SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

December 2021, Vol. 64 222301:1–222301:13
https://doi.org/10.1007/s11432-019-2855-3

Functional signatures: new definition and constructions

Qingwen GUO[1], Qiong HUANG[1,2*], Sha MA[1], Meiyan XIAO[1], Guomin YANG[3] & Willy SUSILO[3]

[1]*College of Mathematics and Informatics, South China Agricultural University, Guangzhou* 510642*, China;*
[2]*Guangzhou Key Laboratory of Intelligent Agriculture, South China Agricultural University, Guangzhou* 510642*, China;*
[3]*School of Computing and Information Technology, University of Wollongong, Wollongong* NSW 2522*, Australia*

Received 2 November 2019/Revised 18 January 2020/Accepted 18 March 2020/Published online 27 October 2021

Abstract Functional signatures (FS) enable a master authority to delegate its signing privilege to an assistant. Concretely, the master authority uses its secret key sk_F to issue a signing key sk_f for a designated function $f \in \mathcal{F}_{\mathsf{FS}}$ and sends both f and sk_f to the assistant $\mathcal{E}$, which is then able to compute a signature σ_f with respect to pk_F for a message y in the range of f. In this paper, we modify the syntax of FS slightly to support the application scenario where a certificate of authorization is necessary. Compared with the original FS, our definition requires that $\mathcal{F}_{\mathsf{FS}}$ is an injective function family and for any $f_0, f_1 \in \mathcal{F}_{\mathsf{FS}}$ there does not exist an intersection between $\mathsf{range}(f_0)$ and $\mathsf{range}(f_1)$. Accordingly, we redefine the security of FS and introduce two additional security notions, called unlinkability and accountability. Signatures σ_f in our definition do not expose the intention of the master authority. We propose two constructions of FS. The first one is a generic construction based on signatures with perfectly re-randomizable keys, non-interactive zero-knowledge proof (NIZK) and traditional digital signatures, and the other is based on RSA (Rivest-Shamir-Adleman) signatures with full domain hash and NIZK. We prove that both schemes are secure under the given security models.

Keywords cloud computation security, digital signature, functional signature, non-interactive zero-knowledge proof, e-commerce

Citation Guo Q W, Huang Q, Ma S, et al. Functional signatures: new definition and constructions. Sci China Inf Sci, 2021, 64(12): 222301, https://doi.org/10.1007/s11432-019-2855-3

1 Introduction

Nowadays cloud computing is profoundly influencing the development of modern economy and society. However, the cloud is not a completely reliable service platform and we should take into consideration cloud storage security and cloud computation security [1]. In this paper, we merely focus on how to achieve cloud computation security and try to provide an alternative approach to prevent a cloud server from performing infected computations.

Boyle et al. [2] have proposed a novel notion called functional signature (FS). FS enables a master authority to delegate the signing process to its assistant $\mathcal{E}$ if $\mathcal{E}$ honestly carries out the specified actions. Therefore, FS can be well applied to scenarios where authorization letters are necessary. Take the scenario shown in Figure 1 as an example. Assume that Alice asks $\mathcal{E}$ (e.g., Bob) for help to sell her fruits. Since the last-sale price may fluctuate according to the order quantity, Alice sets a price range in advance, and issues an authorization letter for $\mathcal{E}$ (e.g., Figure 2). Then $\mathcal{E}$ is able to show the latest price information to the customers on behalf of Alice. Concretely, in FS the master authority issues a signing key sk_f for a designated function f with the master secret key sk_F (with respect to pk_F), i.e., $\mathrm{sk}_f \leftarrow \mathsf{FS.KGen}(\mathrm{sk}_F, f)$ and sends (sk_f, f) to $\mathcal{E}$. If the cloud $\mathcal{E}$ honestly performs the f on an input x (which satisfies the appointed complex policy), $\mathcal{E}$ can compute a desirable signature $\sigma_f \leftarrow \mathsf{FS.Sign}(\mathrm{sk}_f, f, x)$ on the output $y = f(x)$. Finally, a receiver checks whether y is admitted by the master authority, i.e., $b \leftarrow \mathsf{FS.Vrfy}(\mathrm{pk}_F, y, \sigma_f)$.

* Corresponding author (email: qhuang@scau.edu.cn)

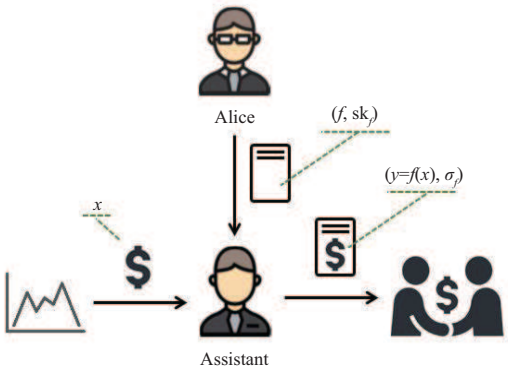

Figure 1 (Color online) An application scenario of FS.

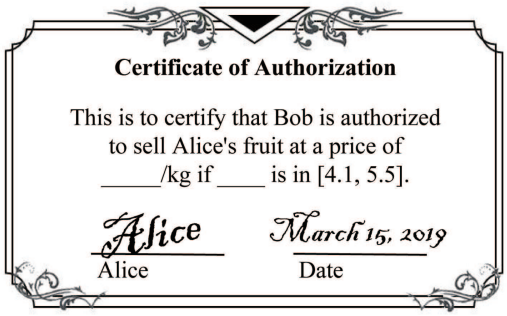

Figure 2 An example of authorization letter.

Unlike [2], this paper assumes that $\mathsf{range}(f_0) \cap \mathsf{range}(f_1) = \emptyset$ for any $f_0, f_1 \in \mathcal{F}_{\mathsf{FS}}$ and $f(x_0) \neq f(x_1)$ for any $x_0, x_1 \in \mathcal{D}_f$ since an authorization letter should be unambiguous. Accordingly, we redefine FS and modify its security models slightly.

A secure FS scheme requires unforgeability and privacy. Unforgeability guarantees that without the knowledge of sk_f (with respect to f) the cloud server cannot deceive the verifier into accepting a forged message $y^* \in \mathsf{range}(f)$. Regarding privacy, Boyle et al. [2] introduced a notion named function privacy, which guarantees that an eavesdropper cannot judge whether (y, σ_f) comes from $\mathsf{FS.Sign}(\mathrm{sk}_{f_0}, f_0, x_0)$ or $\mathsf{FS.Sign}(\mathrm{sk}_{f_1}, f_1, x_1)$ if $f_0(x_0) = f_1(x_1)$. Unfortunately, this privacy notion no longer works in the scenario above because there does not exist a tuple (f_0, x_0, f_1, x_1) in $\mathcal{F}_{\mathsf{FS}}$ such that $f_0(x_0) = f_1(x_1)$. Hence, we introduce another privacy notion called unlinkability to close this gap. Informally, unlinkability ensures that a probabilistic polynomial-time (PPT) adversary cannot determine if σ_f is a real signature from the signer (i.e., the assistant) or a simulated one output by the simulation algorithm $\mathsf{FS.Sim}(\mathrm{pk}_F, f, x)$. Therefore, $(y = f(x), \sigma_f)$ does not expose the cooperation intention of the master authority to others except the verifier. A potential application of unlinkable FS is business negotiation. Suppose that $\mathcal{E}$ is an agency and $\mathcal{B}$ is a potential business partner of the master authority. The master authority issues an authorization letter (sk_f, f) to $\mathcal{E}$ and asks $\mathcal{E}$ to enter into a contract with $\mathcal{B}$ on its behalf. $\mathcal{E}$ draws up a draft contract $y = f(x)$ in accordance with specific conditions and produces σ_f on y to show the cooperation intention of the master authority with $\mathcal{B}$. If $\mathcal{B}$ accepts the draft contract, then they run a fair exchange protocol [3–5] to sign the final cooperation intention agreement. Otherwise, $\mathcal{E}$ and $\mathcal{B}$ obtain nothing from each other. Notice that in the above process the master authority cannot deny the fact that it did issue an authorization letter to $\mathcal{E}$ when $\mathcal{E}$ submits (sk_f, f) to the judge if there is any dispute. So the output of the fair exchange protocol and the (sk_f, f) can be used to constitute an undeniable contract between the master authority and $\mathcal{B}$.

Our contributions. In this work, we make the following contributions.

(1) We modify the definition of FS [2] slightly in order to support the application scenarios where authorization letters are necessary. Preciously, in our paper $\mathcal{F}_{\mathsf{FS}}$ is a special function family such that there does not exist a pair (f_0, x_0, f_1, x_1) in $\mathcal{F}_{\mathsf{FS}}$ with $f_0(x_0) = f_1(x_1)$ since authorization letters are unambiguous. Accordingly, we revise the security of FS and introduce two security notions: unlinkability and accountability. The unlinkability guarantees that an adversary cannot identify the source of $(\hat{m}, \sigma_f)$, and the accountability guarantees that the master authority cannot deny the fact that it did delegate its signing privilege to $\mathcal{E}$ (c.f. Definitions 6 and 7).

(2) Inspired by the constructions in [2], we provide two constructions with respect to our definition of FS. The first one takes advantage of signatures with perfectly re-randomizable keys (c.f. [6]), non-interactive zero-knowledge proof and digital signatures, and the second one is based on the RSA-FDH signature scheme [7] and zero-knowledge proof [8]. We prove them to be secure under the proposed security models.

Organization. Section 2 reviews some related studies in the literature. Section 3 recalls some cryptographic primitives that are important in our constructions. Section 4 gives the revised definition of FS as well as the corresponding security models. Section 5 gives a generic construction of FS, and Section 6 gives our second construction of FS. Finally, we conclude this work in Section 7.

2 Related work

Taking into account the cloud computation security, researchers have proposed homomorphic signatures [9–12], redactable signatures [13, 14], sanitizable signatures [6, 15], and proxy signatures [16, 17]. These primitives are closely related to FS, and in this part we discuss the relations among them.

Homomorphic signature (HS) is a cryptographic primitive which allows any entity to verify the origin of the calculation result $\hat{m}$. Suppose that a data owner $\mathcal{H}$ wants to outsource its data $\boldsymbol{m} = \{m_1, \ldots, m_k\}$ to a cloud server $\mathcal{E}$, and it runs $\sigma_i \leftarrow \mathsf{HS.Sign}(\mathrm{sk}_H, m_i)$ to generate a vector of signatures $\boldsymbol{\sigma}$ which can be used to authenticate $\boldsymbol{m}$. After that $\mathcal{E}$ is able to compute a homomorphic signature σ_h on $\hat{m} = f(\boldsymbol{m})$ by running $\sigma_h \leftarrow \mathsf{HS.Eval}(\mathrm{pk}_H, f, \cdot, \boldsymbol{\sigma})$ without knowing sk_H. Specially, only if the input of f is $\boldsymbol{m}$ from $\mathcal{H}$ and the output is $\hat{m}$, does it hold that $\mathsf{HS.Vrfy}(\mathrm{pk}_H, f, \cdot, \hat{m}, \sigma_h) = 1$. Since the context-hiding property of HS guarantees that σ_h does not leak information about the input $\boldsymbol{m}$ to the verifier, linearly HS [9, 11, 12] enjoys wide application prospects (e.g., $f(\mathrm{weith}, \mathrm{age}) = 0.2 \cdot \mathrm{weight} - 0.2 \cdot \mathrm{age}$ can be used to calculate the bone density). Though σ_h enables the third party to believe that $\mathcal{E}$ did perform computations on $\mathcal{H}$'s data, it does not authenticate the function f and the inputs are fixed.

Redactable signature (RS) is another primitive which can be seen as a special type of HS. In RS, a data owner $\mathcal{R}$ executes the signing algorithm to produce a malleable signature $\ddot{\sigma}$ on a designated document $\ddot{m}$ (e.g., an electronic medical record) and uploads $(\ddot{m}, \ddot{\sigma})$ to $\mathcal{E}$. Then $\mathcal{E}$ (i.e., the redactor) removes sensitive portions of $\ddot{m}$ to get a disclosed data $\dot{m}$ and computes a redactable signature $\dot{\sigma}$ on behalf of $\mathcal{R}$ accordingly. Ma et al. [13] made a survey on RS and summarized the existing works: RS based on commit vector, RS based on Merkle hash, RS based on bilinear pairing, and RS based on the accumulator. There is no doubt that RS is useful. However, RS cannot stop a signature holder from modifying $\dot{m}$. In other words, an eavesdropper can further delete some portions of $\dot{m}$ to get $\hat{m}$ and compute a valid redactable signature $\hat{\sigma}$ on $\hat{m}$ (c.f. [14]). Therefore, RS is not suitable for our application scenario.

