To get a valid signature on $\hat{m}^* = f^*(\vec{m})$, all $\mathcal{A}$ has to do is to find $\vec{m}'$ such that $f'(\vec{m}') = f^*(\vec{m})$ and run $\sigma_h^* \Leftarrow \mathcal{HS}.\mathsf{Eval}(\mathrm{hpk}, \tau^*, f^*, \vec{\sigma})$ and $(\hat{m}^*\|\tau^*, \sigma_{f'\|\tau^*}) \Leftarrow \mathcal{FS}.\mathsf{Sign}(f'\|\tau^*, \mathrm{sk}_{f',\tau^*}, \vec{m}')$, respectively. Eventually, $\mathcal{A}$ outputs $(\sigma_h^*, \sigma_{f'\|\tau^*})$ as its final forgery (w.r.t. Type_1). The main reason causing such a problem is that in FS an adversary is not permitted to sign $\hat{m}^*$ if it has requested a key $\mathrm{sk}_{f'}$ for which $\hat{m}^* \in \mathrm{range}(f')$. Therefore, we tailor the definition of unforgeability to capture it, and prove the AFHS construction above to be *weakly unforgeable* (c.f. Definition 4.2). In other words, inheriting from FS our first construction also has a restriction that $\mathcal{A}$ cannot send (f', τ^*) to oracle $\mathcal{O}_{\mathsf{KGen}}(\cdot)$ such that $\hat{m}^* \in \mathrm{range}(f')$. Below we provide another construction of AFHS, which enhances the security guarantee and satisfies the *unforgeability* (c.f. Definition 4.3).

6.1. The new construction

The new construction is inspired by Boyle *et al.*'s second construction of functional signature [3]. A main idea is to bind $\sigma_{f\|\tau}$ and σ_h together, so that the adversary could not replace $\sigma_{f\|\tau}$ with another. Let $\mathcal{DS}$ be a DS scheme, and $\Pi_{\text{zk-SNARK}}$ be an adaptive zero-knowledge SNARK system for the following $\mathcal{NP}$ language:

$$\begin{aligned} L = \{(\tau, f, \hat{m}, \mathrm{dpk})\colon \exists\ (\vec{m}, \vec{\sigma}, \sigma_{f\|\tau})\} \\ \text{s.t. } \forall i \in [k]\ \mathcal{DS}.\mathsf{Verify}(\mathrm{dpk}, \tau\|m_i\|i, \sigma_i) = 1 \wedge \\ f(\vec{m}) = \hat{m} \wedge \mathcal{DS}.\mathsf{Verify}(\mathrm{dpk}, f\|\tau, \sigma_{f\|\tau}) = 1\}. \end{aligned}$$

Our concrete construction works as below.

- $\mathsf{Setup}(1^\lambda, k)$:
  - $(\mathrm{dsk}, \mathrm{dpk}) \Leftarrow \mathcal{DS}.\mathsf{Setup}(1^\lambda)$;
  - $\mathrm{crs} \Leftarrow \Pi_{\text{zk-SNARK}}.\mathsf{Setup}(1^\lambda)$;
  - sk $\Leftarrow$ dsk;
  - pk $\Leftarrow$ (dpk, crs);
  - Output (sk, pk).
- $\mathsf{Sign}(\mathrm{sk}, \tau, m_i, i)$:
  - $\sigma_i \Leftarrow \mathcal{DS}.\mathsf{Sign}(\mathrm{dsk}, \tau\|m_i\|i)$;
  - Output σ_i.
- $\mathsf{Verify}(\mathrm{pk}, \tau, \hat{m}, \sigma_h, f)$:
  - Parse pk as (dpk, crs);
  - $b \Leftarrow \Pi_{\text{zk-SNARK}}.\mathsf{Verify}(\mathrm{crs}, (\tau, f, \hat{m}, \mathrm{dpk}), \sigma_h)$;

- Output b.