The functionality of sanitizable signature (SS) is analogous to RS. But unlike RS, the system of SS should maintain a key pair $(\mathrm{sk}_{\mathcal{S}}, \mathrm{pk}_{\mathcal{S}})$ (of data owner $\mathcal{S}$) and a key pair $(\mathrm{sk}_{\mathcal{E}}, \mathrm{pk}_{\mathcal{E}})$ (of sanitizer $\mathcal{E}$) simultaneously. Fleischhacker et al. [6] proposed an unlinkable SS scheme using signature with re-randomizable keys, CCA-secure public-key encryption and non-interactive zero-knowledge proof. Their instantiation is concrete and efficient. Camenisch et al. [15] proposed another invisible SS scheme based on an interesting primitive dubbed chameleon-hash functions with ephemeral trapdoors. FS can be considered as a variant of SS, but there is no need for FS to maintain the key pair $(\mathrm{sk}_{\mathcal{E}}, \mathrm{pk}_{\mathcal{E}})$. Meanwhile, FS does not require the transparency property [6].

Boyle et al. [2] formalized the definition of FS, and showed an application of FS in verifiable computation (VC, c.f. [18]). Afterwards, Backes et al. [19] introduced a more malleable primitive named delegatable functional signature (DFS). In DFS, a master authority can outsource the computation $\mathcal{F}$ to $\mathcal{E}$ and $\mathcal{E}$ can further delegate the computation $\mathcal{F}' \subseteq \mathcal{F}$ to its sub-contractor. To enhance the security of FS, Li et al. [20] proposed private FS where the master authority can transform an input x into a ciphertext c_x in order to keep $\mathcal{E}$ away from the firsthand data. Recently, Guo et al. [21] introduced a new primitive, called authorized function homomorphic signature (AFHS), which can be used to check whether the cloud server $\mathcal{E}$ honestly performs a specified function on user's data. Their black-box construction is based on FS.

FS also shares similarity with proxy signature (PS) [16] which allows the delegation of signing rights. In PS, the master authority signs on a warrant w, which is composed of a message part and a freshly generated public key pk'_P, and gives w and the private key sk'_P (with respect to pk'_R) to $\mathcal{E}$. In fact, the idea of the first trivial FS construction in [2] is similar to PS.

3 Preliminaries

3.1 Signatures with perfectly re-randomizable keys

A signature scheme with re-randomizable keys allows a signer to re-randomize a key pair $(\mathrm{sk}_R, \mathrm{pk}_R)$ to a fresh key pair $(\mathrm{sk}'_R, \mathrm{pk}'_R)$ and then prove the origin of $(\mathrm{sk}'_R, \mathrm{pk}'_R)$ (using zero-knowledge proof). In addition, the signer can sign on a message m and generate a signature σ_r with respect to pk_R as well as another signature $\widetilde{\sigma_r}$ with respect to pk'_R.

Definition 1 (Signature with perfectly re-randomizable keys, RRS [6]). An RRS scheme consists of a tuple of PPT algorithms $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{RrSk}, \mathsf{RrPk})$, as follows.

$\mathsf{Setup}(1^\lambda) \to (\mathrm{sk}_R, \mathrm{pk}_R)$. The setup algorithm takes as input a security parameter 1^λ, and outputs a secret key sk_R and a public key pk_R for the signer.

$\mathsf{Sign}(\mathrm{sk}_R, m) \to \sigma_r$. The signing algorithm takes as input sk_R and a designated message $m \in \mathcal{M}_{\mathsf{RRS}}$ (where $\mathcal{M}_{\mathsf{RRS}}$ is the message space), and outputs a signature σ_r.

$\mathsf{Vrfy}(\mathrm{pk}_R, m, \sigma_r) \to b$. The verification algorithm takes as input pk_R and the alleged pair (m, σ_r), and outputs a bit b (i.e., boolean variable).

$\mathsf{RrSk}(\mathrm{sk}_R, \delta) \to \mathrm{sk}'_R$. The secret key re-randomization algorithm takes as input sk_R and a randomness $\delta \in \Delta_{\mathsf{RRS}}$ (where Δ_{RRS} is the randomness space), and outputs a fresh secret key sk'_R.

$\mathsf{RrPk}(\mathrm{pk}_R, \delta) \to \mathrm{pk}'_R$. The public key re-randomization algorithm takes as input pk_R and δ, and outputs a fresh public key pk'_R (with respect to sk'_R).

Correctness. A necessary property of RRS is correctness [6]. Formally,

(1) For any $(\mathrm{sk}_R, \mathrm{pk}_R) \in \{\mathsf{RRS.Setup}(1^\lambda)\}$ and any $m \in \mathcal{M}_{\mathsf{RRS}}$, it holds that

$$\Pr\left[\sigma_r \leftarrow \mathsf{RRS.Sign}(\mathrm{sk}_R, m) : \mathsf{RRS.Vrfy}(\mathrm{pk}_R, m, \sigma_r) \to 0\right] \leqslant \mathsf{negl}(\lambda);$$

(2) For any $(\mathrm{sk}_R, \mathrm{pk}_R) \in \{\mathsf{RRS.Setup}(1^\lambda)\}$, any $m \in \mathcal{M}_{\mathsf{RRS}}$ and any $\delta \in \Delta_{\mathsf{RRS}}$, it holds that

$$\begin{aligned}\Pr\big[\mathrm{sk}'_R \leftarrow \mathsf{RRS.RrSk}(\mathrm{sk}_R, \delta) \wedge \sigma_r \leftarrow \mathsf{RRS.Sign}(\mathrm{sk}'_R, m)\\ \wedge\, \mathrm{pk}'_R \leftarrow \mathsf{RRS.RrPk}(\mathrm{pk}_R, \delta) : \mathsf{RRS.Vrfy}(\mathrm{pk}'_R, m, \sigma_r) \to 0\big] \leqslant \mathsf{negl}(\lambda);\end{aligned}$$

(3) If $(\mathrm{sk}_R, \mathrm{pk}_R)$ and $(\mathrm{sk}''_R, \mathrm{pk}''_R)$ are honestly generated by algorithm $\mathsf{RRS.Setup}$, for a $\delta \leftarrow \Delta_{\mathsf{RRS}}$ it holds that $(\mathrm{sk}''_R, \mathrm{pk}''_R) \overset{c}{\approx} (\mathrm{sk}'_R, \mathrm{pk}'_R)$ where $\mathrm{sk}'_R \leftarrow \mathsf{RRS.RrSk}(\mathrm{sk}_R, \delta)$ and $\mathrm{pk}'_R \leftarrow \mathsf{RRS.RrPk}(\mathrm{pk}_R, \delta)$.

Unforgeability. Consider the following game where $\mathcal{C}_{\mathsf{RRS}}$ is a challenger and $\mathcal{A}_{\mathsf{RRS}}$ is a PPT adversary.

(1) $\mathcal{C}_{\mathsf{RRS}}$ runs $(\mathrm{sk}_R, \mathrm{pk}_R) \leftarrow \mathsf{RRS.Setup}(1^\lambda)$, gives pk_R to $\mathcal{A}_{\mathsf{RRS}}$ and initializes an empty set Q_{RRS}.

(2) $\mathcal{A}_{\mathsf{RRS}}$ has access to two signing oracles $\mathsf{RRS}.\mathcal{O}^1_{\mathsf{Sign}}$ and $\mathsf{RRS}.\mathcal{O}^2_{\mathsf{Sign}}$. More precisely, $\mathcal{A}_{\mathsf{RRS}}$ adaptively makes signature queries on messages of its choices, and $\mathcal{C}_{\mathsf{RRS}}$ responds to these queries as follows:

- $\mathsf{RRS}.\mathcal{O}^1_{\mathsf{Sign}}$: Given m as input, compute $\sigma_r \leftarrow \mathsf{RRS.Sign}(\mathrm{sk}_R, m)$, output σ_r and set $Q_{\mathsf{RRS}} := Q_{\mathsf{RRS}} \cup \{m\}$.
- $\mathsf{RRS}.\mathcal{O}^2_{\mathsf{Sign}}$: Given (m, δ) as input, compute $\mathrm{sk}'_R \leftarrow \mathsf{RRS.RrSk}(\mathrm{sk}_R, \delta)$, output $\sigma_r \leftarrow \mathsf{RRS.Sign}(\mathrm{sk}'_R, m)$ and set $Q_{\mathsf{RRS}} := Q_{\mathsf{RRS}} \cup \{m\}$.

(3) $\mathcal{A}_{\mathsf{RRS}}$ outputs a forgery $(m^*, \sigma^*_r, \delta^*)$ and wins the game if $m^* \notin Q_{\mathsf{RRS}}$ and either of the following holds:

- $\mathsf{RRS.Vrfy}(\mathrm{pk}_R, m^*, \sigma^*_r) = 1$; or
- $\mathsf{RRS.Vrfy}\big(\mathsf{RRS.RrPk}(\mathrm{pk}_R, \delta^*), m^*, \sigma^*_r\big) = 1$.

The advantage $\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{RRS}}(\lambda)$ of $\mathcal{A}_{\mathsf{RRS}}$ is defined as the probability of winning the game.

Definition 2 (Unforgeability of RRS [6]). An RRS scheme $\mathsf{RRS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{RrSk}, \mathsf{RrPk})$ is unforgeable if $\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{RRS}}(\lambda)$ is negligible.

3.2 Non-interactive zero-knowledge proof

With a non-interactive zero-knowledge proof system with respect to language $L \overset{\mathrm{def}}{=} \{\iota : \exists\, \omega \text{s.t.} \mathsf{Check}(\iota, \omega) = 1\}$, a prover $\mathcal{P}$ holding a witness ω is able to convince the verifier $\mathcal{V}$ that the claimed statement ι is true. The zero-knowledge property guarantees that a malicious verifier $\mathcal{V}^*$ cannot learn information about ω from the transcript during the process, and the argument of knowledge property guarantees that a malicious prover $\mathcal{P}^*$ without the corresponding witness cannot offer a correct proof π^*.

Definition 3 (Non-interactive zero-knowledge argument of knowledge, NIZK [22]). An NIZK system for a language $L \in NP$ consists of six PPT algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{Check}, \mathsf{Sim}, \mathsf{Ext})$ and satisfies completeness, adaptive soundness, adaptive zero-knowledge and argument of knowledge.

$\mathsf{Setup}(1^\lambda) \to \mathrm{crs}$. The setup algorithm takes as input a security parameter 1^λ, and outputs a common reference string crs.

$\mathsf{Prove}(\mathrm{crs}, \iota, \omega) \to \pi$. The proving algorithm takes as input crs, a statement $\iota \in L$ and a witness ω for which $\mathsf{Check}(\iota, \omega) = 1$, and outputs a proof π.

$\mathsf{Vrfy}(\mathrm{crs}, \iota, \pi) \to b$. The verification algorithm takes as input crs, the claimed pair (ι, π), and outputs a bit b.

$\mathsf{Check}(\iota,\omega)\to b$. The checking algorithm takes as input ι and ω, and output a bit b.

Sim. The simulator is divided into two parts: (1) $\mathsf{Sim}^1(1^\lambda)\to(\mathrm{crs},\mathrm{td})$ and (2) $\mathsf{Sim}^2(\mathrm{crs},\mathrm{td},\iota)\to\pi_s$. Algorithm Sim^1 takes as input 1^λ, and outputs a simulated crs and a trapdoor td. Algorithm Sim^2 takes as input crs, td and ι, and outputs a simulated proof π_s.

Ext. The extractor is divided into two parts: (1) $\mathsf{Ext}^1(1^\lambda)\to(\mathrm{crs},\mathrm{td})$ and (2) $\mathsf{Ext}^2(\mathrm{crs},\mathrm{td},\iota,\pi)\to\omega$. Algorithm Ext^1 takes as input 1^λ, and outputs a crs and a relevant trapdoor td. Algorithm Ext^2 takes as input crs, td, a claim statement ι and a corresponding proof π, and outputs a witness ω.

Completeness. It requires that

$$\Pr\big[\mathrm{crs}\leftarrow\mathsf{NIZK.Setup}(1^\lambda)\wedge\mathsf{NIZK.Check}(\iota,\omega)=1\wedge\pi\leftarrow\mathsf{NIZK.Prove}(\mathrm{crs},\iota,\omega):\mathsf{NIZK.Vrfy}(\mathrm{crs},\iota,\pi)\to 0\big]\leqslant\mathsf{negl}(\lambda).$$

Adaptive soundness. It requires that

$$\Pr\big[\mathrm{crs}\leftarrow\mathsf{NIZK.Setup}(1^\lambda)\wedge\mathcal{A}_{\mathcal{P}^*}(\mathrm{crs})\to(\iota^*,\pi^*)\wedge\iota^*\notin L:\mathsf{NIZK.Vrfy}(\mathrm{crs},\iota^*,\pi^*)\to 1\big]\leqslant\mathsf{negl}(\lambda).$$

Adaptive zero-knowledge. It requires that

$$\big|\Pr\big[\mathrm{crs}\leftarrow\mathsf{NIZK.Setup}(1^\lambda):\mathcal{A}_{\mathcal{V}^*}{}^{\mathsf{NIZK.Prove}(\mathrm{crs},\iota,\omega)}(\mathrm{crs})\to 1\big]-\Pr\big[(\mathrm{crs},\mathrm{td})\leftarrow\mathsf{NIZK.Sim}^1(1^\lambda):\mathcal{A}_{\mathcal{V}^*}^{\mathsf{NIZK.Sim}^2(\mathrm{crs},\mathrm{td},\iota)}(\mathrm{crs})\to 1\big]\big|\leqslant\mathsf{negl}(\lambda).$$

Argument of knowledge. It requires that

$$\big|\Pr\big[\mathrm{crs}\leftarrow\mathsf{NIZK.Setup}(1^\lambda):\mathcal{A}_{\mathcal{P}^*}(\mathrm{crs})\to 1\big]-\Pr\big[(\mathrm{crs},\mathrm{td})\leftarrow\mathsf{NIZK.Ext}^1(1^\lambda):\mathcal{A}_{\mathcal{P}^*}(\mathrm{crs})\to 1\big]\big|\leqslant\mathsf{negl}(\lambda),$$

and

$$\Pr\big[(\mathrm{crs},\mathrm{td})\leftarrow\mathsf{NIZK.Ext}^1(1^\lambda)\wedge\mathcal{A}_{\mathcal{P}^*}(\mathrm{crs})\to(\iota^*,\pi^*)\wedge\mathsf{NIZK.Vrfy}(\mathrm{crs},\iota^*,\pi^*)\to 1\wedge\mathsf{NIZK.Ext}^2(\mathrm{crs},\mathrm{td},\iota^*,\pi^*)\to\omega^*:\mathsf{NIZK.Check}(\iota^*,\omega^*)=0\big]\leqslant\mathsf{negl}(\lambda).$$

4 Functional signature

4.1 Formal definition

Below we present a revised definition of FS.

Definition 4 (Functional signature, FS)**.** An FS scheme consists of the following five PPT algorithms $(\mathsf{Setup},\mathsf{KGen},\mathsf{RfPk},\mathsf{Sign},\mathsf{Vrfy})$.

$\mathsf{Setup}(1^\lambda)\to(\mathrm{sk}_F,\mathrm{pk}_F)$. The setup algorithm is run by the master authority. It takes as input a security parameter 1^λ and outputs a pair of keys $(\mathrm{sk}_F,\mathrm{pk}_F)$.

$\mathsf{KGen}(\mathrm{sk}_F,f)\to\mathrm{sk}_f$. The signing key generation algorithm is executed by the master authority. It takes as input sk_F and a function f from the function space $\mathcal{F}_{\mathsf{FS}}$, and outputs a signing key sk_f (with respect to f) for the assistant.

$\mathsf{RfPk}(\mathrm{pk}_F)\to\widetilde{\mathrm{pk}_F}$. The public key refreshment algorithm is executed by the verifier. It takes as input pk_F, and outputs a refreshing public key $\widetilde{\mathrm{pk}_F}$ with respect to pk_F. It requires the verifier to maintain a stateful list $\mathcal{L}_{\mathrm{pk}_F}=\{(\mathrm{pk}_F,\widetilde{\mathrm{pk}_F})\}$.

$\mathsf{Sign}(\widetilde{\mathrm{pk}_F},\mathrm{sk}_f,f,x)\to(y,\sigma_f)$. The signing algorithm is run by the assistant. It takes as input $\widetilde{\mathrm{pk}_F}$, sk_f, f and an input x from the domain $\mathcal{D}_f$ of f, and outputs $y=f(x)$ as well as a functional signature σ_f on y. Particularly, the algorithm directly outputs $\perp$ if the origin of $\widetilde{\mathrm{pk}_F}$ is not pk_F.

$\mathsf{Vrfy}(\mathrm{pk}_F,\widetilde{\mathrm{pk}_F},y,\sigma_f)\to b$. The verification algorithm is executed by the verifier. It takes as input $(\mathrm{pk}_F,\widetilde{\mathrm{pk}_F})$ in $\mathcal{L}_{\mathrm{pk}_F}$, y and σ_f, and outputs a bit b, which is 1 for acceptance and 0 for rejection.

Remark 1. In our definition of FS, σ_f can only be verified by a designated verifier who is chosen by the assistant. The verifier runs algorithm RfPk only once for each session and then the assistant can sign on the function output y according to $\widetilde{\mathrm{pk}_F}$ from the verifier. Since algorithm RfPk does not require any secret parameters, the verifier should additionally maintain a list $\mathcal{L}_{\mathrm{pk}_F}=(\mathrm{pk}_F,\widetilde{\mathrm{pk}_F})$ to check whether the assistant has executed algorithm Sign honestly, i.e., using the session-related $\widetilde{\mathrm{pk}_F}$.

Correctness. An FS scheme $\mathsf{FS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{RfPk}, \mathsf{Sign}, \mathsf{Vrfy})$ is correct if it holds that

$$\Pr\big[(\mathrm{sk}_F, \mathrm{pk}_F) \leftarrow \mathsf{FS.Setup}(1^\lambda) \wedge f \in \mathcal{F}_{\mathsf{FS}} \wedge \mathrm{sk}_f \leftarrow \mathsf{FS.KGen}(\mathrm{sk}_F, f) \wedge \widetilde{\mathrm{pk}_F} \leftarrow \mathsf{FS.RfPk}(\mathrm{pk}_F) \\ \wedge x \in \mathcal{D}_f \wedge (y, \sigma_f) \leftarrow \mathsf{FS.Sign}(\widetilde{\mathrm{pk}_F}, \mathrm{sk}_f, f, x) : \mathsf{FS.Vrfy}(\mathrm{pk}_F, \widetilde{\mathrm{pk}_F}, y, \sigma_f) \to 0\big] \leqslant \mathsf{negl}(\lambda).$$

Succinctness. It guarantees that the size of σ_f is independent of the size of (f, x) and demands that

$$(\mathrm{sk}_F, \mathrm{pk}_F) \leftarrow \mathsf{FS.Setup}(1^\lambda) \wedge f \in \mathcal{F}_{\mathsf{FS}} \wedge \mathrm{sk}_f \leftarrow \mathsf{FS.KGen}(\mathrm{sk}_F, f) \wedge \widetilde{\mathrm{pk}_F} \leftarrow \mathsf{FS.RfPk}(\mathrm{pk}_F) \\ \wedge x \in \mathcal{D}_f \wedge (y, \sigma_f) \leftarrow \mathsf{FS.Sign}(\widetilde{\mathrm{pk}_F}, \mathrm{sk}_f, f, x) \wedge |\sigma_f| = \mathsf{poly}(\lambda),$$

where $\mathsf{poly}(\cdot)$ is a polynomial.

4.2 Security models

Unforgeability. Consider the following game where $\mathcal{A}_{\mathsf{FS}}$ is a PPT adversary and $\mathcal{C}_{\mathsf{FS}}$ is a challenger. Due to the unlinkability (Definition 6), the adversary here models an outsider rather than the agency and the (designated) verifier.

(1) $\mathcal{C}_{\mathsf{FS}}$ runs algorithm $\mathsf{FS.Setup}$ to obtain $(\mathrm{sk}_F, \mathrm{pk}_F)$. The pk_F is published while the sk_F is kept secret by $\mathcal{C}_{\mathsf{FS}}$.

(2) $\mathcal{C}_{\mathsf{FS}}$ initializes an empty table $T_{\mathsf{FS}} = (\cdot, \cdot)$ and $\mathcal{A}_{\mathsf{FS}}$ has access to two oracles $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$ and $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ which are defined as follows. For simplicity, we assume that the oracles return the same answer if $\mathcal{A}_{\mathsf{FS}}$ issues a repeated query.

• $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$: It takes as input f, generates a fresh sk_f by running algorithm $\mathsf{FS.KGen}$, inserts an item (f, sk_f) into T_{FS} and returns sk_f as the answer.

• $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$: It takes as input (f, x), and outputs a σ_f and the corresponding $\widetilde{\mathrm{pk}_F}$. Concretely, it works as follows:

(a) If there is (f, sk_f) in T_{FS}, generate $\widetilde{\mathrm{pk}_F} \leftarrow \mathsf{RfPk}(\mathrm{pk}_F)$, compute $(y, \sigma_f) \leftarrow \mathsf{FS.Sign}(\widetilde{\mathrm{pk}_F}, \mathrm{sk}_f, f, x)$, and insert $(\mathrm{pk}_F, \widetilde{\mathrm{pk}_F})$ into a list $\mathcal{L}_{\mathrm{pk}_F}$;

(b) Otherwise, generate a fresh sk_f honestly, add (f, sk_f) into T_{FS}, and compute $\widetilde{\mathrm{pk}_F}$ and σ_f using sk_f as above.

(3) Finally, $\mathcal{A}_{\mathsf{FS}}$ submits a forgery $(\widetilde{\mathrm{pk}_F^*}, y^*, \sigma_f^*)$ to $\mathcal{C}_{\mathsf{FS}}$ and wins the game if it holds that

• $(\mathrm{pk}_F, \widetilde{\mathrm{pk}_F^*}) \in \mathcal{L}_{\mathrm{pk}_F}$; and

• $\mathsf{FS.Vrfy}(\mathrm{pk}_F, \widetilde{\mathrm{pk}_F^*}, y^*, \sigma^*) = 1$; and

• there is no KGen query f such that $y^* \in \mathsf{range}(f)$; and

• a signature on y^* has not been queried before.

The advantage $\mathsf{Adv}_{\mathsf{FS}}^{\mathrm{uf}}(\lambda)$ of $\mathcal{A}_{\mathsf{FS}}$ is defined as the probability of winning the game.

Definition 5 (Unforgeability of FS)**.** An FS scheme $\mathsf{FS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{RfPk}, \mathsf{Sign}, \mathsf{Vrfy})$ is unforgeable if $\mathsf{Adv}_{\mathsf{FS}}^{\mathrm{uf}}(\lambda)$ is negligible.

Unlinkability. Let $\mathsf{FS.Sim}$ denote a simulator. Consider the following game played between a challenger $\mathcal{C}_{\mathsf{FS}}$ and a PPT adversary $\mathcal{A}_{\mathsf{FS}}$.

(1) $\mathcal{C}_{\mathsf{FS}}$ runs $(\mathrm{sk}_F, \mathrm{pk}_F) \leftarrow \mathsf{FS.Setup}(1^\lambda)$ and gives $(\mathrm{sk}_F, \mathrm{pk}_F)$ to $\mathcal{A}_{\mathsf{FS}}$.

(2) $\mathcal{A}_{\mathsf{FS}}$ adaptively chooses a function f and an input $x \in \mathcal{D}_f$ and then submits them to $\mathcal{C}_{\mathsf{FS}}$.

(3) $\mathcal{C}_{\mathsf{FS}}$ flips a coin $b \leftarrow \{0, 1\}$ and proceeds as follows:

(a) If $b = 0$, run $\widetilde{\mathrm{pk}_F} \leftarrow \mathsf{FS.RfPk}(\mathrm{pk}_F)$;

(b) Otherwise, compute $(\widetilde{\mathrm{pk}_F}, \mathrm{td}) \leftarrow \mathsf{FS.Sim}^1(\mathrm{pk}_F)$ where td denotes a trapdoor.