- $\mathsf{KGen}(\mathrm{sk}, f, \tau)$:
  - $\mathrm{sk}_{f,\tau} \Leftarrow \mathcal{DS}.\mathsf{Sign}(\mathrm{dsk}, f\|\tau)$;
  - Output $\mathrm{sk}_{f,\tau}$.

- $\mathsf{HEval}(\mathrm{pk}, \mathrm{sk}_{f,\tau}, \tau, f, \overrightarrow{\sigma}, \overrightarrow{m})$:
  - Parse pk as (dpk, crs);
  - $x \Leftarrow (\tau, f, f(\overrightarrow{m}), \mathrm{dpk})$;
  - $w \Leftarrow (\overrightarrow{m}, \overrightarrow{\sigma}, \mathrm{sk}_{f,\tau})$;
  - $\sigma_h \Leftarrow \Pi_{\text{zk-SNARK}}.\mathsf{Prove}(\mathrm{crs}, x, w)$;
  - Output σ_h.

Remark 1. A zk-SNARK system can be used to verify the correctness of a specific computation efficiently. Here we encode this specific computation as

$$C(\{x_i\}_{i=1}^k, \{y_i\}_{i=1}^k, z) \overset{\text{def}}{=} f(\overrightarrow{x}) \wedge \mathcal{DS}.\mathsf{Verify}_{\mathrm{dpk}}(f\|\tau, z) \wedge \prod_{i=1}^{k} \mathcal{DS}.\mathsf{Verify}_{\mathrm{dpk}}(\tau\|x_i\|i, y_i),$$

where $(\{x_i\}_{i=1}^k, \{y_i\}_{i=1}^k, z)$ are the inputs of C and $\hat{m}$ is the output. The task of the cloud is to persuade the verifier that there is indeed a witness $\omega = (\{m_i\}_{i=1}^k, \{\sigma_i\}_{i=1}^k, \mathrm{sk}_{f,\tau})$ such that $C(\{m_i\}_{i=1}^k, \{\sigma_i\}_{i=1}^k, \mathrm{sk}_{f,\tau}) = \hat{m}$.

6.2. Security analysis

Theorem 6.1. *If $\mathcal{DS}$ is an adaptively CMA-secure* DS *scheme, and $\Pi_{\text{zk-SNARK}}$ is an adaptive zk-SNARK system, our construction above is unforgeable.*

The intuition behind is that a valid forgery provided by $\mathcal{A}$ implies that $\mathcal{C}$ has never signed $f^*\|\tau^*$ or $\overrightarrow{m}^*$. Since $\Pi_{\text{zk-SNARK}}$ is an argument of knowledge, we can use the extractor E_{P^*} to recover a valid forgery against $\mathcal{DS}$.

Proof. We use the adversary $\mathcal{A}$ to build an algorithm $\mathcal{A}_{\mathcal{DS}}$ to break $\mathcal{DS}$. $\mathcal{A}_{\mathcal{DS}}$ is given dpk and has access to the oracle $\mathcal{DS}.\mathcal{O}_{\mathsf{Sign}}(\cdot)$. It works as below.

(1) $\mathcal{A}_{\mathcal{DS}}$ runs $(\mathrm{crs}, \mathrm{td}) \Leftarrow \Pi_{\text{zk-SNARK}}.E^1_{P^*}(1^\lambda)$, sets $\mathrm{pk} \Leftarrow (\mathrm{dpk}, \mathrm{crs})$ and invokes $\mathcal{A}$ on input pk.

(2) It answers $\mathcal{A}$'s queries to oracles $\mathcal{O}_{\mathsf{Sign}}$, $\mathcal{O}_{\mathsf{KGen}}$ and $\mathcal{O}_{\mathsf{HEval}}$ as follows.

- $\mathcal{O}_{\mathsf{Sign}}(\tau, m)$:
  - If m is the first query for τ, set $i_\tau = 1$, output $\sigma_i \Leftarrow \mathcal{DS}.\mathcal{O}_{\mathsf{Sign}}(\tau\|m\|i_\tau)$.
  - If $i_\tau = k$, abort.
  - Otherwise, increase the counter i_τ by 1, and output the corresponding signature σ_i.