Next, $\mathcal{C}_{\mathsf{FS}}$ returns $(\widetilde{\mathrm{pk}_F}, y, \sigma_f)$ with the following function:

$$(y, \sigma_f) := \begin{cases} \mathsf{FS.Sign}\big(\widetilde{\mathrm{pk}_F}, \mathsf{FS.KGen}(\mathrm{sk}_F, f), f, x\big), & \text{if } b = 0, \\ \mathsf{FS.Sim}^2(\widetilde{\mathrm{pk}_F}, \mathrm{td}, f, x), & \text{otherwise.} \end{cases}$$

(4) Finally, $\mathcal{A}_{\mathsf{FS}}$ outputs a bit b'.

The advantage $\mathsf{Adv}_{\mathsf{FS}}^{\mathrm{unlink}}(\lambda)$ of $\mathcal{A}_{\mathsf{FS}}$ is defined as $|\Pr[b = b'] - 1/2|$.

Definition 6 (Unlinkability of FS)**.** We say that an FS scheme $\mathsf{FS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{RfPk}, \mathsf{Sign}, \mathsf{Vrfy})$ is unlinkable if $\mathsf{Adv}_{\mathsf{FS}}^{\mathrm{unlink}}(\lambda)$ is negligible.

Remark 2. Our model of unforgeability is a little different from that in [2]. In our model, $\mathcal{C}_{\mathsf{FS}}$ executes algorithm $\mathsf{FS.RfPk}$ honestly and "updates" the key $\widetilde{\mathrm{pk}_F}$ dynamically along with each session so that $\mathcal{A}_{\mathsf{FS}}$ cannot fool the verifier. This operation can be done with NIZK. In each session the verifier runs $\mathrm{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ correctly and appends crs to pk_F to "refresh" the public key. Without td with respect to crs, $\mathcal{A}_{\mathsf{FS}}$ cannot make a valid forgery. In contrast with Definition 5, $\mathcal{C}_{\mathsf{FS}}$ in Definition 6 is not required to produce $\widetilde{\mathrm{pk}_F}$ honestly. The main purpose of Definition 6 is to prevent the verifier from disclosing the cooperation intention of Alice to anyone else. In an unlinkable FS scheme no one can decide if the verifier "refreshed" the public key pk_F (w.r.t Alice) in the right way and got the real signature or not.

Below we introduce another security notion called accountability, taking inspiration from [6]. Informally, the accountability requires that the origin of the signing key sk_f should be undeniable when the master authority tries to frame an innocent assistant.

Accountability. Only when sk_f is a valid signing key for f under pk_F will algorithm $\mathsf{FS.Judge}(\mathrm{pk}_F, f, \mathrm{sk}_f)$ output 1. Consider the following game in which $\mathcal{C}_{\mathsf{FS}}$ is a challenger and $\mathcal{A}_{\mathsf{FS}}$ is a PPT adversary.

(1) $\mathcal{C}_{\mathsf{FS}}$ calls algorithm $\mathsf{FS.Setup}$ to get $(\mathrm{sk}_F, \mathrm{pk}_F)$ and sends pk_F to $\mathcal{A}_{\mathsf{FS}}$.

(2) $\mathcal{A}_{\mathsf{FS}}$ adaptively chooses f and submits it to oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$ to obtain sk_f.

• $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$: It takes as input a function f, and works as follows:

(a) If there is an entry (f, sk_f) in the history list, return sk_f directly;

(b) Otherwise, output a fresh $\mathrm{sk}_f \leftarrow \mathsf{FS.KGen}(\mathrm{sk}_F, f)$ and store (f, sk_f).

(3) Finally, $\mathcal{A}_{\mathsf{FS}}$ produces a pair (f^*, sk_{f^*}) and wins the game if it holds that

• $\mathsf{FS.Judge}(\mathrm{pk}_F, f^*, \mathrm{sk}_{f^*}) = 1$; and

• there is no item (f^*, sk_{f^*}) in the history list.

The advantage $\mathsf{Adv}^{\mathrm{acc}}_{\mathsf{FS}}(\lambda)$ of $\mathcal{A}_{\mathsf{FS}}$ is defined as the probability of winning the game.

Definition 7 (Accountability of FS)**.** An FS scheme $\mathsf{FS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{RfPk}, \mathsf{Sign}, \mathsf{Vrfy})$ is accountable if $\mathsf{Adv}^{\mathrm{acc}}_{\mathsf{FS}}(\lambda)$ is negligible.

Remark 3. The accountability guarantees that a PPT adversary cannot produce a valid signing key sk_{f^*} with respect to f^* without knowing sk_F. Actually, an unforgeable FS scheme is also accountable. Otherwise, one can use $\mathcal{A}^{\mathrm{acc}}_{\mathsf{FS}}$ (with respect to Definition 7) as a subroutine to obtain a forgery $(\widetilde{\mathrm{pk}^*_F}, y^*, \sigma^*_f)$ against the unforgeability of FS. Precisely, $\mathcal{A}^{\mathrm{uf}}_{\mathsf{FS}}$ (with respect to Definition 5) uses $\mathcal{A}^{\mathrm{acc}}_{\mathsf{FS}}$ to obtain sk_{f^*} and then uses sk_{f^*} to sign on $y^* = f^*(x)$.

5 Our first construction

In this section we propose a construction of FS employing RRS as a building block. The idea is as follows. To issue sk_f for $f \in \mathcal{F}_{\mathsf{FS}}$, the master authority calls algorithms $\mathsf{RRS.RrSk}$ and $\mathsf{RRS.RrPk}$ with δ to obtain a fresh $(\mathrm{sk}'_R, \mathrm{pk}'_R)$, computes σ_r on f with respect to pk'_R, and sets $\mathrm{sk}_f := (\mathrm{pk}'_R, \delta, \sigma_r)$. Only with the knowledge of δ can one prove that the function f is admitted.

5.1 The construction

Let $\mathsf{RRS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{RrSk}, \mathsf{RrPk})$ be an RRS scheme and $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{Check}, \mathsf{Sim}, \mathsf{Ext})$ be an NIZK system for the following NP language:

$$L = \big\{(\mathrm{pk}_R, \mathrm{pk}'_R) : \exists \delta \text{ s.t. } \mathrm{pk}'_R = \mathsf{RRS.RrPk}(\mathrm{pk}_R, \delta)\big\}.$$

Let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme and $\mathsf{Hash} : \{0,1\}^* \to \mathcal{M}_{\mathsf{RRS}}$ be a hash function. Our FS construction works as below.

$\mathsf{FS.Setup}$. It takes as input 1^λ, executes $(\mathrm{sk}_R, \mathrm{pk}_R) \leftarrow \mathsf{RRS.Setup}(1^\lambda)$ and outputs ($\mathrm{sk}_F = \mathrm{sk}_R$, $\mathrm{pk}_F = \mathrm{pk}_R$). For brevity, the following algorithms implicitly get pk_F as input.

$\mathsf{FS.KGen}$. It takes as input (sk_F, f), executes $(\mathrm{sk}_D, \mathrm{pk}_D) \leftarrow \mathsf{DS.Setup}(1^\lambda)$, picks $\delta \leftarrow \Delta_{\mathsf{RRS}}$, produces $\mathrm{sk}'_R \leftarrow \mathsf{RRS.RrSk}(\mathrm{sk}_R, \delta)$ and $\mathrm{pk}'_R \leftarrow \mathsf{RRS.RrPk}(\mathrm{pk}_R, \delta)$, computes $\sigma_r \leftarrow \mathsf{RRS.Sign}(\mathrm{sk}'_R, \mathsf{Hash}(f \| \mathrm{pk}_D))$ and outputs $\mathrm{sk}_f := (\mathrm{pk}'_R, \sigma_r, \delta, \mathrm{sk}_D, \mathrm{pk}_D)$.

$\mathsf{FS.RfPk}$. It takes as input pk_F, executes $\mathrm{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and outputs $\widetilde{\mathrm{pk}_F} := (\mathrm{pk}_F, \mathrm{crs})$. It adds the tuple $(\mathrm{pk}_F, \mathrm{crs})$ into a list $\mathcal{L}_{\mathrm{pk}_F}$.

$\mathsf{FS.Sign}$. It takes as input $(\widetilde{\mathrm{pk}_F}, \mathrm{sk}_f, f, x)$ and works as follows:

• Parse $\widetilde{\mathrm{pk}_F}$ as $(\mathrm{pk}_R, \mathrm{crs})$, and output $\perp$ directly if $\mathrm{pk}_R \neq \mathrm{pk}_F$;

• Parse sk_f as $(\mathrm{pk}'_R, \sigma_r, \delta, \mathrm{sk}_D, \mathrm{pk}_D)$, set $\iota := (\mathrm{pk}_R, \mathrm{pk}'_R)$ and $\omega := \delta$, compute $\pi \leftarrow \mathsf{NIZK.Prove}(\mathrm{crs}, \iota, \omega)$, generate $\sigma_d \leftarrow \mathsf{DS.Sign}(\mathrm{sk}_D, x)$, compute $y := f(x)$ and output $\sigma_f := (\mathrm{pk}'_R, f\|\mathrm{pk}_D, \sigma_r, \pi, x, \sigma_d)$.

$\mathsf{FS.Vrfy}$. It takes as input $(\mathrm{pk}_F, \widetilde{\mathrm{pk}_F} = (\mathrm{pk}_R, \mathrm{crs}), y, \sigma_f)$ and works as follows:

• Output 1 if

$$(\mathrm{pk}_F,\mathrm{crs}) \in \mathcal{L}_{\mathrm{pk}_F} \wedge \mathrm{pk}_R = \mathrm{pk}_F \wedge \mathsf{NIZK.Vrfy}\big(\mathrm{crs}, (\mathrm{pk}_R, \mathrm{pk}'_R), \pi\big) = 1 \wedge y = f(x)$$
$$\wedge\, \mathsf{DS.Vrfy}(\mathrm{pk}_D, x, \sigma_d) = 1 \wedge x \in \mathcal{D}_f \wedge \mathsf{RRS.Vrfy}\big(\mathrm{pk}'_R, \mathsf{Hash}(f\|\mathrm{pk}_D), \sigma_r\big) = 1;$$

• Otherwise, output 0.

Here we demonstrate how to instantiate the black-box construction with compatible modules. Fleischhacker et al. [6] showed that a slightly modified version of Schnorr's signature scheme satisfies the unforgeability of RRS in the random oracle model. Besides, it holds that $\mathrm{pk}'_R/\mathrm{pk}_R = g^\delta$ for this modified version. We can implement the NIZK system with Schnorr's protocol and Fiat-Shamir transform under the assumption that the discrete logarithm problem is hard.

The verifier can recover (f, x) from y if $y = f(x)$ is regarded as an unambiguous document (i.e., "Alice agrees to sell her fruit at a price of $\underline{x}$/kg" where '_' is a special identifier). Thanks to this key point, our first construction enjoys succinctness. A signature $\sigma_f = (\mathrm{pk}'_R, f\|\mathrm{pk}_D, \sigma_r, \pi, x, \sigma_d)$ can be compressed into $\sigma_f = (\mathrm{pk}'_R, \mathrm{pk}_D, \sigma_r, \pi, \sigma_d)$. If DS is a conventional signature scheme, e.g., [23], and RRS is the modified Schnorr's signature scheme [6], it holds that $|(\mathrm{pk}'_R\|\mathrm{pk}_D\|\sigma_r\|\sigma_d)| = \mathsf{poly}(\lambda)$. We can further replace NIZK with zk-SNARK [2] to make π succinct. Then the size of σ_f is independent of that of (f, x).

5.2 Security analysis

Theorem 1. If $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vrfy})$ is unforgeable, $\mathsf{RRS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{RrSk}, \mathsf{RrPk})$ is unforgeable, Hash is collision-resistant and $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{Check}, \mathsf{Sim}, \mathsf{Ext})$ is an adaptive NIZK system, our first construction of FS is unforgeable.

Proof. Denote $\mathcal{A}_{\mathsf{FS}}$'s final forgery by (y^*, σ_f^*) and suppose that $y^* = f^*(x^*)$. Consider the following two cases for $\mathcal{A}_{\mathsf{FS}}$'s attack.

Forge$_f$: $\mathcal{A}_{\mathsf{FS}}$ did not send a query f^* to oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$ nor a query $(f^*, \cdot)$ to oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$. In other words, there does not exist a record $(f, \cdot, \cdot)$ in the history list of oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ such that $f = f^*$.

Forge$_x$: $\mathcal{A}_{\mathsf{FS}}$ did not send a query f^* to oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$, and it did send a query (f, x) to oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ such that $f = f^*$ and $x \neq x^*$. In another word, there is at least one record $(f, \cdot, \cdot)$ in the history list of oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ such that $f = f^*$.

Below we show that $\mathcal{A}_{\mathsf{FS}}$ wins the game with negligible probability in both cases above.

Forge$_f$: If $\mathcal{A}_{\mathsf{FS}}$ is able to produce a valid forgery in this case, we can build an efficient algorithm $\mathcal{A}_{\mathsf{RRS}}$ to "break" the security of RRS.

(1) $\mathcal{A}_{\mathsf{RRS}}$ is given pk_R and access to oracles $\mathsf{RRS}.\mathcal{O}^1_{\mathsf{Sign}}$ and $\mathsf{RRS}.\mathcal{O}^2_{\mathsf{Sign}}$ and sets $\mathrm{pk}_F := \mathrm{pk}_R$.

(2) Then $\mathcal{A}_{\mathsf{RRS}}$ answers $\mathcal{A}_{\mathsf{FS}}$'s queries to oracles $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$ and $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ as follows:

• $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}(f)$: Given f, it runs $(\mathrm{sk}_D, \mathrm{pk}_D) \leftarrow \mathsf{DS.Setup}(1^\lambda)$, samples $\delta \leftarrow \Delta_{\mathsf{RRS}}$ uniformly at random, and submits $\big(\mathsf{Hash}(f\|\mathrm{pk}_D), \delta\big)$ to oracle $\mathsf{RRS}.\mathcal{O}^2_{\mathsf{Sign}}$ to get σ_r. Finally, it computes $\mathrm{pk}'_R \leftarrow \mathsf{RRS.RrPk}(\mathrm{pk}_R, \delta)$ and returns $\mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \mathrm{sk}_D, \mathrm{pk}_D)$.

• $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}(f, x)$: If there is no item (f, sk_f) in T_{FS}, it generates $\mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \mathrm{sk}_D, \mathrm{pk}_D)$ for f in the same way as in $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$. It runs $(\mathrm{crs}, \mathrm{td}) \leftarrow \mathsf{NIZK.Ext}^1(1^\lambda)$ to generate $\widetilde{\mathrm{pk}_F} = (\mathrm{pk}_F = \mathrm{pk}_R, \mathrm{crs})$ and computes $\pi \leftarrow \mathsf{NIZK.Prove}(\mathrm{crs}, (\mathrm{pk}_R, \mathrm{pk}'_R), \delta)$ and $\sigma_d \leftarrow \mathsf{DS.Sign}(\mathrm{sk}_D, x)$. Finally, it outputs $\sigma_f = (\mathrm{pk}'_R, f\|\mathrm{pk}_D, \sigma_r, \pi, x, \sigma_d)$ on $y = f(x)$ and $\widetilde{\mathrm{pk}_F}$, and inserts $(\mathrm{pk}_F, \mathrm{crs}, \mathrm{td})$ into $\mathcal{L}_{\mathrm{pk}_F}$.

(3) Given $(\widetilde{\mathrm{pk}^*_F} = (\mathrm{pk}_R, \mathrm{crs}^*), y^*, \sigma_f^* = (\mathrm{pk}^*_R, f^*\|\mathrm{pk}^*_D, \sigma^*_r, \pi^*, x^*, \sigma^*_d))$ from $\mathcal{A}_{\mathsf{FS}}$, $\mathcal{A}_{\mathsf{RRS}}$ computes $\delta^* \leftarrow \mathsf{NIZK.Ext}^2(\mathrm{crs}^*, \mathrm{td}^*, (\mathrm{pk}_R, \mathrm{pk}^*_R), \pi^*)$ where $(\mathrm{pk}_F, \mathrm{crs}^*, \mathrm{td}^*)$ is in $\mathcal{L}_{\mathrm{pk}_F}$ and submits the tuple $(m^* = \mathsf{Hash}(f^*\|\mathrm{pk}^*_D), \delta^*, \sigma^*_r)$ to its own challenger $\mathcal{C}_{\mathsf{RRS}}$.

Note that for any $f_0, f_1 \in \mathcal{F}_{\mathsf{FS}}$ it holds that $\mathsf{range}(f_0) \cap \mathsf{range}(f_1) = \emptyset$ and there does not exist (f, sk_f) in T_{FS} such that $y^* \in \mathsf{range}(f)$. We can deduce that $\mathcal{A}_{\mathsf{RRS}}$ did not issue sk_{f^*} for f^* and $m^* \notin Q_{\mathsf{RRS}}$. Moreover, guaranteed by the argument of knowledge of NIZK, it holds that δ^* is valid. If $\mathcal{A}_{\mathsf{FS}}$ wins the game with non-negligible probability, $\mathcal{A}_{\mathsf{RRS}}$ "breaks" the security of RRS with non-negligible probability as well, contradicting the assumption that scheme RRS is unforgeable. Therefore, $\mathcal{A}_{\mathsf{FS}}$ outputs a valid forgery with negligible probability in this case.

Forge$_x$: If $\mathcal{A}_{\mathsf{FS}}$ can produce a valid forgery in this case, we can build another efficient algorithm $\mathcal{A}_{\mathsf{DS}}$ to "break" the security of DS.

(1) $\mathcal{A}_{\mathsf{DS}}$ is given pk_D and access to oracle $\mathsf{DS}.\mathcal{O}_{\mathsf{Sign}}$. Let $\mathsf{poly}(\lambda)$ be a polynomial upper-bound on the number of signature queries made by $\mathcal{A}_{\mathsf{FS}}$. Then $\mathcal{A}_{\mathsf{DS}}$ picks $i \leftarrow [\mathsf{poly}(\lambda)]$ guessing that the i-th $\mathsf{FS.Sign}$ query is the first one that is related to f^*. $\mathcal{A}_{\mathsf{DS}}$ runs $(\mathrm{sk}_R, \mathrm{pk}_R) \leftarrow \mathsf{RRS.Setup}(1^\lambda)$ and $\mathrm{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$, and sets $\mathrm{pk}_F := \mathrm{pk}_R$ and $\widetilde{\mathrm{pk}_F} := (\mathrm{pk}_R, \mathrm{crs})$.

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

• $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}(f)$: $\mathcal{A}_{\mathsf{DS}}$ calls algorithm $\mathsf{DS.Setup}(1^\lambda)$ to obtain $(\widetilde{\mathrm{sk}_D}, \widetilde{\mathrm{pk}_D})$ and picks $\delta \leftarrow \Delta_{\mathsf{RRS}}$. Then $\mathcal{A}_{\mathsf{DS}}$ runs $\mathrm{sk}'_R \leftarrow \mathsf{RRS.RrSk}(\mathrm{sk}_R, \delta)$ and $\mathrm{pk}'_R \leftarrow \mathsf{RRS.RrPk}(\mathrm{pk}_R, \delta)$ respectively, and computes $\sigma_r \leftarrow \mathsf{RRS.Sign}(\mathrm{sk}'_R, \mathsf{Hash}(f \| \widetilde{\mathrm{pk}_D}))$. Finally, $\mathcal{A}_{\mathsf{DS}}$ outputs $\mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \widetilde{\mathrm{sk}_D}, \widetilde{\mathrm{pk}_D})$ for f.

• $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}(f, x)$: If the oracle has never been accessed before, $\mathcal{A}_{\mathsf{DS}}$ initializes $j := 1$; otherwise, $\mathcal{A}_{\mathsf{DS}}$ sets $j := j + 1$. $\mathcal{A}_{\mathsf{DS}}$ runs $\mathrm{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ to get $\widetilde{\mathrm{pk}_F}$ and inserts $(\mathrm{pk}_F, \mathrm{crs})$ into $\mathcal{L}_{\mathrm{pk}_F}$.

(a) If $j \neq i$ and there is $(f, \mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \widetilde{\mathrm{sk}_D}, \widetilde{\mathrm{pk}_D}))$ in T_{FS}, $\mathcal{A}_{\mathsf{DS}}$ produces σ_f on $y = f(x)$ using this sk_f.

(b) If $j \neq i$ and there is $(f, \mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \perp, \mathrm{pk}_D))$ in T_{FS}, $\mathcal{A}_{\mathsf{DS}}$ runs $\pi \leftarrow \mathsf{NIZK.Prove}(\mathrm{crs}, (\mathrm{pk}_R, \mathrm{pk}'_R), \delta)$ and submits x to its own oracle $\mathsf{DS}.\mathcal{O}_{\mathsf{Sign}}$ to obtain σ_d on x. Finally, $\mathcal{A}_{\mathsf{DS}}$ outputs $\sigma_f = (\mathrm{pk}'_R, f \| \mathrm{pk}_D, \sigma_r, \pi, x, \sigma_d)$.

(c) If $j \neq i$ and there does not exist (f, sk_f) in T_{FS}, $\mathcal{A}_{\mathsf{DS}}$ gains $\mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \widetilde{\mathrm{sk}_D}, \widetilde{\mathrm{pk}_D})$ for f through oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$, runs $\pi \leftarrow \mathsf{NIZK.Prove}(\mathrm{crs}, (\mathrm{pk}_R, \mathrm{pk}'_R), \delta)$ and $\sigma_d \leftarrow \mathsf{DS.Sign}(\widetilde{\mathrm{sk}_D}, x)$, returns $\sigma_f = (\mathrm{pk}'_R, f \| \widetilde{\mathrm{pk}_D}, \sigma_r, \pi, x, \sigma_d)$ as the answer and inserts (f, sk_f) into T_{FS}.

(d) If $j = i$ and there exists $(f, \mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \widetilde{\mathrm{sk}_D}, \widetilde{\mathrm{pk}_D}))$ in T_{FS}, $\mathcal{A}_{\mathsf{DS}}$ aborts the game. This event will not happen if $\mathcal{A}_{\mathsf{DS}}$ guesses i correctly. ($\mathcal{A}_{\mathsf{DS}}$ guessed that the i-th $\mathsf{FS.Sign}$ query is the first one related to f^*. If there is an item (f, sk_f) in T_{FS}, it means that $\mathcal{A}_{\mathsf{DS}}$ guessed wrong.)

(e) If $j = i$ and there is no $(f, \mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \widetilde{\mathrm{sk}_D}, \widetilde{\mathrm{pk}_D}))$ in T_{FS}, $\mathcal{A}_{\mathsf{DS}}$ chooses δ uniformly at random to generate sk'_R and pk'_R, computes $\sigma_r \leftarrow \mathsf{RRS.Sign}(\mathrm{sk}'_R, \mathsf{Hash}(f \| \mathrm{pk}_D))$ and adds $(f, \mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \perp, \mathrm{pk}_D))$ to T_{FS}, and produces σ_f on $y = f(x)$ as step 2(b).

(3) Given a forgery $(\widetilde{\mathrm{pk}^*_F} = (\mathrm{pk}_R, \mathrm{crs}^*), y^*, \sigma^*_f = (\mathrm{pk}^*_R, f^* \| \mathrm{pk}^*_D, \sigma^*_r, \pi^*, x^*, \sigma^*_d))$ from $\mathcal{A}_{\mathsf{FS}}$, $\mathcal{A}_{\mathsf{DS}}$ sends (x^*, σ^*_d) to its own challenger $\mathcal{C}_{\mathsf{DS}}$.

Notice that for any $f_0, f_1 \in \mathcal{F}_{\mathsf{FS}}$ it holds that $\mathsf{range}(f_0) \cap \mathsf{range}(f_1) = \emptyset$ and for any $f \in \mathcal{F}_{\mathsf{FS}}$ it holds that f is an injective function. We can deduce that $\mathcal{C}_{\mathsf{DS}}$ did not produce σ_d on $x^* = f^{*-1}(y^*)$. If $\mathcal{A}_{\mathsf{FS}}$ wins the game with non-negligible probability, $\mathcal{A}_{\mathsf{DS}}$ "breaks" the security of DS with non-negligible probability as well, contradicting the assumption that scheme DS is unforgeable. Therefore, $\mathcal{A}_{\mathsf{FS}}$ outputs a valid forgery with negligible probability in this case.