- $\mathcal{O}_{\mathsf{KGen}}(f, \tau)$: Simply output $\mathrm{sk}_{f,\tau} \Leftarrow \mathcal{DS}.\mathcal{O}_{\mathsf{Sign}}(f\|\tau)$.
- $\mathcal{O}_{\mathsf{HEval}}(\tau, f, \overrightarrow{\sigma}, \overrightarrow{m})$:
  - If $(f, \tau, \mathrm{sk}_{f,\tau})$ can be found in the database, output $\hat{\sigma} \Leftarrow \Pi_{\text{zk-SNARK}}.\mathsf{Prove}(\mathrm{crs}, \iota, \omega)$ where $\iota = (\tau, f, f(\overrightarrow{m}), \mathrm{dpk})$ and $\omega = (\overrightarrow{m}, \overrightarrow{\sigma}, \mathrm{sk}_{f,\tau})$.
  - Otherwise, submit the tuple (f, τ) to $\mathcal{FS}.\mathcal{O}_{\mathsf{Key}}(\cdot)$ to get a key $\mathrm{sk}_{f,\tau}$, and output $\hat{\sigma}$ as above.

(3) Finally, $\mathcal{A}$ outputs a forgery $(\tau^*, \hat{m}^*, \hat{\sigma}^*, f^*)$. Utilize $\Pi_{\text{zk-SNARK}}.E^2_{P^*}(\mathrm{crs}, \mathrm{td}, \hat{m}^*, \hat{\sigma}^*)$ to extract a witness $\omega^* = (\overrightarrow{m}^*, \overrightarrow{\sigma}^*, \sigma_f^*)$.

If $\mathcal{A}$ forges a valid tuple $(\tau^*, \hat{m}^*, \hat{\sigma}^*, f^*)$, we have $\hat{m}^* \in \mathrm{range}(f^*)$. Otherwise, we can construct another $\mathcal{A}_\Pi$ to break the special soundness property of $\Pi_{\text{zk-SNARK}}$ (i.e. if $\hat{m}^* \notin \mathrm{range}(f^*)$, $\hat{m}^* \notin L$)).

Since the crs generated by $\Pi_{\text{zk-SNARK}}.\mathsf{Setup}(1^\lambda)$ and that generated by $\Pi_{\text{zk-SNARK}}.E^1_{P^*}(1^\lambda)$ are statistically close, the view of $\mathcal{A}$ in the game above is identically distributed to that in a real attack against our AFHS scheme. If $\mathcal{A}$ outputs a Type_1 forgery (i.e. $f^*\|\tau^* \notin \{f\|\tau\}_{Query}$), $\mathcal{A}_{\mathcal{DS}}$ outputs $(f^*\|\tau^*, \sigma_f^*)$ as its own forgery. If $\mathcal{A}$ outputs a Type_2 forgery (i.e. $\tau^* \notin \{\tau\}_{Query}$), $\mathcal{A}_{\mathcal{DS}}$ outputs $(\tau^*\|m_1^*\|1, \sigma_1^*)$ as it own forgery. If $\mathcal{A}$ output a Type_3 forgery (i.e. $\hat{m} \notin \{f^*(\overrightarrow{m})\}_{Query}$), $\mathcal{A}_{\mathcal{DS}}$ can also find a forgery $(\tau^*\|m_i^*\|i, \sigma_i^*)$ at some i since $\overrightarrow{m}^* \notin \{\overrightarrow{m}\}_{Query}$. Guaranteed by the unforgeability of $\mathcal{DS}$, we can draw a conclusion that $\Pr[\mathsf{Forge}_1]$, $\Pr[\mathsf{Forge}_2]$ and $\Pr[\mathsf{Forge}_3]$ are all negligible and thus the probability that $\mathcal{A}$ breaks the unforgeability of our AFHS scheme is negligible as well. □

Theorem 6.2. *If $\Pi_{\text{zk-SNARK}}$ is an adaptive zk-SNARK system, our construction above is weakly context hiding.*

Proof. Consider the following two games:

Game 1. The real-world AFHS privacy challenge game.