Theorem 2. If $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{Check}, \mathsf{Sim}, \mathsf{Ext})$ is an adaptive NIZK system capturing zero-knowledge property, our first construction of FS is unlinkable.

Proof. Here we describe how the simulator $\mathsf{FS.Sim} = (\mathsf{Sim}^1, \mathsf{Sim}^2)$ works in detail.

$\mathsf{FS.Sim}^1$. This algorithm gets as input pk_F, works as below and outputs $(\widetilde{\mathrm{pk}_F}, \mathrm{td})$:

• Compute $(\mathrm{crs}_s, \mathrm{td}_s) \leftarrow \mathsf{NIZK.Sim}^1(1^\lambda)$;

• Set $\widetilde{\mathrm{pk}_F} := (\mathrm{pk}_R, \mathrm{crs}_s)$ and $\mathrm{td} := \mathrm{td}_s$.

$\mathsf{FS.Sim}^2$. This algorithm gets as input $(\widetilde{\mathrm{pk}_F}, \mathrm{td}, f, x)$, works as below and outputs (y, σ_f):

• Run $(\mathrm{sk}_D, \mathrm{pk}_D) \leftarrow \mathsf{DS.Setup}(1^\lambda)$ and compute $\sigma_d \leftarrow \mathsf{DS.Sign}(\mathrm{sk}_D, x)$, run $(\mathrm{sk}''_R, \mathrm{pk}''_R) \leftarrow \mathsf{RRS.Setup}(1^\lambda)$ and compute $\sigma_r \leftarrow \mathsf{RRS.Sign}(\mathrm{sk}''_R, f \| \mathrm{pk}_D)$;

• Produce $\pi_s \leftarrow \mathsf{NIZK.Sim}^2(\mathrm{crs}_s, \mathrm{td}_s, (\mathrm{pk}_R, \mathrm{pk}''_R))$;

• Compute $y := f(x)$ and set $\sigma_f := (\mathrm{pk}''_R, f \| \mathrm{pk}_D, \sigma_r, \pi_s, x, \sigma_d)$.

Guaranteed by the correctness of RRS, we have $\mathrm{pk}''_R \overset{c}{\approx} \mathrm{pk}'_R$. Furthermore, guaranteed by the zero-knowledge property of NIZK, we have $\mathrm{crs}_s \overset{c}{\approx} \mathrm{crs}$ (which implies that the simulated refreshing public key is indistinguishable from the real refreshing public key) and $\pi_s \overset{c}{\approx} \pi$. It is obvious that the simulated functional signature and the real functional signature are computationally indistinguishable.

Theorem 3. If $\mathsf{RRS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{RrSk}, \mathsf{RrPk})$ is unforgeable and Hash is collision-resistant, our first construction of FS is accountable.

Proof. Here we only demonstrate how to construct the algorithm $\mathsf{FS.Judge}$ (c.f. Remark 3).

$\mathsf{FS.Judge}$. This algorithm takes $(\mathrm{pk}_F = (\mathrm{pk}_R, \mathrm{crs}), f, \mathrm{sk}_f = (\mathrm{pk}'_R, \sigma_r, \delta, \mathrm{sk}_D, \mathrm{pk}_D))$ and outputs a bit b.

- Set $b = 1$ if it holds that $\mathsf{RRS.RrPk}(\mathrm{pk}_R, \delta) = \mathrm{pk}'_R \wedge \mathsf{RRS.Vrfy}\big(\mathrm{pk}'_R, \mathsf{Hash}(f\|\mathrm{pk}_D), \sigma_r\big) = 1$;
- Set $b = 0$ otherwise.

6 Our second construction

Below we introduce another construction of FS taking inspiration from Guillou-Quisquater protocol [8]. The idea is as follows. The master authority uses its private key to produce an RSA signature σ_1 on f and passes it to the assistant. Then the assistant generates a proof π (which is viewed as σ_f) to persuade the verifier to believe that it does know a valid RSA signature on f. Guaranteed by the argument of knowledge (c.f. Definition 3) of NIZK, a PPT adversary cannot produce a valid forgery.

6.1 The construction

Let $\mathsf{Hash}_1 : \{0,1\}^* \to \mathbb{Z}_e$ and $\mathsf{Hash}_2, \mathsf{Hash}_3 : \{0,1\}^* \to \mathbb{Z}_N^*$ be three hash functions. Denote the setup algorithm of the RSA assumption [7] by $\mathsf{RSA.Setup}(1^\lambda) \to (N, e, d)$. Our second construction of FS works as below.

$\mathsf{FS.Setup}$. It takes as input 1^λ, runs $(N, e, d) \leftarrow \mathsf{RSA.Setup}(1^\lambda)$ and outputs $\mathrm{sk}_F = d$ and $\mathrm{pk}_F = (N, e, \mathsf{Hash}_2, \mathsf{Hash}_3)$.

$\mathsf{FS.KGen}$. It takes as input (sk_F, f), runs $(\widetilde{N}, \widetilde{e}, \widetilde{d}) \leftarrow \mathsf{RSA.Setup}(1^\lambda)$, sets $\sigma_1 := \mathsf{Hash}_2(\widetilde{N}\|\widetilde{e}\|f)^d \bmod N$, and outputs $\mathrm{sk}_f = (\widetilde{N}, \widetilde{e}, \widetilde{d}, \sigma_1)$.

$\mathsf{FS.RfPk}$. It takes as input pk_F and outputs $\widetilde{\mathrm{pk}_F} = (N, e, \mathsf{Hash}_1^{(s)})$ where $\mathsf{Hash}_1^{(s)}$ is the hash function programmed by the verifier in the session s. It adds $(\mathrm{pk}_F, \widetilde{\mathrm{pk}_F})$ into a list $\mathcal{L}_{\mathrm{pk}_F}$.

$\mathsf{FS.Sign}$. It takes as input $(\widetilde{\mathrm{pk}_F} = (N, e, \mathsf{Hash}_1^{(s)}), \mathrm{sk}_f = (\widetilde{N}, \widetilde{e}, \widetilde{d}, \sigma_1), f, x)$, runs $\sigma_2 := \mathsf{Hash}_3(x)^{\widetilde{d}} \bmod \widetilde{N}$, chooses $\widetilde{\pi_1} \leftarrow \mathbb{Z}_N^*$ uniformly at random, sets $\pi_1 := \widetilde{\pi_1}^e \bmod N$ and $\pi_2 := \mathsf{Hash}_1^{(s)}(N\|e\|\pi_1\|\widetilde{N}\|\widetilde{e}\|f\|x)$ and $\pi_3 := \sigma_1^{\pi_2} \cdot \widetilde{\pi_1} \bmod N$, and outputs $\sigma_f := (x, \sigma_2, \widetilde{N}, \widetilde{e}, f, \pi_1, \pi_3)$.

$\mathsf{FS.Vrfy}$. It takes as input $(\mathrm{pk}_F, \widetilde{\mathrm{pk}_F}, y, \sigma_f)$, computes $\pi_2 := \mathsf{Hash}_1^{(s)}(N\|e\|\pi_1\|\widetilde{N}\|\widetilde{e}\|f\|x)$, and

- outputs $b = 1$ if it holds that $\mathsf{Hash}_3(x) = \sigma_2^{\widetilde{e}} \bmod \widetilde{N} \wedge \pi_3^e = \mathsf{Hash}_2(\widetilde{N}\|\widetilde{e}\|f)^{\pi_2} \cdot \pi_1$;
- outputs $b = 0$ otherwise.

Similar to the first construction, $\sigma_f = (x, \sigma_2, \widetilde{N}, \widetilde{e}, f, \pi_1, \pi_3)$ in the second construction can be compressed into $\sigma_f = (\sigma_2, \widetilde{N}, \widetilde{e}, \pi_1, \pi_3)$ and it holds that $|\sigma_f| = \mathsf{poly}(\lambda)$.

6.2 Security analysis

Theorem 4. If the RSA problem is hard relative to algorithm $\mathsf{RSA.Setup}(1^\lambda)$, our second construction of FS is unforgeable in the random oracle model.

Proof. Assume that $y^* = f^*(x^*)$ and $\sigma_f^* = (x^*, \sigma_2^*, \widetilde{N}^*, \widetilde{e}^*, f^*, \pi_1^*, \pi_3^*)$. There are two cases for $\mathcal{A}_{\mathsf{FS}}$'s attack.

Forge$_f$: $\mathcal{A}_{\mathsf{FS}}$ outputs a forgery with respect to f^*. In another word, $\mathcal{A}_{\mathsf{FS}}$ did not send sent a query of the form $(f^*, \cdot)$ to oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ where the notation "$\cdot$" denotes a wildcard.

Forge$_x$: $\mathcal{A}_{\mathsf{FS}}$ outputs a forgery with respect to x^*. In other words, $\mathcal{A}_{\mathsf{FS}}$ did not send a query (f^*, x^*) to oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$, and there exists at least one record (f, x, σ_f) in the history list of $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ such that $f = f^*$ and $x \neq x^*$.

The proof is similar to the proof of Theorem 1. Below we only show that in both cases $\mathcal{A}_{\mathsf{FS}}$ wins the game with negligible probability.

Forge$_f$: If $\mathcal{A}_{\mathsf{FS}}$ outputs a valid forgery with overwhelming probability, we can build an efficient algorithm $\mathcal{A}_{\mathsf{RSA}}$ to solve the RSA problem with respect to $\mathsf{RSA.Setup}(1^\lambda)$.

(1) $\mathcal{A}_{\mathsf{RSA}}$ is given $(N_{\mathsf{RSA}}, e_{\mathsf{RSA}}, Y_{\mathsf{RSA}})$, and sets $\mathrm{pk}_F := (N_{\mathsf{RSA}}, e_{\mathsf{RSA}}, \mathsf{Hash}_2, \mathsf{Hash}_3)$. Suppose that there are $\mathsf{poly}_0(\lambda)$ records in the history list of $\mathcal{O}_{\mathsf{Hash}_2}$. $\mathcal{A}_{\mathsf{RSA}}$ picks $i_0 \leftarrow [\mathsf{poly}_0(\lambda)]$ uniformly at random guessing that the i_0-th Hash_2 record is about $\widetilde{N}^*\|\widetilde{e}^*\|f^*$.

(2) $\mathcal{A}_{\mathsf{RSA}}$ answers $\mathcal{A}_{\mathsf{FS}}$'s queries to the oracles $\mathcal{O}_{\mathsf{Hash}_1^{(s)}}$, $\mathcal{O}_{\mathsf{Hash}_2}$, $\mathcal{O}_{\mathsf{Hash}_3}$, $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$, and $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ as follows. To simplify the matters, here we suppose that $\mathcal{A}_{\mathsf{FS}}$ never submits the same query twice.

- $\mathcal{O}_{\mathsf{Hash}_1^{(s)}}(N_{\mathsf{RSA}}\|e_{\mathsf{RSA}}\|\pi_1\|\widetilde{N}\|\widetilde{e}\|f\|x)$: $\mathcal{A}_{\mathsf{RSA}}$ samples $\pi_2 \leftarrow \mathbb{Z}_{e_{\mathsf{RSA}}}$ and returns π_2 directly.
- $\mathcal{O}_{\mathsf{Hash}_2}(\widetilde{N}\|\widetilde{e}\|f)$: If this oracle has not been accessed before, $\mathcal{A}_{\mathsf{RSA}}$ initializes $j_0 := 1$; otherwise, $\mathcal{A}_{\mathsf{RSA}}$ sets $j_0 := j_0 + 1$.

(a) If $j_0 = i_0$, $\mathcal{A}_{\mathsf{RSA}}$ stores an item $(\widetilde{N}\|\widetilde{e}\|f, \perp, Y_{\mathsf{RSA}})$ and returns Y_{RSA}.

(b) Otherwise, $\mathcal{A}_{\mathsf{RSA}}$ picks $\sigma_1 \leftarrow \mathbb{Z}_N^*$ uniformly at random, computes $Y := \sigma_1^{e_{\mathsf{RSA}}} \bmod N_{\mathsf{RSA}}$, adds an item $(\widetilde{N}\|\widetilde{e}\|f, \sigma_1, Y)$ to the history list, and returns Y.

• $\mathcal{O}_{\mathsf{Hash}_3}(x, \widetilde{N}\|\widetilde{e})$: $\mathcal{A}_{\mathsf{RSA}}$ directly outputs $h_x \leftarrow \mathbb{Z}_{\widetilde{N}}^*$.