- $\mathcal{C}$ generates $\mathrm{crs} \Leftarrow \Pi_{\text{zk-SNARK}}.\mathsf{Setup}(1^\lambda)$ and $(\mathrm{dsk}, \mathrm{dpk}) \Leftarrow \mathcal{DS}.\mathsf{Setup}(1^\lambda)$ and invokes $\mathcal{A}$ on input ((dpk, crs), dsk).

- $\mathcal{A}$ submits $(\overrightarrow{m_0}, \overrightarrow{m_1}, f)$ for which $\hat{m} = f(\overrightarrow{m_0}) = f(\overrightarrow{m_1})$.
- $\mathcal{C}$ flips a coin $b \leftarrow \{0, 1\}$, computes $\overrightarrow{\sigma} \Leftarrow \mathcal{DS}.\mathsf{Sign}(\mathrm{dsk}, \overrightarrow{m_b}, \tau)$ where $\tau \leftarrow \{0, 1\}^{\lambda}$, and sets $\mathrm{sk}_{f,\tau} \Leftarrow \mathcal{DS}.\mathsf{Sign}(\mathrm{dsk}, f\|\tau)$. Then $\mathcal{C}$ calls $\pi \Leftarrow \Pi_{\text{zk-SNARK}}.\mathsf{Prove}(\mathrm{crs}, x, \omega)$ where $x = (\tau, f, \hat{m}, \mathrm{dpk})$ and $\omega = (\overrightarrow{m_b}, \overrightarrow{\sigma}, \mathrm{sk}_{f,\tau})$, and returns $(\tau, \hat{m}, \pi)$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b' and wins the game if $b' = b$. Denote the event that $\mathcal{A}$ successfully guesses the bit b in this game by X_0.

Game 2. This game is the same as Game 1, except that the common reference string is generated as $(\mathrm{crs}, \mathrm{td}) \Leftarrow \Pi.\mathcal{S}^{\mathrm{crs}}(1^{\lambda})$, and the proof is generated as $\pi \Leftarrow \Pi_{\text{zk-SNARK}}.\mathcal{S}^{\mathsf{Prove}}(\mathrm{crs}, \mathrm{td}, (\tau, f, \hat{m}, \mathrm{dpk}))$. We use the notation X_1 to represent the event that $\mathcal{A}$ successfully guesses the bit b in this game.

As per the adaptive zero-knowledge property of $\Pi_{\text{zk-SNARK}}$, it holds that $|\Pr[X_0] - \Pr[X_1]| \leq \mathrm{negl}(\lambda)$. Furthermore, it is clear that $\Pr[X_1]$ is $1/2$ since π is simulated by $\Pi_{\text{zk-SNARK}}.\mathcal{S}^{\mathsf{Prove}}(\mathrm{crs}, \mathrm{td}, (\tau, f, \hat{m}, \mathrm{dpk}))$ and is independent of the bit b. Therefore, we conclude that $|\Pr[X_0] - 1/2| \leq \mathrm{negl}(\lambda)$. □

6.3. Extension

In our scheme, the data owner seems to have to calculate a new function key $\mathrm{sk}_{f,\tau}$ when she/he wants to authorize a new function for the agency. Actually, the second construction can be extended to support authorizing multiple functions at a time. The extended version works almost as before expect that $\Pi_{\text{zk-SNARK}}$ now is an adaptive zero-knowledge SNARK system for the following new language:

$$\begin{aligned} L = \{(\tau, \{f_1, \ldots, f_n\}, \hat{m}, \mathrm{dpk})\colon &\exists\ (\overrightarrow{m}, \overrightarrow{\sigma}, \sigma_{f_1\|\cdots\|f_n\|\tau}) \\ \text{s.t. } \forall\, i \in [k]\ \ &\mathcal{DS}.\mathsf{Verify}(\mathrm{dpk}, \tau\|m_i\|i, \sigma_i) = 1 \wedge \\ &f_1(\overrightarrow{m})\|\cdots\|f_n(\overrightarrow{m}) = \hat{m} \wedge \\ &\mathcal{DS}.\mathsf{Verify}(\mathrm{dpk}, f_1\|\cdots\|f_n\|\tau, \sigma_{f_1\|\cdots\|f_n\|\tau}) = 1\}. \end{aligned}$$

7. INSTANTIATION DISCUSSION

Both of the two constructions of AFHS are generic. The first construction combines a HS scheme and a function signature scheme. There are known concrete constructions of HS, e.g. [4, 6–8, 9, 10], however, to the best of our knowledge, there is no concrete and efficient construction of FS in the literature.

Regarding the second construction, which makes use of a standard signature scheme and a zk-SNARK argument system. There are plenty of efficient instantiations of standard signature in the literature. However, the constructions of zk-SNARK are complex. Usually, we need to transform the statement to be proved to an arithmetic circuit satisfiability problem instance, and then invoke the zk-SNARK system to produce the proof. The process of proof generation is time costly, but the verification could be very fast. To the best of our knowledge, one of the most efficient construction of zk-SNARK is due to Groth [22], in which a proof is only three group elements and verification consists of checking a single pairing product equations using three pairings in total. We leave it as one of our future works to design concrete and efficient constructions of our AFHS schemes.

8. CONCLUSION

In this paper, we introduced the notion of authorized function HS, which enables the data owner to control the behavior of the agency and reduces the possibility of dispute. To demonstrate it, we proposed a generic construction of AFHS, which is based on functional signature and traditional HS and satisfies weak unforgeability and weak privacy. We then provided another construction of AFHS based on zk-SNARKs, which strengthens the unforgeability. As another future work, we plan to study how to enhance the privacy in AFHS and introduce the delegatability into AFHS, e.g. allowing the agency to delegate its computation privilege to another one.

FUNDING

This work was supported by the National Natural Science Foundation of China [61472146, 61872152]; Guangdong Natural Science Funds for Distinguished Young Scholar [2014A030306021]; Guangdong Program for Special Support of Top-notch Young Professionals [2015TQ01X796]; and Pearl River Nova Program of Guangzhou [201610010037].

ACKNOWLEDGMENTS

We gratefully acknowledge the anonymous reviewers as well as the editor of *The Computer Journal* for their invaluable comments.

REFERENCES

[1] Wei, L., Zhu, H., Cao, Z., Dong, X., Jia, W., Chen, Y. and Vasilakos, A.V. (2014) Security and privacy for storage and computation in cloud computing. *Inf. Sci.*, **258**, 371–386.

[2] Johnson, R., Molnar, D., Song, D. and Wagner, D. (2002) Homomorphic signature schemes. *Cryptographers Track at the*

RSA Conf., San Jose, CA, USA, 18–22 February, pp. 244–262. Springer, Berlin.

[3] Boyle, E., Goldwasser, S. and Ivan, I. (2014) Functional signatures and pseudorandom functions. *Int. Workshop on Public Key Cryptography*, Buenos Aires, Argentina, 26–28 March, pp. 501–519. Springer, Berlin.

[4] Schabhüser, L., Buchmann, J. and Struck, P. (2017) A linearly homomorphic signature scheme from weaker assumptions. *IMA Int. Conf. Cryptography and Coding*, Oxford, UK, 12–14 December, pp. 261–279. Springer, Berlin.

[5] Tsabary, R. (2017) An equivalence between attribute-based signatures and homomorphic signatures, and new constructions for both. *Theory of Cryptography Conf.*, Baltimore, MD, USA, 12–15 November, pp. 489–518. Springer, Berlin.

[6] Attrapadung, N., Libert, B. and Peters, T. (2013) Efficient completely context-hiding quotable and linearly homomorphic signatures. *Public Key Cryptography*, Nara, Japan, 26 February–1 March, pp. 386–404. Springer, Berlin.

[7] Freeman, D.M. (2012) Improved security for linearly homomorphic signatures: A generic framework. *Public Key Cryptography*, Darmstadt, Germany, 21–23 May, pp. 697–714. Springer, Berlin.