• $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}(f)$: $\mathcal{A}_{\mathsf{RSA}}$ computes $(\widetilde{N}, \widetilde{e}, \widetilde{d}) \leftarrow \mathsf{RSA.Setup}(1^\lambda)$, submits $\widetilde{N}\|\widetilde{e}\|f$ to oracle $\mathcal{O}_{\mathsf{Hash}_2}$ to get $(\widetilde{N}\|\widetilde{e}\|f, \sigma_1, Y)$, and outputs $\mathrm{sk}_f = (\widetilde{N}, \widetilde{e}, \widetilde{d}, \sigma_1)$.

• $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}(f, x)$: $\mathcal{A}_{\mathsf{RSA}}$ sets $\widetilde{\mathrm{pk}_F} := (N_{\mathsf{RSA}}, e_{\mathsf{RSA}}, \mathsf{Hash}_1^{(s)})$ and inserts it into $\mathcal{L}_{\mathrm{pk}_F}$. $\mathcal{A}_{\mathsf{RSA}}$ submits f to oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$ to obtain $\mathrm{sk}_f = (\widetilde{N}, \widetilde{e}, \widetilde{d}, \sigma_1)$, computes $\sigma_2 := \mathcal{O}_{\mathsf{Hash}_3}(x, \widetilde{N}\|\widetilde{e})^{\widetilde{d}} \bmod \widetilde{N}$, chooses $\widetilde{\pi_1} \leftarrow \mathbb{Z}_{N_{\mathsf{RSA}}}^*$, computes $\pi_1 := \widetilde{\pi_1}^{e_{\mathsf{RSA}}} \bmod N_{\mathsf{RSA}}$, submits $N_{\mathsf{RSA}}\|e_{\mathsf{RSA}}\|\pi_1\|\widetilde{N}\|\widetilde{e}\|f\|x$ to oracle $\mathcal{O}_{\mathsf{Hash}_1^{(s)}}$ to obtain π_2, computes $\pi_3 := \sigma_1^{\pi_2} \cdot \widetilde{\pi_1} \bmod N_{\mathsf{RSA}}$, and outputs $y = f(x)$ and $\sigma_f = (x, \sigma_2, \widetilde{N}, \widetilde{e}, f, \pi_1, \pi_3)$.

(3) $\mathcal{A}_{\mathsf{FS}}$ outputs a forgery $(\widetilde{\mathrm{pk}_F^*} = (N_{\mathsf{RSA}}, e_{\mathsf{RSA}}, \mathsf{Hash}_1^*), y^*, \sigma_f^* = (x^*, \sigma_2^*, \widetilde{N}^*, \widetilde{e}^*, f^*, \pi_1^*, \pi_3^*))$ where it holds that $(N_{\mathsf{RSA}}, e_{\mathsf{RSA}}, \mathsf{Hash}_1^*)$ is in $\mathcal{L}_{\mathrm{pk}_F}$ and $\mathsf{Hash}_1^*(N_{\mathsf{RSA}}\|e_{\mathsf{RSA}}\|\pi_1^*\|\widetilde{N}^*\|\widetilde{e}^*\|f^*\|x^*) = \pi_2^*$. Then $\mathcal{A}_{\mathsf{RSA}}$ reprograms $\mathsf{Hash}_1^*(N_{\mathsf{RSA}}\|e_{\mathsf{RSA}}\|\pi_1^*\|\widetilde{N}^*\|\widetilde{e}^*\|f^*\|x^*) = \pi_2^{**}$ and rewinds $\mathcal{A}_{\mathsf{FS}}$ to get $(\widetilde{\mathrm{pk}_F^*}, y^*, \sigma_f^* = (x^*, \sigma_2^*, \widetilde{N}^*, \widetilde{e}^*, f^*, \pi_1^{**}, \pi_3^{**}))$. It holds that $(\pi_3^*/\pi_3^{**})^{e_{\mathsf{RSA}}} = (Y_{\mathsf{RSA}})^{\pi_2^* - \pi_2^{**}}$. $\mathcal{A}_{\mathsf{RSA}}$ computes $X_{\mathsf{RSA}} = Y_{\mathsf{RSA}}^U \cdot (\pi_3^*/\pi_3^{**})^V$ where $U \cdot e_{\mathsf{RSA}} + V \cdot (\pi_2^* - \pi_2^{**}) = \gcd(e_{\mathsf{RSA}}, \pi_2^* - \pi_2^{**}) = 1$[1)].

Guaranteed by the RSA assumption, it holds that

$$\Pr[\mathbf{Forge}_f : \mathcal{A}_{\mathsf{FS}} \text{ wins}]/\mathsf{poly}_0(\lambda) \leqslant \Pr[\mathcal{A}_{\mathsf{RSA}} \text{ wins}] \leqslant \mathsf{negl}(\lambda),$$

and we can further deduce that $\Pr[\mathbf{Forge}_f : \mathcal{A}_{\mathsf{FS}} \text{ wins}] \leqslant \mathsf{negl}(\lambda)$.

$\mathbf{Forge}_x$: If $\mathcal{A}_{\mathsf{FS}}$ outputs a valid forgery with non-negligible probability, we can build an efficient algorithm $\mathcal{A}_{\mathsf{RSA}}$ to solve the RSA problem with respect to $\mathsf{RSA.Setup}(1^\lambda)$.

(1) $\mathcal{A}_{\mathsf{RSA}}$ is given $(N_{\mathsf{RSA}}, e_{\mathsf{RSA}}, Y_{\mathsf{RSA}})$, runs $(N, e, d) \leftarrow \mathsf{RSA.Setup}(1^\lambda)$, and sets $\mathrm{sk}_F := d$ and $\mathrm{pk}_F := (N, e, \mathsf{Hash}_2, \mathsf{Hash}_3)$. Assume that the oracle $\mathcal{O}_{\mathsf{Hash}_3}$ can be accessed at most $\mathsf{poly}_1(\lambda)$ times. $\mathcal{A}_{\mathsf{RSA}}$ picks $i_1 \leftarrow [\mathsf{poly}_1(\lambda)]$ guessing that the i_1-th Hash_3 query is the first Hash_3 query related to x^*. Assume that the oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ can be accessed at most $\mathsf{poly}_2(\lambda)$ times. $\mathcal{A}_{\mathsf{RSA}}$ picks $i_2 \leftarrow [\mathsf{poly}_2(\lambda)]$ guessing that the i_2-th $\mathsf{FS.Sign}$ query is the first $\mathsf{FS.Sign}$ query related to f^*.

(2) $\mathcal{A}_{\mathsf{RSA}}$ answers $\mathcal{A}_{\mathsf{FS}}$'s queries to the oracles $\mathcal{O}_{\mathsf{Hash}_1^{(s)}}$, $\mathcal{O}_{\mathsf{Hash}_2}$, $\mathcal{O}_{\mathsf{Hash}_3}$, $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$, and $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}$ as follows.

• $\mathcal{O}_{\mathsf{Hash}_1^{(s)}}(N\|e\|\pi_1\|\widetilde{N}\|\widetilde{e}\|f\|x)$: $\mathcal{A}_{\mathsf{RSA}}$ picks $\pi_2 \leftarrow \mathbb{Z}_e$ uniformly at random and outputs π_2.

• $\mathcal{O}_{\mathsf{Hash}_2}(\widetilde{N}\|\widetilde{e}\|f)$: $\mathcal{A}_{\mathsf{RSA}}$ picks $h_f \leftarrow \mathbb{Z}_N^*$ uniformly at random and outputs h_f.

• $\mathcal{O}_{\mathsf{Hash}_3}(x, \widetilde{N}\|\widetilde{e})$: If this oracle has not been accessed, $\mathcal{A}_{\mathsf{RSA}}$ initializes $j_1 := 1$; otherwise, $\mathcal{A}_{\mathsf{RSA}}$ sets $j_1 := j_1 + 1$.

(a) If $j_1 = i_1$ and $(\widetilde{N}, \widetilde{e}) \neq (N_{\mathsf{RSA}}, e_{\mathsf{RSA}})$, $\mathcal{A}_{\mathsf{RSA}}$ aborts the game. If $\mathcal{A}_{\mathsf{RSA}}$ gets the right i_1, this event will not happen and it implies that $x = x^*$ and $\mathsf{Hash}_3(x) = Y_{\mathsf{RSA}}$.

(b) If $j_1 = i_1$ and $(\tilde{N}, \tilde{e}) = (N_{\mathsf{RSA}}, e_{\mathsf{RSA}})$, $\mathcal{A}_{\mathsf{RSA}}$ inserts an item $((x, N_{\mathsf{RSA}}\|e_{\mathsf{RSA}}), \perp, Y_{\mathsf{RSA}})$ into the history list and outputs Y_{RSA}.

(c) Otherwise, $\mathcal{A}_{\mathsf{RSA}}$ chooses $\sigma_2 \leftarrow \mathbb{Z}_{\widetilde{N}}^*$, computes $Y := \sigma_2^{\widetilde{e}} \bmod \widetilde{N}$, outputs Y and inserts an item $((x, \widetilde{N}\|\widetilde{e}), \sigma_2, Y)$ into the history list.

• $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}(f)$: If there is $(f, \perp)$ in T_{FS}, $\mathcal{A}_{\mathsf{RSA}}$ aborts the game; otherwise, $\mathcal{A}_{\mathsf{RSA}}$ runs $(\widetilde{N}, \widetilde{e}, \widetilde{d}) \leftarrow \mathsf{RSA.Setup}(1^\lambda)$, computes $h_f := \mathcal{O}_{\mathsf{Hash}_2}(\widetilde{N}\|\widetilde{e}\|f)$ and $\sigma_1 := h_f^d \bmod N$, adds (f, sk_f) to T_{FS} and outputs $\mathrm{sk}_f = (\widetilde{N}, \widetilde{e}, \widetilde{d}, \sigma_1)$.

• $\mathsf{FS}.\mathcal{O}_{\mathsf{Sign}}(f, x)$: If this oracle has not been accessed before, $\mathcal{A}_{\mathsf{RSA}}$ initializes $j_2 := 1$; otherwise, $\mathcal{A}_{\mathsf{RSA}}$ sets $j_2 := j_2 + 1$. $\mathcal{A}_{\mathsf{RSA}}$ sets $\widetilde{\mathrm{pk}_F} := (N, e, \mathsf{Hash}_1^{(s)})$ and inserts it into $\mathcal{L}_{\mathrm{pk}_F}$.

(a) If $j_2 = i_2$ and there is (f, sk_f) in T_{FS}, $\mathcal{A}_{\mathsf{RSA}}$ aborts the game. This event will not happen if $\mathcal{A}_{\mathsf{RSA}}$ gets the right i_2.

(b) If $j_2 = i_2$ and there is no (f, sk_f) in T_{FS}, $\mathcal{A}_{\mathsf{RSA}}$ submits $(x, N_{\mathsf{RSA}}\|e_{\mathsf{RSA}})$ to oracle $\mathcal{O}_{\mathsf{Hash}_3}$ to get $((x, N_{\mathsf{RSA}}\|e_{\mathsf{RSA}}), \sigma_2, Y)$, submits $N_{\mathsf{RSA}}\|e_{\mathsf{RSA}}\|f$ to oracle $\mathcal{O}_{\mathsf{Hash}_2}$ to get h_f, computes $\sigma_1 := h_f^d \bmod N$, picks $\widetilde{\pi_1} \leftarrow \mathbb{Z}_N^*$, computes $\pi_1 := \widetilde{\pi_1}^e \bmod N$, submits $N\|e\|\pi_1\|N_{\mathsf{RSA}}\|e_{\mathsf{RSA}}\|f\|x$ to oracle $\mathcal{O}_{\mathsf{Hash}_1^{(s)}}$ to get

1) **Lemma.** Given N, elements $y', y \in \mathbb{Z}_N^*$ and integers e, e' for which it holds that $\gcd(|e|, |e'|) = 1$ and $y'^e = y^{e'}$, an eth root of y can be computed in polynomial time [23].

π_2, computes $\pi_3 := \sigma_1^{\pi_2} \cdot \widetilde{\pi_1} \bmod N$, outputs $\sigma_f = (x, \sigma_2, N_{\mathsf{RSA}}, e_{\mathsf{RSA}}, f, \pi_1, \pi_3)$ and adds $(f, \perp)$ to the history list of $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$.