[8] Libert, B., Peters, T., Joye, M. and Yung, M. (2015) Linearly homomorphic structure-preserving signatures and their applications. *Designs Codes Cryptogr.*, **77**, 441–477.

[9] Boneh, D. and Freeman, D.M. (2011) Homomorphic signatures for polynomial functions. *Annual Int. Conf. Theory and Applications of Cryptographic Techniques*, Tallinn, Estonia, 15–19 May, pp. 149–168. Springer, Berlin.

[10] Catalano, D., Fiore, D. and Warinschi, B. (2014) Homomorphic signatures with efficient verification for polynomial functions. *Int. Cryptology Conf.*, Santa Barbara, CA, USA, 17–21 August, pp. 371–389. Springer, Berlin.

[11] Gorbunov, S., Vaikuntanathan, V. and Wichs, D. (2015) Leveled fully homomorphic signatures from standard lattices. *Proc. Forty-Seventh Annual ACM Symposium on Theory of Computing*, Portland, OR, USA, 14–17 June, pp. 469–477. ACM, New York, NY, USA.

[12] Lai, R.W., Tai, R.K., Wong, H.W. and Chow, S.S. (2016) A zoo of homomorphic signatures: multi-key and key-homomorphism. *IACR Cryptology ePrint Archive*, **2016**, 834.

[13] Backes, M., Meiser, S. and Schröder, D. (2016) Delegatable functional signatures. *Public Key Cryptography*, Taipei, Taiwan, 6–9 March, pp. 357–386. Springer, Berlin.

[14] Benabbas, S., Gennaro, R. and Vahlis, Y. (2011) Verifiable delegation of computation over large datasets. *Annual Cryptology Conf.*, Santa Barbara, CA, USA, 14–18 August, pp. 111–131. Springer, Berlin.

[15] Demirel, D., Schabhüser, L. and Buchmann, J.A. (2017) *Privately and Publicly Verifiable Computing Techniques-A Survey*. Springer, Berlin.

[16] Gennaro, R., Gentry, C. and Parno, B. (2010) Non-interactive verifiable computing: Outsourcing computation to untrusted workers. *Annual Cryptology Conf.*, Santa Barbara, CA, USA, 15–19 August, pp. 465–482. Springer, Berlin.

[17] Parno, B., Raykova, M. and Vaikuntanathan, V. (2012) How to delegate and verify in public: Verifiable computation from attribute-based encryption. *Theory of Cryptography Conf.*, Taormina, Sicily, Italy, 19–21 March, pp. 422–439. Springer, Berlin.

[18] Li, M., Ruan, N., Qian, Q., Zhu, H., Liang, X. and Yu, L. (2017) SPFM: scalable and privacy-preserving friend matching in mobile cloud. *IEEE Internet Things J.*, **4**, 583–591.

[19] Attrapadung, N., Libert, B. and Peters, T. (2012) Computing on authenticated data: New privacy definitions and constructions. *Int. Conf. Theory and Application of Cryptology and Information Security*, Beijing, China, 2–6 December, pp. 367–385. Springer, Berlin.

[20] Fleischhacker, N., Krupp, J., Malavolta, G., Schneider, J., Schröder, D. and Simkin, M. (2016) Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. *Public Key Cryptography*, Taipei, Taiwan, 6–9 March, pp. 301–330. Springer, Berlin.

[21] Gennaro, R., Gentry, C., Parno, B. and Raykova, M. (2013) Quadratic span programs and succinct NIZKs without PCPs. *Annual Int. Conf. Theory and Applications of Cryptographic Techniques*, Athens, Greece, 26–30 May, pp. 626–645. Springer, Berlin.

[22] Groth, J. (2016) On the size of pairing-based non-interactive arguments. *Advances in Cryptology—EUROCRYPT 2016—35th Annual Int. Conf. Theory and Applications of Cryptographic Techniques, Proc., Part II*, Vienna, Austria, May 8–12, Lecture Notes in Computer Science, 9666, pp. 305–326. Springer.