(c) If $j_2 > i_2$ and there is $(f, \perp)$ in the history list, $\mathcal{A}_{\mathsf{RSA}}$ submits $(x, N_{\mathsf{RSA}} \| e_{\mathsf{RSA}})$ to oracle $\mathcal{O}_{\mathsf{Hash}_3}$ to get $((x, N_{\mathsf{RSA}} \| e_{\mathsf{RSA}}), \sigma_2, Y)$. Then $\mathcal{A}_{\mathsf{RSA}}$ picks $\widetilde{\pi_1} \leftarrow \mathbb{Z}_N^*$, computes $\pi_1 := \widetilde{\pi_1}^e \bmod N$, submits $N \| e \| \pi_1 \| N_{\mathsf{RSA}} \| e_{\mathsf{RSA}} \| f \| x$ to oracle $\mathcal{O}_{\mathsf{Hash}_1^{(s)}}$ to get π_2, and sets $\pi_3 := \sigma_1^{\pi_2} \cdot \widetilde{\pi_1} \bmod N$ where $\sigma_1 = (\mathcal{O}_{\mathsf{Hash}_2}(N_{\mathsf{RSA}} \| e_{\mathsf{RSA}} \| f))^d$. $\mathcal{A}_{\mathsf{RSA}}$ outputs $\sigma_f = (x, \sigma_2, N_{\mathsf{RSA}}, e_{\mathsf{RSA}}, f, \pi_1, \pi_3)$.

(d) Otherwise, $\mathcal{A}_{\mathsf{RSA}}$ submits f to oracle $\mathsf{FS}.\mathcal{O}_{\mathsf{KGen}}$ to get $\mathrm{sk}_f = (\widetilde{N}, \widetilde{e}, \widetilde{d}, \sigma_1)$, submits $(x, \widetilde{N} \| \widetilde{e})$ to $\mathcal{O}_{\mathsf{Hash}_3}$ to get $((x, \widetilde{N} \| \widetilde{e}), \sigma_2, Y)$. Then $\mathcal{A}_{\mathsf{RSA}}$ chooses $\widetilde{\pi_1} \leftarrow \mathbb{Z}_N^*$, computes $\pi_1 := \widetilde{\pi_1}^e \bmod N$, submits $N \| e \| \pi_1 \| N_{\mathsf{RSA}} \| e_{\mathsf{RSA}} \| f \| x$ to oracle $\mathcal{O}_{\mathsf{Hash}_1^{(s)}}$ to get π_2, and sets $\pi_3 := \sigma_1^{\pi_2} \cdot \widetilde{\pi_1} \bmod N$. Finally, $\mathcal{A}_{\mathsf{RSA}}$ outputs $\sigma_f = (x, \sigma_2, \widetilde{N}, \widetilde{e}, f, \pi_1, \pi_3)$.

(3) Given $(\widetilde{\mathrm{pk}_F^*}, y^*, \sigma_f^* = (x^*, \sigma_2^*, N_{\mathsf{RSA}}, e_{\mathsf{RSA}}, f^*, \pi_1^*, \pi_3^*))$, $\mathcal{A}_{\mathsf{RSA}}$ sends $X_{\mathsf{RSA}} = \sigma_2^*$ to its own challenger.

Guaranteed by the RSA assumption, it holds that

$$\Pr[\mathbf{Forge}_x : \mathcal{A}_{\mathsf{FS}} \text{ wins}]/(\mathsf{poly}_1(\lambda) \cdot \mathsf{poly}_2(\lambda)) \leqslant \Pr[\mathcal{A}_{\mathsf{RSA}} \text{ wins}] \leqslant \mathsf{negl}(\lambda),$$

and we can further deduce that $\Pr[\mathbf{Forge}_x : \mathcal{A}_{\mathsf{FS}} \text{ wins}] \leqslant \mathsf{negl}(\lambda)$.

Theorem 5. If the RSA problem is hard relative to algorithm $\mathsf{RSA.Setup}(1^\lambda)$, our second construction of FS is unlinkable in the random oracle model.

Proof. Here we show how to construct the simulator $\mathsf{FS.Sim} = (\mathsf{Sim}^1, \mathsf{Sim}^2)$ in details.

$\mathsf{FS.Sim}^1$. This algorithm takes as input pk_F and returns $(\widetilde{\mathrm{pk}_F}, \mathrm{td})$ as follows:

- Parse pk_F as (N, e);
- Set the reprogrammability of $\mathsf{Hash}_1^{(s)}$ as the trapdoor td;
- Set $\widetilde{\mathrm{pk}_F} := (N, e, \mathsf{Hash}_1^{(s)})$.

Specially, the verifier enjoys the reprogrammability of $\mathsf{Hash}_1^{(s)}$ since $\mathsf{Hash}_1^{(s)}$ is appointed by the verifier.

$\mathsf{FS.Sim}^2$. This algorithm takes as input $(\widetilde{\mathrm{pk}_F}, \mathrm{td}, f, x)$ and returns (y, σ_f) as follows:

- Compute $(\widetilde{N}, \widetilde{e}, \widetilde{d}) \leftarrow \mathsf{RSA.Setup}(1^\lambda)$;
- Compute $\sigma_2 := \mathsf{Hash}_3(x)^{\widetilde{d}} \bmod \widetilde{N}$, choose $\pi_3 \leftarrow \mathbb{Z}_N^*$ and $\pi_2 \leftarrow \mathbb{Z}_e$ and compute $\pi_1 := \pi_3^e \cdot \mathsf{Hash}_2(\widetilde{N} \| \widetilde{e} \| f)^{-\pi_2}$;
- Reprogram $\mathsf{Hash}_1^{(s)}$ such that $\mathsf{Hash}_1^{(s)}(N \| e \| \pi_1 \| \widetilde{N} \| \widetilde{e} \| f \| x) = \pi_2$;
- $y := f(x)$ and $\sigma_f := (x, \sigma_2, \widetilde{N}, \widetilde{e}, f, \pi_1, \pi_3)$.

It is obvious that $\mathsf{FS.Vrfy}(\mathrm{pk}_F, \widetilde{\mathrm{pk}_F}, y, \sigma_f) = 1$. The distribution of π_3 is uniform over $\mathbb{Z}_N^*$ in both the real transcript and the simulated transcript. Furthermore, the distribution of π_1 is uniform over $\mathbb{Z}_N^*$ in both the real transcript and the simulated transcript. We can conclude that the simulated transcript and the real transcript are computationally indistinguishable.

Theorem 6. If the RSA problem is hard relative to the algorithm $\mathsf{RSA.Setup}(1^\lambda)$, our second construction of FS is accountable in the random oracle model.

Proof. Here we demonstrate how to build the algorithm $\mathsf{FS.Judge}$.

$\mathsf{FS.Judge}$. This algorithm, on input $(\mathrm{pk}_F = (N, e), f, \mathrm{sk}_f = (\widetilde{N}, \widetilde{e}, \widetilde{d}, \sigma_1))$, outputs a bit b as follows:

- Output 1 if it holds that $\sigma_1^e = \mathsf{Hash}_2(\widetilde{N} \| \widetilde{e} \| f) \bmod N$;
- Output 0 otherwise.

7 Conclusion

In this paper, we modified the definition of FS [2] slightly to support applications where an authorization letter is necessary. Specifically, we focus on a function family $\mathcal{F}_{\mathsf{FS}}$ for which there is no such a pair (f_0, x_0, f_1, x_1) that $f_0(x_0) = f_1(x_1)$. We revised the security models of FS accordingly and introduced two additional security notions called unlinkability and accountability. Compared with the original FS, signatures σ_f in our definition do not expose the intention of the master authority. Therefore, with our revised primitive an agency is able to negotiate with the verifier on behalf of the master authority and simultaneously guarantees the benefit of the master authority. We proposed two constructions of FS and proved them to be secure.

In our schemes, the verifier must maintain a stateful list $\mathcal{L}_{\mathrm{pk}_F}$ and the assistant cannot sign on messages until the verifier runs the algorithm RfPk once. In the future work, we will explore how to use designated-verifier signature (DVS) [24] to enhance our scheme. Besides, we will explore how to combine FS with the fair exchange protocols/concurrent signatures [3–5] exactly to build a powerful e-commerce system.

Acknowledgements This work was supported by Major Program of Guangdong Basic and Applied Research (Grant No. 2019B03-0302008), National Natural Science Foundation of China (Grant Nos. 61872152, 61872409), and Science and Technology Program of Guangzhou (Grant No. 201902010081). The authors are grateful to the anonymous reviewers for their invaluable comments.

References

1 Wei L, Zhu H, Cao Z, et al. Security and privacy for storage and computation in cloud computing. Inf Sci, 2014, 258: 371–386
2 Boyle E, Goldwasser S, Ivan I. Functional signatures and pseudorandom functions. In: Proceedings of International Workshop on Public Key Cryptography, Buenos Aires, 2014. 501–519
3 Chen L, Kudla C, Paterson K G. Concurrent signatures. In: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, 2004. 287–305
4 Guo Q W, Cui Y Z, Zou X M, et al. Generic construction of privacy-preserving optimistic fair exchange protocols. J Int Serv Inf Secur, 2017, 7: 44–56
5 Huang Q, Yang G M, Wong D S, et al. Ambiguous optimistic fair exchange: definition and constructions. Theory Comput Sci, 2015, 562: 177–193
6 Fleischhacker N, Krupp J, Malavolta G, et al. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In: Proceedings of International Workshop on Public Key Cryptography, Taipei, 2016. 301–330
7 Katz J, Lindell Y. Introduction to Modern Cryptography. Boca Raton: CRC Press, 2014
8 Guillou L C, Quisquater J J. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In: Proceedings of Conference on the Theory and Application of Cryptography, Santa Barbara, 1990. 216–231
9 Attrapadung N, Libert B, Peters T. Efficient completely context-hiding quotable and linearly homomorphic signatures. In: Proceedings of International Workshop on Public Key Cryptography, Nara, 2013. 386–404
10 Catalano D, Fiore D, Warinschi B. Homomorphic signatures with efficient verification for polynomial functions. In: Proceedings of Annual Cryptology Conference, Santa Barbara, 2014. 371–389
11 Freeman D M. Improved security for linearly homomorphic signatures: a generic framework. In: Proceedings of International Workshop on Public Key Cryptography, Darmstadt, 2012. 697–714
12 Libert B, Peters T, Joye M, et al. Linearly homomorphic structure-preserving signatures and their applications. Des Codes Cryptogr, 2015, 77: 441–477
13 Ma J H, Liu J H, Wu W, et al. Survey on redactable signatures. J Comput Res Dev, 2017, 54: 2144–2152
14 Johnson R, Molnar D, Song D, et al. Homomorphic signature schemes. In: Proceedings of Cryptographers Track at the RSA Conference, San Jose, 2002. 244–262
15 Camenisch J, Derler D, Krenn S, et al. Chameleon-hashes with ephemeral trapdoors and applications to invisible sanitizable signatures. In: Proceedings of International Workshop on Public Key Cryptography, Amsterdam, 2017. 152–182
16 Kim S, Park S, Won D. Proxy signatures, revisited. In: Proceedings of International Conference on Information and Communications Security, Beijing, 1997. 223–232
17 Yu Y, Mu Y, Susilo W, et al. Provably secure proxy signature scheme from factorization. Math Comput Model, 2012, 55: 1160–1168
18 Demirel D, Schabhüser L, Buchmann J. Privately and Publicly Verifiable Computing Techniques — A Survey. Berlin: Springer, 2017
19 Backes M, Meiser S, Schröder D. Delegatable functional signatures. In: Proceedings of International Workshop on Public Key Cryptography, Taipei, 2016. 357–386
20 Li S, Liang B, Xue R. Private functional signatures: definition and construction. In: Proceedings of Australasian Conference on Information Security and Privacy, Wollongong, 2018. 284–303
21 Guo Q W, Huang Q, Yang G M. Authorized function homomorphic signature. Comput J, 2018, 61: 1897–1908
22 Goldreich O. Foundations of Cryptography: Volume 1, Basic Tools. Cambridge: Cambridge University Press, 2007
23 Katz J. Digital Signatures. Berlin: Springer, 2010
24 Huang Q, Yang G M, Wong D S, et al. Efficient strong designated verifier signature schemes without random oracle or with non-delegatability. Int J Inf Secur, 2011, 10: 373–385